1

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

PRECISIONSM RESOLUTION TARGETS

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

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

nity [5, 12, 8], the changes may not be fully implemented for several years. The Ontario Telepresence Project (OTP) environment has already incorporated ATM into its teleconferencing environment without waiting for these standards to be developed.

Advances in video compression techniques have introduced devices that allow video signals to be transmitted across non-dedicated, standard telephone lines. It is common for audio and video devices, such as cameras and VCRs, to have an interface that allows them to be controlled directly by computer workstations through serial or parallel port connections.

These advances in technology make possible a low-cost, video conferencing system among sites equipped with audio and video devices under the control of a workstation. Software that allows people separated by distance to co-operate on a shared task can be integrated with software that controls the physical devices in a conference to provide a cohesive set of logical services to the user. Such a system would allow a user to control resources within the context of a conference despite physical location or implementation. All software components would be informed of events during the conference, allowing a co-ordinated response of all components involved in the conference.

Users would be provided with an abstract set of services that would be translated into low-level resource control commands, and would be shielded from the low-level details of controlling physical devices and logical resources within the conference. Standardizing the services would also allow conferences to be held among a number of conference rooms without prior exchange of specialized software.

Although a number of architectures for similar video conferencing environments are under development, the emphasis of these projects is on managing and controlling data streams among conferencing sites, rather than on providing an integrated software architecture for controlling physical devices, logical resources, and telecommunications connections.

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

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 0-315-98594-1

Canada

Name David B. Brown

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.

Philosophy, Literature

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

PHILOSOPHY, RELIGION AND THEOLOGY

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

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Management 0385
Marketing 0501
Canadian Studies 0503
Economics 0505
General 0508
Agricultural 0509
Commerce-Business 0510
Finance 0511
History 0558
Labor 0366
Theory 0351
Folklore 0578
Geography 0578
Gerontology 0578
History 0578
General

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
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 0615
General 0616
International Law and Relations 0617
Public Administration 0814
Recreation 0452
Social Work 0626
Sociology 0627
General 0938
Criminology and Penology 0631
Demography 0628
Ethnic and Racial Studies 0629
Individual and Family Studies 0630
Industrial and Labor Relations 0700
Public and Social Welfare 0344
Social Structure and Development 0709
Theory and Methods 0999
Transportation 0453
Urban and Regional Planning 0453
Women's Studies

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

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

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0358
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

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

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
Chemistry 0485
General 0749
Agricultural 0486
Analytical 0487
Biochemistry 0488
Inorganic 0738
Nuclear 0490
Organic 0491
Pharmaceutical 0494
Physical 0495
Polymer 0754
Radiation 0405
Mathematics 0605
Physics 0986
General 0606
Acoustics 0608
Astronomy and Astrophysics 0718
Atmospheric Science 0607
Atomic 0798
Electronics and Electricity 0759
Elementary Particles and High Energy 0609
Fluid and Plasma 0610
Molecular 0752
Nuclear 0756
Optics 0611
Radiation 0463
Solid State
Statistics
Applied Sciences
Applied Mechanics 0346
Computer Science 0984

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

PSYCHOLOGY

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

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

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

A Software Architecture for Telepresence Conference Control

submitted by Gerald L. Boersma, B.Eng., in partial fulfillment of the requirements for the degree of Master of Engineering.

Chair,
Department of Systems and
Computer Engineering

Dr. Gerald Karam,
Thesis Supervisor

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
December 2, 1994

Abstract

This thesis develops a flexible software architecture which provides a cohesive multipoint video conferencing service among a set of sites connected by a computer network. Physical resources, such as audio and video devices located at a site, and logical resources, such as a distributed database, may be used within the conference. Users are provided with a set of logical services that provide control over physical devices and logical resources within the context of the conference despite the physical location or implementation. The architecture supports dynamic configuration: resources can be added to and removed from the system without affecting conferences in progress.

A concern in the development of such an architecture is the inter-operability of software components. This thesis uses an emerging standard for conference control to define a set of services to the user and the corresponding network protocol to provide these services.

This architecture has been designed, implemented, and tested for the Ontario Telepresence Project laboratory.

Acknowledgements

I would like to thank Alex Hubbard and the other research engineers of the Department of Systems and Computer Engineering for their help, without which the implementation of the software for this thesis would have taken much longer. I would also like to thank the engineers of the Ontario Telepresence Project, in particular Chris Passier and Dominic Richens, for providing a listening ear and helpful suggestions while I bounced ideas off them.

I would next like to thank Gerald Karam, my thesis supervisor, for not only providing guidance during my research, but also providing me with access to the resources of the Ontario Telepresence Project for my research.

I would like to also thank my family and friends for their support. Thank you Annemarie, Eric, and Tim for reading the final draft of my thesis, providing encouragement and insight during my research, and listening to the stories of my successes and failures. I would also like to thank my father for his support as to my decision to complete this degree.

Finally, I would like to thank NSERC and the Department of Systems and Computer Engineering for the funding they provided.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Thesis Objectives	4
1.3	Thesis Overview	6
2	Background	8
2.1	Related Work	8
2.1.1	Software Architectures	10
2.1.2	Services	13
2.1.3	Protocols	16
2.2	Ontario Telepresence Project	19
2.2.1	Hardware	20
2.2.2	Software	21
2.3	Multipoint Communications Service	23
2.4	Network Services Infrastructure	25
2.4.1	Communication Services	25
2.4.2	Distributed Resource Control	27
2.4.3	Remote Device Control	27
2.5	Generic Conference Control	28
2.5.1	Architecture	28
2.5.2	Services	29
2.5.3	Protocol	31
3	System Architecture	33
3.1	Conferences within OTP	33
3.2	System Goals	35
3.3	Definition of a Conference	36
3.4	System Representation	37
3.4.1	OOSE	37

3.4.2	Machine Charts	38
3.4.3	Timethreads	38
3.5	Requirements Model	40
3.5.1	System Context	40
3.5.2	System Functionality	43
3.6	Analysis Model	47
3.6.1	Subsystems	47
3.6.2	Communication between Components	51
3.7	System Architecture	52
3.7.1	System Components	52
3.7.2	Features of the Architecture	57
3.7.3	Fulfillment of Requirements	58
3.7.4	Goals of Other Architectures	59
4	Design and Implementation	62
4.1	Design Model	62
4.1.1	GCC Server	62
4.1.2	GCC Services and Protocol Module	65
4.1.3	Site Controller	68
4.1.4	Resource Controller	70
4.2	Implementation Environment	72
4.3	Implementation of the Design Model	73
4.3.1	Inter-Process Communication	73
4.3.2	Format of Messages	73
4.3.3	Processing of Events	74
4.3.4	Messages from Clients	75
4.3.5	Specification of Event Behaviour	75
4.3.6	Event-Driven Software Model	76
4.4	Implementation Model	79
4.4.1	Threading Classes	79
4.4.2	Frameworks	82
4.4.3	Components	82
4.4.4	Modules	82
5	Testbed	84
5.1	Configuration of Testbed	84
5.1.1	Hardware	84
5.1.2	Software	86
5.2	Test Scenarios	88

5.2.1	Meta-Conference Management	88
5.2.2	Conference Management Policy	90
5.2.3	Device Access Policy	91
5.2.4	Flexibility of Resource Controllers	94
6	Conclusions	95
6.1	Research Summary	95
6.2	Future Work	99
A	Node Controller User's Guide	106
B	Resource Controller User's Guide	117
C	GCC Server User's Guide	122

List of Figures

2.1	Architecture of MMCC	12
2.2	Architecture of CCCP	13
2.3	NSI Architecture	26
2.4	GCC Conferencing Terminal as Defined by ITU	30
3.1	OOSE Analysis Objects	38
3.2	Machine Chart Notation	39
3.3	Thread Notation	40
3.4	Requirements Model	41
3.5	Analysis Model	49
3.6	Participant Requests List of Conferences	53
3.7	Convener Requests to Create Conference	54
3.8	System Architecture	55
4.1	High-Level Design Model	63
4.2	Design of GCC Server	64
4.3	GCC Services and Protocol Module	66
4.4	Design of Site Controller	69
4.5	Design of the Resource Controller	71
4.6	Model for Event-Driven Software	77
4.7	Event Handling for the GCC Server	78
4.8	Class Library and Modules	80
5.1	Hardware Configuration of the Testbed	85
5.2	Software Configuration of the Testbed	87

List of Acronyms

A/V	Audio/Video
API	Application Programmer's Interface
ATM	Asynchronous Transfer Mode
CCP	Conference Control Protocol
CCCP	Conference Control Channel Protocol
CODEC	COder/DECoder
DAN	Desktop Area Network
DRC	Distributed Resource Controller
GCC	Generic Conference Control
GUI	Graphical User Interface
IETF	Internet Engineering Task Force
IIF	Integrated Interactive Intermedia Facility
IP	Internet Protocol
IPC	Inter-Process Communication
ISI	Information Sciences Institute
ITU	International Telecommunications Union
LAN	Local Area Network
LDC	Logical Device Controller
LLNL	Lawrence Livermore National Library
MBone	Multicast Backbone
MCS	Multipoint Communications Service
MICE	Multimedia Interactive Collaboration for European researchers
MMCC	Multiparty Multimedia Conference Control
MMuSiC	Multiparty Multimedia Session Control
MTP	Multicast Transport Protocol
NSI	Network Services Infrastructure
NTP	Network Time Protocol
OOSE	Object-Oriented Software Engineering
OTP	Ontario Telepresence Project

PDC	Physical Device Controller
PDU	Protocol Data Unit
RPC	Remote Procedure Call
RSVP	Resource ReSerVation Protocol
ST	Stream Protocol
TCC	Tele .esence Conference Control
TCL	Tool Command Language
TCP	Transmission Control Protocol

Chapter 1

Introduction

1.1 Motivation

Video conferencing enables group communication among people separated by distance. Although video conferencing environments have been designed and implemented since the mid-1970s [27], these facilities were expensive to build and maintain. Although reduction in travel expenses and savings in travel time has been a much stated justification for the expense of these video conferencing systems [26], studies in the sociological aspects of video conferencing show that there is great benefit in allowing people to conduct meetings from within their familiar environment [11].

Early video conferencing environments consisted of a specially designed conferencing site with several audio and video devices such as cameras, speakers, and microphones. The video and audio signals were shipped across dedicated, wideband telecommunications connections, and were quite expensive. In some cases, the technology was too complex for the end-users, and required technicians to operate the system on behalf of the users. As a result, there were only a few dozen installations in place by 1980 [26].

These video conferencing environments were not integrated. A user operated

each device through its own hardware interface, and a burden was placed on users to be familiar with the operation of each specific device. The devices that a user could control were limited to those within their physical reach. This approach was not flexible, since a user's interaction with a device was limited to its interface.

Recent advances in video and audio technology have made possible low-cost video conferencing environments that are directly accessible by the user. Advances in computer networks permit a software architecture that presents users with a set of co-ordinated, logical services to the user for controlling devices and shared software within a video conference context.

Computer networks, such as the Internet, provide a way for computers to ship data among sites connected to the network. The Internet does not presently have the bandwidth capabilities, however, to support instantaneous, full-motion video. There is much research in the area of protocols for the Internet that would support the shipping of real-time video data, assuming that the bandwidth required is available. However, given the size and complexity of the Internet, such standards will be under discussion and the services defined by the standards will not be available across the entire Internet for a number of years. Public-access networks also introduce the issues of reliability and security of data transmitted across the network. However, the Internet is ideal for low-bandwidth data transmission which is typical for out-of-band control messages.

Using current telecommunications technology rather than the Internet for shipping video signals allows a video conferencing system to be implemented using present technology, rather than waiting for new standards to be applied to computer networks. Such an approach takes better advantage of emerging improvements in telecommunication bandwidth.

For example, Asynchronous Transfer Mode (ATM) technology is emerging as a new standard for providing bandwidth to applications. Although ATM support in the Internet Protocol is a subject of continuing research in the Internet commu-

nity [5, 12, 8], the changes may not be fully implemented for several years. The Ontario Telepresence Project (OTP) environment has already incorporated ATM into its teleconferencing environment without waiting for these standards to be developed.

Advances in video compression techniques have introduced devices that allow video signals to be transmitted across non-dedicated, standard telephone lines. It is common for audio and video devices, such as cameras and VCRs, to have an interface that allows them to be controlled directly by computer workstations through serial or parallel port connections.

These advances in technology make possible a low-cost, video conferencing system among sites equipped with audio and video devices under the control of a workstation. Software that allows people separated by distance to co-operate on a shared task can be integrated with software that controls the physical devices in a conference to provide a cohesive set of logical services to the user. Such a system would allow a user to control resources within the context of a conference despite physical location or implementation. All software components would be informed of events during the conference, allowing a co-ordinated response of all components involved in the conference.

Users would be provided with an abstract set of services that would be translated into low-level resource control commands, and would be shielded from the low-level details of controlling physical devices and logical resources within the conference. Standardizing the services would also allow conferences to be held among a number of conference rooms without prior exchange of specialized software.

Although a number of architectures for similar video conferencing environments are under development, the emphasis of these projects is on managing and controlling data streams among conferencing sites, rather than on providing an integrated software architecture for controlling physical devices, logical resources, and telecommunications connections.

1.2 Thesis Objectives

The objective of this thesis is to develop a software architecture, referred to as Telepresence Conference Control (TCC), which provides a set of logical services to control physical devices and shared software in a video conference context among any number of sites with workstations connected by a computer network. Any technique can be used to ship audio, video, and application data among sites. Each physical device is connected to a workstation, and workstations within a site are connected by a Local Area Network (LAN). Any number of similarly equipped sites, connected by a computer network such as the Internet, are able to hold any number of concurrent conferences with other sites. Each site may have any number of audio and video sources using any type of communications link for audio, video, and application data; therefore, each site may be involved in a number of conferences at the same time.

A site is a set of audio and video sources and sinks treated as a group. Either a formal definition, such as official organizational definitions, or a less formal definition, such as a group of people that sit in the same area and normally work together, may be used to define a site.

The software architecture presents a set of abstract services to the users at a site which control the state of conferences in which a site participates. It also provides a set of abstract services to control devices within other remote sites as well as the local one. The software architecture is flexible enough to allow devices to be added and removed from it without disturbing conferences in progress.

The architecture also provides the mechanism for conferencing among any sites and accommodates different conference policies. Conference sites are not required to install a different software architecture for each type of conference in which they participate. Inter-operability among different implementations of conferencing software is facilitated by globally-accepted standards, which define a standardized set of

services and network protocol. The architecture uses the Generic Conference Control standard, as defined by the standardization committee of the ITU, to support inter-operability among software components located at different sites.

A server running on a UNIX workstation at each site provides the set of abstract services for managing a conference at the site. Clients of this server may be any platform which supports TCP/IP, such as Macintosh or DOS-based machines with proprietary software. This client-server architecture provides the flexibility of allowing clients to attach and detach at will, without affecting the current state of conferences as remembered by the server.

Peer-level conference control servers co-operate to establish conferences among sites. The conference control service is thus distributed among the sites participating in the conference.

The conference control architecture is applied to the Ontario Telepresence Project (OTP). Telepresence combines telecommunication and computing technology to support a sense of social proximity among co-workers, despite separation by space and time [14, 39]. The OTP laboratory consists of a number of physically-separated sites. Each site is equipped with several video sources and sinks, which are dynamically connected with a configurable A/V PBX-like switch, and allow for concurrent point-to-point, analog connections within the site. Video and audio mixers allow for multi-point connections among different parties. Workstations at each site are connected by a LAN. Devices are connected to workstations, and device control is distributed among the workstations within the site.

Sites use any type of connection to carry video and audio signals. Some examples of possible connections are non-dedicated telephone lines, along which a video coder/decoder (codec) at one site sends a digitally encoded signal to a codec at another site, or analog baseband connections. OTP is currently using ATM to ship video and audio data across dedicated digital connections between a site located in Ottawa and a site located in Toronto. Audio and video mixers support multi-party

connections; although these connections can support any number of sources, practical technical constraints limit them to four sources. Each site also has at least one workstation connected to the Internet.

Software tools developed at OTP are used to implement the architecture. The Network Services Infrastructure (NSI) [34], a toolkit for developing distributed applications within the OTP environment, provides services such as global naming, message ordering, and access to resources to OTP applications. The conference control architecture uses NSI services for communication among distributed conference control servers.

A testbed which consists of three sites in the OTP domain is used to verify the conference control architecture. These sites enter and leave a number of concurrent conferences and switch among the different conferences. Each site controls devices located at other remote sites, and sites concurrently access devices within the context of different conferences. This test scenario and others show that TCC meets the requirements for a conference control architecture.

1.3 Thesis Overview

Chapter two gives the background for the development of TCC. A number of conference control systems are described, with emphasis on their strengths and weaknesses as related to conference control. The services of GCC are also described. The OTP environment, with emphasis on existing software, is described in more detail. Relevant software components, such as the NSI, are also discussed.

Chapter three gives a high-level description of the requirements for a conference control system at OTP. It describes the services that users at OTP require. It also describes the software architecture that TCC provides for implementing resource control within OTP, such as device drivers.

Chapter four documents the implementation of TCC. It describes the software

processes, components, and techniques used in the deployment of the system.

Chapter five describes the testbed used to verify the operation of TCC at the OTP laboratory. Some observations of the behaviour of the final system as it relates to the original requirements are also made.

Chapter six summarizes the thesis objectives of a conference control software architecture and shows how they have been fulfilled with the deployment of TCC. Directions for future work are also provided.

Chapter 2

Background

The emphasis of TCC on the development of a flexible conference control architecture, rather than on video compression methods or data transfer protocols, is explained. Several existing conference control architectures are summarized, with emphasis on services and protocols. The GCC standard is summarized with a description of how it solves issues common to conferencing control. The NSI services are also described.

2.1 Related Work

Much of the current research regarding video and teleconferencing concentrates on protocols for the transfer of real-time, wide bandwidth data across the Internet.

The SStream protocol (ST) provides performance guarantees to real-time data through the use of resource reservation and multicast communication [16]. The Resource ReSeRVation P-otocol (RSVP) allows a user to reserve a minimum bandwidth at Internet routers between a source and destination [54].

A number of applications demonstrate application-level protocols for data transfer using the Internet protocol. The LBL Visual Audio Tool (vat) [33], the Inria Video conferencing System (ivs) [53], and the LBL Shared Whiteboard (wb) [32]

use the Network Voice Protocol and the Packet Video Protocol for communicating audio [15] and video [17] data.

However, these services are not concerned with issues related to conference control. The Multiparty Multimedia Session Control (MMuSiC) Internet working group addresses session control functionality within the Internet paradigm [30]. The goal of the MMuSiC working group is to provide a software infrastructure for controlling audio, video, and groupware applications within a conference. However, each application uses the Internet exclusively for data transfer, where the experimental Internet services provide for the timely shipping of audio and video data between sites.

The nature of the Internet also has an effect on the resulting session control functionality. Issues of robustness, heterogeneity, and scalability of a teleconferencing architecture over the Internet are primary concerns in the development of the session control functionality [18]. A number of conferencing systems which use the Internet for shipping session control messages and for audio and video data, using the experimental Internet protocols for real-time data, are described with emphasis on issues related to conference control.

ISI Multimedia Conference Control

The Multimedia Conference Control (MMCC) project, from the Information Sciences Institute (ISI) at the University of Southern California, provides an application interface to an experimental packet-switched teleconferencing system [48, 49]. Since the Internet does not yet have the hardware to support the bandwidth required for video packet transmission, it uses the Distributed Simulation Network (DSInet), a dedicated, wideband network isolated from the rest of the Internet which uses satellite connections to provide the bandwidth for the transmission of video signals.

BBN VideoTeam

VideoTeam, a product developed by a BBN Systems and Technologies, provides an environment for conferences to be held between participants in remotely located conference rooms [22, 23]. It also uses the packet-switched DSI-net for communication between video, audio, and groupware applications. The architecture emphasizes reliability issues related to the communication network, which reflects its commercial implementation.

MICE

The goal of the Multimedia Interactive Collaboration for European Researchers (MICE) project, from the University College of London, is to provide a general architecture for conference control that can support any type of floor control (such as which site is currently conducting a conference and how it requests conductorship) and session management policy, any independently developed application, any number of conference participants, and any type of end-system [29]. The project is also implemented in the Internet domain.

2.1.1 Software Architectures

Each of these projects adopt a similar model for providing conference control services to the user. Session managers at sites which participate in a conference communicate with session managers at other sites using a distributed messaging service, and co-ordinate their response to events in the conference. This type of communication is a *horizontal* conference control protocol.

The session managers may also communicate with local instances of applications to co-ordinate the response of all applications at a site. This type of communication is a *vertical* conference control protocol.

The user requests conference control services from the session manager at a site.

The user interacts directly with each instance of an application at the site to use the services of the application, which communicate with peer-level instances of the application at other sites using the communications services provided.

ISI Multimedia Conference Control

Figure 2.1 shows the architecture of MMCC. The responsibilities of MMCC can be divided into three levels: connection control, configuration control, and the user interface. The connection control layer provides a set of services to the user for session control and encapsulates the peer-level MMCC protocol. The Configuration Control layer manages devices and media at a local site, and requests local audio, video, and groupware agents to provide media streams consistent with the agent type. The user interface presents a Graphical User Interface (GUI) to the user for accessing services provided by MMCC. The horizontal conference control protocol is the Conference Control Protocol (CCP).

BBN VideoTeam

The BBN architecture consists of a peer-level conference managers which communicate with the “Sticky” conference control protocol as the horizontal protocol. VideoTeam does not implement a vertical conference control protocol; audio, video, and groupware applications at a site are not co-ordinated within the conference.

MICE

The architecture of the MICE project is shown in figure 2.2. Peer-level session and floor controllers communicate on the conference control channel using the Conference Control Channel Protocol (CCCP). The session controller encapsulates the session control policies for a conference, and presents a consistent user interface for conference control services. The floor controller implements a different floor control

Figure 2.1: Architecture of MMCC

policy for a conference into a separate entity. Applications communicate directly using the communication services provided, and receive messages from the conference control channel.

Figure 2.2: Architecture of CCCP

2.1.2 Services

The services provided by a session manager fall into several categories:

- *Meta-conference management services* advertise conferences in progress, start and terminate conferences, and invite sites to participate in a conference.
- *Membership control services* take care of who is currently in a conference and controls access to applications participating in a conference.
- *Application control services* take care of starting and stopping applications within a particular conference.
- *Floor management services* determine which users have control over particular applications within the context of the conference.
- *Network management services* establish the network connections between sites participating in a conference with the available data communication services. It is also concerned with connectivity of the network.

Implementations of conference control services differ because of differences in teleconferencing policy. Some of the current research concentrates on defining a set of services that provide a mechanism for implementing any type of teleconferencing policy. One of the activities of the MMuSiC working group is the extracting of a set of general services based on particular conference control environments.

ISI Multimedia Conference Control

The services of MMCC are aimed exclusively towards conference setup and termination. A participant can join a conference on invitation only. When a conference is established, the conference convener must specify the complete list of participants to be contacted. Only the participants that can be reached within a specified time period participate in the conference. A mechanism for automatically accepting invitations to participate in conferences without intervention by the user, referred to as an "autopilot mode", is also provided.

A specific floor control policy, or a mechanism for implementing any type of floor policy, is not implemented in MMCC. Meta-conference management services are not provided. A site knows about a conference only through invitation. MMCC only supports conferences which are closed, in which the membership of the conference must be predetermined before the conference begins. Therefore, MMCC dictates a specific conference policy for all participants, without allowing control by the participants over the policy.

BBN Videoteam

The services that VideoTeam offers are limited to two areas: it manages a conference as a whole by administering lists of conference participants, and it manages media resources, which are streams of real-time audio and video data transmitted among participants. Services for meta-conference management and floor control are not provided. To participate in a conference, a site must direct a request to a site already in the conference, or be invited to participate by another site.

MICE

The MICE project concentrates on providing an infrastructure for communication between entities within a conference control architecture. A specific floor control policy or conference management policy is not specified, since these are supplied by user-defined session and floor controllers. The services that the Conference Control Channel Protocol (CCCP) provides are analogous to a broadcast bus between the session and floor controllers and media agents. All messages appear at each component, which then filters out the messages that are directed to that component.

Although CCCP does provide a mechanism used to group applications into logical groups, it does not support the organization of these groups into conferences. Support for the abstraction of groups into conferences must be implemented in the session controller.

2.1.3 Protocols

The primary concern of a horizontal conference control protocol is to guarantee that each site has the correct information to be able to co-ordinate services properly with other sites within a conference.

A conference control protocol may take two approaches in providing this guarantee. A “tight” conference control protocol forces each site to have the same view of a global conference state. A “loose” conference control protocol does not assume the same view of a global site across all sites in a group. Each site may have its own view of the global state which likely differs with another site’s view, or each state maintains a limited subset of information about contacting other sites in the conference and does not attempt to construct a global view of the state of the conference.

A “tight” conference control protocol may use a multicast or broadcast transmission service, which sends a message from one site to a group of receiving sites. Single-source message ordering guarantees that if a single source site sends messages to a group, all destination sites receive the messages from that site in the same order. Multiple-source message ordering guarantees that if two source sites send a message to a group, all destination sites get the messages from the sources in the same order [28]. Multiple-source message ordering guarantees that each destination site in a group will have the same state as all other destination sites after receiving and processing the messages.

If message delivery is not ordered, a conference control protocol must guarantee that sites will have the same view of the global conference state after receiving messages in a different order. One approach is to have each sender timestamp each message with the time of a globally synchronized clock. Receivers would then order the messages according to the timestamp. However, providing a synchronized clock among a set of distributed processes is a non-trivial problem [36].

The Multicast Backbone (MBone) provides a virtual multicast network to the user by seamlessly integrating local networks, such as Ethernet, and existing Internet

routers, which may or may not support IP multicast, to provide a multicast service which spans the existing Internet [38]. IP multicast modifies Internet routers to provide a multicast service which manages dynamic groups and minimizes the number of packets sent [20, 21]. Although IP multicast supports single source message ordering, it does not support multiple source message ordering. Multicast Transport Protocol (MTP) is a flow-controlled, atomic multicasting transport protocol which provides a standardized set of services for different multicast service providers, such as IP multicast [7, 6].

Although a “tight” conference control protocol guarantees the same view of a global state once all sites receive the same message, differing transmission delays between the sending site and each receiving site cause the view of the global state to be different while transmissions of a message is in progress.

ISI Multimedia Conference Control

CCP is a “tight” conference control protocol that synchronizes the views of the global state among conference sites by transmitting the complete state of the conference, including a complete list of participants, and a globally synchronized timestamp with every protocol message. Therefore, message ordering is translated into a global clock synchronization problem, where clock synchronization services are provided by the Internet Network Time Protocol (NTP) [40].

The protocol uses an unordered group messaging service, such as IP multicast. Destination sites use the timestamp to order messages, which requires each destination site to buffer incoming messages and delay processing to account for transmission delays between sites. The protocol is also transaction-oriented, where all messages are of a send-reply nature. Timeouts for replies are dynamically calculated and updated to reflect the transmission delay between connection managers.

If sites do not agree on the current state of a conference, sites contact each other to negotiate and correct the state according to a synchronization algorithm. The

state synchronization algorithm does not scale to an arbitrary number of sites, since a site must contact every other site to correct its view of the state of the conference.

BBN Videoteam

The “Sticky” protocol is a loose conference control protocol which assumes that the same view of a global conference state across sites within a group is not possible because of transmission delays and unreliable communications between sites. Each site within a conference communicates with only one other site during the course of the conference; no multicast or broadcast communications mechanism is used. The protocol is transaction-based and messages contain a locally unique transaction identifier. If the connection to a site is lost, other sites within the conference are not affected.

MICE

All messages in CCCP consist of a source tuple, a list of destination tuples, and a function with a parameter list. Each tuple includes an address, the type, and an instantiation or a unique identifier for the tuple. Applications receive a CCCP message and decide whether to process it based on the list of destination tuples.

CCCP uses a multicast service, such as IP multicast, to transmit messages from the sender to the receivers. Although CCCP provides single source message ordering, it does not provide multiple source message ordering at receiving entities, instead specifying that message ordering is an application-level responsibility. Globally synchronized clocks, similar to the approach with the CCP protocol, are recommended for multiple source ordering.

The type of the application is specified using a class-hierarchy approach, for example, ‘floor.manager.holder’ or ‘video.send’. Therefore, different implementations of applications may process messages intended for the same class of services. Security is provided by access lists, which specify which hosts can invoke an application,

and by encryption of all messages.

2.2 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. Each user has a media space, where a media space is defined as

An electronic setting in which groups of people can work together, even when they are not resident in the same place or present at the same time. In a media space, people can create real-time visual and acoustic environments that span physically separate areas. They can also control the recording, accessing and replaying of images and sounds from those environments [11].

The concept of telepresence was developed by Xerox PARC [50]. Bellcore's Cruiser project [25, 35] and Xerox EuroPARC's RAVE project are implementations of telepresence.

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 [42]. Recent advances in this technology have made low-cost video conferencing services available, and have expanded it beyond specially equipped conference rooms. The decreasing cost of video technology makes it affordable to bring one or more cameras, monitors, microphones, and speakers to each person's work area which defines their media space.

The Ontario Telepresence Project (OTP) has further developed the concept of telepresence by developing a number of products. OTP is not only concerned with the technical issues of implementing the required technology, but also investigate the social issues of people using the technology in their work environment. Telepresence

demonstrates the social and technical aspects of applying technology to interactions between collaborators [47, 41, 14].

2.2.1 Hardware

OTP spans two locations: University of Toronto and Carleton University. The technology is also deployed at Xerox PARC, Xerox Rochester, and ITRC in Toronto. ITRC and the OTP location in Toronto are connected by a baseband, analog connection. The locations at University of Toronto and Carleton University are connected by a dedicated ATM connection¹. Connections with other sites are made over non-dedicated telephone lines.

Video and audio connections over non-dedicated phone lines are established with coder/decoders (codecs), which encode a full-duplex video and audio signal, compress it using an algorithm such as H.261 or a proprietary algorithm, and ship it digitally across telephone lines. Two switch-56 telephone lines provide a bandwidth of 112 Kbps, sufficient to ship a signal with about a 0.5 second delay. This delay is noticeable, and can introduce some confusion with social protocols [39], but can be eliminated with the use of broadband telecommunications technology, such as ATM. OTP is using a 45 Mbps ATM connection for two full-duplex A/V channels which provide instantaneous, full-motion video with no significant delay.

Each media space within a site is equipped with a number of multimedia devices: a camera, monitor, speaker, and microphone. These devices are connected directly to an A/V switch, which provides flexible, PBX-like control over analog audio and video connections between the devices.

Multi-site conferences can be supported in a number of ways. One approach is to use video and audio mixing technology. A signal from each site within the conference is shipped to a common site. This site combines the video signal from

¹The ATM connection between Ottawa and Toronto is a temporary connection, maintained from October to the end of December, 1994.

each site into a single, aggregate video signal, and combines the audio signal from each site into a single signal using audio mixing technology. The resulting audio and video signal is shipped back to all participating sites. Any number of video signals from sites may be combined into a single signal, but practical problems with identifying the current speaker and the resolution of the signal practically limits the multiplexing of video signals to four quadrants [39].

Another approach is to use voice-activated switching, where the signal that is shipped to all sites is the signal from the site with the current speaker. This site is identified by automatic voice recognition. This service is provided by the telephone company. However, voice activated signalling often chooses the wrong site because of background noise such as coughing or rustling of papers.

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 workstation located in the laboratory. The Toronto and Ottawa laboratories are also connected to the Internet.

2.2.2 Software

Integrated Interactive Intermedia Facility

The Interactive Integrated Interactive Intermedia Facility (IIIF) [24] implements the media space abstraction by hiding low-level device control of A/V switches, CODECs, VCRs, and cameras. A user can request IIIF to connect to a specific person, and IIIF then handles all low-level device control so that the media spaces are connected. Individuals can also indicate their availability by specifying a door state, which IIIF uses for access permissions to a media space.

The services of IIIF are provided to users within a site in a client/server capacity. The IIIF server is located on a suitable machine within a conveniently clustered group. Clients use TCP/IP 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.

IIIF's strength lies in its abstraction of device communication paths into connections between media spaces. However, it does not adequately support the concept of multi-site conferences between groups of people. A media space is defined in the context of an individual, not groups of people in a meeting room. IIIF supplies rudimentary control over devices during the course of the conference, but this control is restricted to devices under control of the local IIIF server. Although IIIF has been expanded to support the point-to-point connection of sites, it does not support connections among more than two sites. IIIF also has a very rigid architecture; it does not support a flexible environment for shared applications or device controllers. Changing the architecture of IIIF requires the entire system to be halted, modified, and restarted, resulting in a loss of service to users during the modification.

The software architecture of IIIF is not flexible. Device drivers are implemented as an integral part of the server, requiring the server to be modified, recompiled, and restarted to add or remove new device drivers. Although IIIF has been modified to support device drivers located on any workstation other than the server within a site, the device drivers must be implemented separately for each remotely located device, since there is no infrastructure support for remotely located device drivers.

IIIF is single-threaded. A user submits a request which is completed before the next request can be handled. A request often implies low-level control of a device; IIIF will not accept new requests while that device is being controlled. This is not a problem for devices that respond quickly to control requests; however, the introduction of CODECs that may take up to twenty seconds to establish a connection has resulted in an architecture in which users who submit local requests (such as controlling a VCR or camera) have to wait until the CODEC connection is established. This results in a considerable performance penalty for the throughput of requests to IIIF.

Desktop Area Network

The Desktop Area Network (DAN) presents a graphical user interface (GUI) to users to simplify control over their media space. The DAN environment provides control over devices within a user's physical work area. It allows a user to specify connections between these devices graphically, and the DAN makes the low-level connections, transparent to the user. It also allows a user to specify preset connections, so that common configurations can be saved and retrieved.

The DAN environment runs on a workstation which all devices under its control must be connected to. The current version of the DAN does not provide a set of services geared towards high-level control of conferences, and does not allow a user to control devices within another person's media space. Furthermore, it does not attempt to provide a platform for distributed, shared conferencing applications. However, work has begun on developing a user interface for a distributed DAN application that will provide basic conference start-up and multiple-room control over A/V devices.

2.3 Multipoint Communications Service

The Multipoint Communications Service (MCS) provides a logical set of communications services which manages a set of point-to-point connections [1, 3]. MCS is specified by a standard from the International Telecommunication Union (ITU) standardization sector as part of the specification for a complete protocol stack for audio/visual applications [2].

The services of MCS manage groups, or domains, where each domain organizes a set of reliable, point-to-point connections between MCS service providers which provide MCS services to a site². Each domain has a number of unicast and multicast

²A site is called a node in the ITU terminology.

channels for data communication and tokens for access to resources. Domains can be created and terminated at any time, and MCS service providers may leave and enter a domain at any time. Any number of domains may exist at any time. Domains provide flexible group membership.

The service providers within a domain and the point-to-point communications links between them form a hierarchical tree, in which the MCS service provider at the top of the tree is designated as the top MCS service provider. The top MCS service provider guarantees single source message ordering and multiple source message ordering. Multiple source message ordering is guaranteed by having the top MCS service provider receive messages from the sending service provider, and then retransmitting the messages back to all receiving service providers. The top MCS service provider also updates and distributes state information of tokens and channel membership.

Centralizing state information within a single service provider provides a simple solution for communication between distributed applications, but introduces concerns related to reliability and throughput. Failure of the centralized service provider would result in a loss of all centralized information, and the current state of the domain would be lost. If a domain has a large number of members and each member is sending a large amount of data which must be globally ordered, the queue of messages at the centralized service provider would continue to grow, exceeding the capacity of the service provider to process them.

Current research in the area of communication between distributed entities concentrates on providing single source and multiple source message ordering and reliable multicast and broadcast algorithms assuming unreliable communications connections and unreliable entities [28, 19, 9, 10]. In these cases, a centralized approach is not possible, since the failure of the centralized component would cause a failure in communications services. These approaches must also scale to any size group; a centralized approach cannot guarantee that the throughput for large groups can be

supported.

However, these approaches increase the number of messages sent between sites, and hence the communications overhead is increased. In an environment where the size of groups are small and the communications connections are reliable, a centralized architecture provides message ordering services with the least overhead for communications. The choice of a centralized architecture illustrates the environment for which MCS is designed. MCS providers and communications connections are expected to be reliable, and membership of domains is expected to be relatively small, so that the throughput of the top MCS provider is sufficient to provide timely message delivery.

2.4 Network Services Infrastructure

The Network Services Infrastructure (NSI) supplies a common interface to the network for Telepresence applications. It provides communications services, services for access to distributed resources, and services for access to remotely-located devices. Figure 2.3 shows the architecture of an NSI service provider at a site.

NSI clients interact with the NSI service process through a standard interface. NSI clients may be located on any workstation within a site; they access the server through a connection over a Local Area Network (LAN). The connection between the NSI clients and the NSI server process is asynchronous. Clients do not have to poll for network events; they are notified asynchronously of appropriate events by the NSI service provider.

2.4.1 Communication Services

The communication services of NSI are provided by MCS, which has been extended to support any number and combination of TCP/IP connections and multipoint networks while still retaining the original MCS inter-node protocol specification,

Figure 2.3: NSI Architecture

as proposed in [45]. The multicast service is provided by LLNL MCAST, which provides a set of multicast services on top of TCP/IP connections [37].

NSI simplifies group communication with MCS by hiding the details of setting up reliable point-to-point or multipoint connections between the members of a group from applications. Applications specify joining to a domain in either a master/slave arrangement or in a truly peer arrangement. The management of the hierarchy of connections and service providers is hidden from the application.

2.4.2 Distributed Resource Control

Distributed applications often make use of distributed resources, where a set of services provided to users do not refer to a specific physical device. Typical distributed resources that the NSI will support include a global name service implemented with an X.500 DNS entity. Other examples include a distributed database, a shared whiteboard application, or a shared editor.

NSI provides a consistent interface for clients to invoke distributed resources, where the distributed service is implemented by the distributed resource controller (DRC), which uses the services of MCS to communicate with peer-level DRCs at other nodes. The DRC allows an application to define logical resources and manage them with distributed algorithms. An application can then access these services, where NSI allows all instances of a distributed application to access the resource in exactly the same way.

2.4.3 Remote Device Control

NSI provides an abstraction to control devices at remote sites. A physical device controller (PDC) at the site where a device is located is responsible for interacting with a low-level device driver. A logical device controller (LDC) is located at each NSI node which wishes to make use of the services offered by the PDC. The LDC

also negotiates access to the device, depending on the algorithm for accessing the device. An LDC is dynamically spawned for each node accessing the device, as long as the node has the correct permissions.

2.5 Generic Conference Control

The Generic Conference Control (GCC) standard is a standard being written by ITU [4]. It defines a software architecture, a set of services, and a conference control protocol for conferencing nodes. It is also part of the suite of standards which develop a protocol stack for the deployment of conferencing applications, and uses MCS as its communications service.

The GCC standard defines a conference as “a number of nodes that are joined together and that are capable of exchanging audiographic and audiovisual information across various computer networks”.

The use of standards which define services and a protocol increase inter-operability of different implementations of conferencing systems. Issues related to the inter-operability in teleconferencing are discussed in [44], and also in the MMuSiC IETF working group.

2.5.1 Architecture

Figure 2.4 shows the architecture of a GCC conferencing node. The user at a conferencing site interacts with the node controller through the node user interface at a conferencing terminal. The node controller requests services from the GCC agent, which communicates with peer GCC agents at other sites using the communication services of MCS.

A set of client applications at a site are informed of changes in the state of a conference by GCC. The applications may also request the current state of the conference, for example, to query the list of current participants in a conference.

Client applications communicate with peer client applications at other sites using the communication services of MCS.

MCS communicates with other sites using the A/V protocol stack [2]. This service hides the low-level details of interacting with a communications network, so that any type of connection can be supported.

Communication between the node controller and client applications, and communication between client applications at a node, are not defined by the standard, and have been left as a local implementation matter.

2.5.2 Services

The services which GCC offers to a user fall under the following categories: conference establishment and termination, conference roster management, application roster management, conference conductorship and bandwidth control. An application registry service provides services that aid in establishing communications among peer client applications.

The conference establishment and termination services are provided for meta-conference management. A site can request to create a new conference, join an existing conference, leave a conference, or modify the parameters of a conference profile.

A conference profile characterizes its operating parameters, such as whether the conference is protected by a password, whether it is locked or unlocked, and whether it is listed. A password-protected conference requires a password from an agent before it can request to join the conference. A locked conference cannot be joined by an agent unless it is explicitly invited by the convener. An unlisted conference is not advertised when a list of conferences in progress is requested.

The convener site of a conference has control over accepting or denying requests related to that conference. The convener site of a special conference, the administrative conference, is the administrator; this site approves or denies requests to

Figure 2.4: GCC Conferencing Terminal as Defined by ITU

create new conferences or to query conferences in progress.

The conference roster is the list of current participants in a conference. A site must explicitly announce its presence in a conference once it has joined in order to participate in a conference. However, sites that wish to listen to a conference in progress but do not want to participate do not announce their presence in the conference. Thus, GCC can not only support a meeting in which all sites participate fully in the conference, but also conferences in which sites only listen, for example, lectures or broadcasts. These sites would not be listed in the conference roster.

The application roster holds the identity of all client applications participating in a particular conference. A client application at a site specifies to enter or leave a conference. Peer applications at other sites can determine whether a peer application at another site is available to a particular conference.

Conferences may also be conducted by a participant of a conference. In a conducted conference, all client applications at other participant sites must grant control to their peer application on the conducting site. Sites may also query the conducting site's identity at any time.

The application registry manages information related to an application's use of MCS communications resources, channels, and tokens within a conference. Applications use its services to discover how to contact peer applications at other sites. The application registry is implemented as a client application.

2.5.3 Protocol

MCS provides the communications services for GCC. A conference maps to a MCS domain. A GCC agent at a site forwards requests for GCC services within a conference from a local node to the top GCC service provider for that conference. The top GCC service provider is the centralized repository of all state information for that conference. Although GCC agents may maintain a separate copy of state information for faster local access, the top GCC service provider is the final authority

for the conference state.

The GCC agent that creates a conference is assigned the role of convener for that conference, and the agent becomes the top GCC service provider, which maps to the top MCS service provider of the domain, for that conference. The top GCC service provider also approves requests to join the conference, which are forwarded from other GCC agents.

The multiple-source message ordering services provided by MCS guarantee state synchronization between all GCC agents. The GCC protocol is therefore a "tight" conference control protocol. MCS assumes reliable communications nodes and communications connections; a failure of the top GCC/MCS node will bring down the entire conference, although the failure of a participant node will not affect other nodes in a conference. The centralized architecture of MCS also implies that conferences may not be among a large number of sites because of the throughput restrictions of the top MCS service provider.

The GCC standard is still under revision, and the GCC standardization committee has not yet arrived at an exact definition for formats of network protocol data units (PDUs). However, a model for different classes of network PDUs and their function have been defined for each proposed service.

Chapter 3

System Architecture

A number of conference control architectures have been examined in section 2.1. However, each of these fail to meet the requirements for a conference control architecture within the OTP environment.

In this chapter, the requirements for TCC are described. The system's major components are identified from an analysis model. These components are mapped to a high-level design model, from which the architecture for TCC is derived. It is shown that the original requirements for TCC are met by this architecture. Other conference control architectures from section 2.1 are compared to this architecture, and why these other architectures do not meet the specific requirements of OTP is also shown.

3.1 Conferences within OTP

The services that TCC offers must support the way conferences are held within the OTP environment, where a conference can be any type of interaction between users. Although some conferences, such as a weekly status meeting, are planned for a specific time and announced to all potential participants, most meetings are more casual in nature. A user at a site may request a conference at any time, and

provided the communications resources are available, any number of these casual meetings between sites may occur at the same time.

The size of meetings are usually limited to three or four sites, with each site occupying a different quadrant of the video signal. Although more sites are able to participate in the conference, dividing the video signal into more than four quadrants makes it difficult for participants to determine the identity of the current speaker and of other participants because of the video image resolution [39].

The membership of some meetings is restricted. The purpose of the meeting may limit the membership to a certain set of sites. Other meetings are open to all sites who wish to join, for example, a meeting where a person at one site asks a question from a person located at any other site who feels qualified to answer.

If a connection to a site within a conference is lost, the meeting is postponed and continued when the problem is repaired. The participation of the lost site is essential to fulfilling the purpose of the meeting. Connections are usually lost because of equipment problems at either site; the video signal is shipped across the telephone lines with a high degree of reliability.

The computer and A/V networks are distinct. Since the A/V network uses highly reliable connections provided by the telephone company, the computer network is more likely to fail than the A/V network. A failure in the computer network does not necessarily terminate the conference; the configuration of the A/V network at the time of the computer network failure is maintained by the A/V equipment. Participants at sites in the conference may lose control over the A/V network, but the conference may be able to continue with the current A/V network configuration, and control over the A/V network may be restored at a later time during the conference. Therefore, conferences within the OTP environment are held with a high degree of reliability; reliability issues are limited to potential problems with A/V equipment at a site.

A user at a site may make a presentation to all other sites involved in the

conference, and may make use of devices such as cameras and VCRs or software packages such as a shared drawing program. During the presentation, the presenter uses verbal cues to control access to these resources by users at other sites.

3.2 System Goals

The purpose of TCC is as follows:

TCC provides services for users to access resources within the context of conferences, regardless of where the resource is located. Users and resources are located at different sites, where a site is represented as a single entity within the conference. A resource may be a physical resource, such as a device, or a logical resource, such as a name server. Access to a resource depends on the role of a site within the conference and the services that the resource offers.

From the characterization of conferences in section 3.1, a number of requirements can be identified:

- *A user at any site must be able to discover which conferences are in progress at any given time, provided that the conference initiator requests it to be advertised.* Since interactions between sites are casual and may occur at any time, potential participant sites should be informed of new conferences as they are created. A site must be able to query which conferences are in progress at any time. However, the site that initiates the conference should have control over whether the conference is advertised.
- *The membership of conferences must be open.* A site must be able to join any conference in progress without being explicitly invited, conditional on permission from the initiator of the conference.

- *The membership of a conference must be controlled.* When a site requests to join a conference, the request is directed to the site responsible for the conference, which must be able to grant or deny membership to the requesting site.
- *The user should be able to implement different conference management policies with minimal changes to the system.* Although the conference management policy within the OTP environment is consistent across all sites, conferences with sites outside OTP may require different policies for managing the conference.
- *The system must allow different types of access policies for different types of resources.* Access policies may depend on the services of the specific resource or the policies of the conferencing environment. Users should also be able to request simultaneous access to a set of resources to make a presentation to other participants of the conference.
- *Changes in the resources at a site must not affect conferences in progress at that site.* Since any number of conferences may be in progress at any time, changes in the resource configuration should affect only those conferences where sites are accessing the resource, and only to the extent that the device is no longer available on the conference.

3.3 Definition of a Conference

The notion of a conference may depend on the particular conferencing scenario for which the conferencing system is being developed. For example, Ott's formalization of conferences starts with a specific description of the type of meetings for which the particular conferencing system has been developed, which in his case, follow the model of a business meeting [43]. A number of attributes are given which describe

the conference participants and their role in the conference, the conferencing environment, and the means of human interaction between the participants. The result is a definition of a conference which is sensitive to the context of Ott's particular conferencing environment.

However, the goal of TCC is to provide an architecture which may support a number of conferencing policies and scenarios. Thus, the definition of a conference will be limited to a more technical definition, similar to that of GCC: a conference is a number of sites that are joined together and are capable of exchanging audio-graphic and audiovisual information across various communication networks, such as a computer network and an A/V network.

3.4 System Representation

TCC is represented by a number of models. These models are developed using Jacobson's Object-Oriented Software Engineering (OOSE) methodology [31], and use the notation for timethreads and machine charts [13].

3.4.1 OOSE

OOSE represents the analysis and design of a system in several models. The **requirements model** consists of actors and use cases. Actors represent the entities that a system interacts with, while use cases specify the functionality that entities request from the system. Actors can be inherited from other actors, indicated with the dashed arrow, which indicates that the derived actors include all functionality of the ancestor actor as part of their behaviour.

The **analysis model** represents the logical structure of a system decomposed into several objects. Objects can be one of three types. The *entity* object models information that is held in a system over a long period of time and survives the life of a single use case. The *interface* object models the interaction of actors with the

system. The *control* object models behaviour that operates on more than one entity object and cannot be represented within entity objects. Groups of objects with related functionality may also be grouped hierarchically into subsystems. Figure 3.1 shows the graphical notation for each type of object.

Figure 3.1: OOSE Analysis Objects

The **design model** maps the analysis model to the implementation environment. Objects in the analysis model are represented by blocks. Blocks may be organized in a hierarchical manner. A single block may have several blocks embedded within it, where each block can represent a subsystem of further blocks.

The **implementation model** is the annotated source code for the system. Although an object-oriented language is not required, its use allows classes within the language to be mapped directly to blocks in the design model. This mapping allows behaviour specified in the design model to be traced directly to the code.

3.4.2 Machine Charts

Machine charts provide a notation that maps to the architectural components of a real-time system. Machine charts are useful at the design phase of a system, where objects in the analysis model are assigned to architectural components. Figure 3.2 summarizes the machine chart notation.

3.4.3 Timethreads

Timethreads are

Figure 3.2: Machine Chart Notation

paths along which time monotonically increases, that are traced through a design to give examples of macroscopic behaviour. A timethread starts at some point of stimulus, touches elements of the design in the order in which they are activated in response to the stimulus, and ends at some point of completion of processing or delivery of response [13].

Timethreads describe the end-to-end behaviour of a system in response to a stimulus. Timethreads are useful for mapping the behaviour of a system onto its models, from the requirements model through to the design model. Multiple timethreads represent concurrent behaviour; synchronization between different sets of behaviour can be represented by synchronization between the corresponding timethreads. Figure 3.3 shows a subset of the threads notation. For the complete notation, see [13].

Figure 3.3: Thread Notation

3.5 Requirements Model

3.5.1 System Context

Figure 3.4 shows the requirements model for TCC. Users of TCC are located at a particular site, and are represented by the conductor, participant, administrator, and convener actors.

Figure 3.4: Requirements Model

A **participant** site takes part in a conference. The participant may join a conference in progress, leave a conference in progress, use a resource in a conference, release a resource from the conference, query resources available, query the conductor and query participants within the conference. The participant site may also query conferences in progress.

A **convener** site manages a conference in progress. The convener may create a conference and terminate a conference, and grants or denies membership to the conference. The site that created the conference is its convener, and the same site remains the convener throughout the conference. There is only one convener for a conference.

A **conductor** site presents material to other participant sites in a conference. A conductor site may conduct a conference and release conductorship of a conference, which requests and releases exclusive access to a set of resources. Other participant sites must wait until the conducting site releases conductorship of these resources.

The convener and conductor sites within a conference are entitled to all the functionality available to a participant site. A dashed arrow in figure 3.4 shows that the convener and conductor actors inherit functionality from the participant actor. Any behaviour specified in the requirements model that involves the participant actor also includes the convener and conductor actor automatically.

A site may be involved in a number of conferences at the same time. It may be a conductor, a convener, or just a participant in each conference. Therefore, the same site may play the different roles represented by the conductor, convener, and participant actors in different conferences.

The **administrator** site provides global conference management. The administrator can grant or deny requests from a potential convener site to create a conference. The administrator site also maintains a list of all conferences in progress and informs all potential participant sites of new conferences as they are created. There is only one administrator site in the set of all potential participant sites for

all conferences.

In figure 3.4, the administrator actor is inherited from the convener actor. The administrator is a convener of a special conference, the *administrative conference*. All potential participant sites join this conference before requesting to join *ordinary conferences*. Therefore, the administrator actor has the behaviour of a convener, with additional functionality for managing all conferences in progress. The administrator can manage any number of conferences at the same time.

The **resource** actor represents resources located at different sites. A resource may be a physical device, such as a VCR or a camera, or a logical service, such as a centralized database or a directory service. The site at which the resource is located is responsible for providing and maintaining the resource. There may be many resources located at a single site.

There may be interactions directly between different actors or between different instances of the same actor. However, these interactions are outside the scope of the system under design. For example, resources may use a special scheme for the transmission of data directly to another resource. This data communication falls outside the scope of TCC.

Users at a site enter requests through a graphical or text-based user interface on a workstation at that site. All requests by users are made on behalf of the site.

3.5.2 System Functionality

System functionality is represented by use cases, which describe the response of a system to a stimulus from a user. The functionality described by use cases are derived directly from the requirements for the system.

Create a Conference

A potential convener site requests to create a new conference. The potential convener site makes this request as a participant of the administrative conference. The request is forwarded to the administrator site, which approves or denies the request. If the request is approved by the administrator, all participants of the administrative conference are informed of the new conference and the requesting site is delegated as the convener of the new conference.

If the administrator site rejects the request for a new conference, the participant of the administrative conference is informed, and the conference is not created.

Join a Conference

A potential participant of a conference requests to join a conference. The potential participant asks the administrator for permission to join the conference. The administrator then forwards the request to the convener of the conference.

In order for the request to be forwarded to the convener, the potential participant must be a participant of the administrative conference. The convener of the conference that the new participant wishes to join must also be a participant of the administrative conference.

The convener approves or denies the request. If the convener approves it, all sites participating in the conference are informed of the new participant. If the convener denies it, the participant is informed and is not added to the conference.

Resources may be required to set up audio and video connections between the site joining a conference and the sites already in the conference. The services of resources required in the conference are requested from the appropriate resources.

Leave A Conference

A participant site of a conference may leave at any time, without requesting permission from the convener. All participants are notified when a participant leaves a conference. Any resources that the participant was using in the conference must be released for access by other participants. Audio and video connections may have to be torn down. The services of resources affected by the participant leaving the conference are requested from the appropriate resource.

Terminate a Conference

The convener site of a conference may terminate a conference at any time. All participants of the conference are informed of the termination of the conference. The administrator is also informed of the conference being terminated, and removes it from the list of conferences in progress. Any resources that were assigned to the conference are made available to participants in other conferences, where the participants in other conferences are notified of the termination of the conference and the subsequent availability of the resources.

Query Conferences

A participant site of the administrative conference may request the conferences in progress at any time. The request is forwarded to the administrator. If the administrator approves the request, a list of all conferences is returned to the participant. If the administrator denies the request, the participant is informed and the list of conferences in progress is not given.

Query Participants

A participant site with a conference may request the list of other participant sites in the conference at any time. The request is forwarded to the convener of the

conference. If the request is approved by the convener, a list of all sites, with a description of their location, is returned to the participant. Otherwise, if the convener denies the request, the list of participant sites in the conference is not returned.

Conduct a Conference

A participant site within any conference may request to be the conductor of the conference at any time. If a conductor is already assigned to the conference, the request is denied. Otherwise, the participant site is assigned the role of conductor for the conference. All other participant sites within the conference are informed of the new conductor site.

A conductor site is granted exclusive access to resources while conducting the conference. If the resources are in use by other participant sites, access to the resources is taken away from the participants and given to the conductor.

Release Conductorship

A conductor site within any conference may request to release conductorship. The conductor site is assigned the role of an ordinary participant in the conference, and all other participant sites within the conference are informed that conductorship has been released.

Access to resources is given back to participant sites when the conductor releases the conductorship of the conference.

Query Conductor

A participant site within any conference may request the identity of the current conductor of the conference. The participant site is informed that either a conductor has not been assigned to the conference, or is given the identity of the conductor

site.

Use a Resource

A participant site may request to use a resource within a conference. If the participant is allowed to access the resource, the request is carried out. If the participant may not access the resource because exclusive access has already been granted to another participant site, the participant is informed and the request is not carried out.

Release a Resource

The participant site that has requested access to the resource may also release the resource. All participant sites are informed that the resource is now available to them.

Query Resources

Any participant site within a conference may request a list of all resources available to them within the conference. The request is forwarded to the convener. If the convener site allows the participant access to the list of resources, the list of resources available is returned to the participant.

3.6 Analysis Model

3.6.1 Subsystems

Figure 3.5 shows the analysis model for TCC, which is divided into a number of subsystems, shown as the site control, resource control, local site management, and inter-site management subsystems. Communication between the subsystems

is shown with an arrow marked with the relationship between instances of one subsystem to another. The analysis objects for each component are also shown.

Inter-Site Management

This subsystem includes all functionality for managing communication among sites within a group. There is one instance of this subsystem in the system.

The **inter-site transporter** control object encapsulates all behaviour for service requests for managing a group of sites, and includes the set of specific services that this subsystem offers to its users. It also forwards messages related to a service to other sites and receives messages from other sites. The **group** entity object keeps track of the membership of a group, and also maintains a unique identifier for the group. The **site** entity object holds information related to a site, such as the name and location of the site and how to contact it.

Local Site Management

This subsystem manages all information related to the role a site plays in a conference. It also maintains the identity of all conferences in which the local site participates. Since any number of sites may participate in conferences, a separate instance of the Local Site Management subsystem is required for each site participating in a conference.

The **conference manager** control object encapsulates all behaviour for service requests for managing the state of a conference as it relates to the site. It includes the set of specific services that it offers to its users at the local site, represented by the **local site** control object. It also receives messages related to these services from other sites from the inter-site management system, represented by the **remote site** control object.

The **conference** object holds the specific details of a conference in progress. If the site is the convener of the conference, it maintains the identity of all participants

Figure 3.5: Analysis Model

in the conference in **participant** entity objects and the identity of the conductor of the conference in the **conductor** entity object.

The **administrative** entity object and the **ordinary** entity object are inherited from the **conference** entity object. Some service requests are directed to a specific type of conference, for example, only a participant of the administrative conference may request a list of conferences in progress. These entity objects hold information about requests specific to a particular type of conference.

Resource Control

This subsystem manages the interaction of participants with a particular resource regardless of its physical location or implementation, where service requests from participants are translated into low-level interactions with a resource. The Resource Control subsystem is instantiated for each resource located at any site. Since more than one resource may be located at a particular site, the relationship between the Local Site Manager and the Resource Control subsystem is a one-to-many relationship.

A participant communicates with a resource through the **resource control** interface object. This interface is defined by the specific functionality offered by the resource. For example, a VCR may offer a PLAY, REWIND, and STOP command to the user.

The **resource manager** encapsulates all behaviour related to accessing and controlling a resource. The **local site** control object verifies a user's access to the resource and directs the service request to the remote site where the resource is located. The **remote site** control object processes service requests from remote sites and controls the resource through the **driver**.

A user at a site may own a resource. For example, if a conductor is assigned to the conference, no other users at other sites may access the resource. The **owner** entity object holds the identity of the site that owns a resource.

The conference manager of the local site management subsystem informs the resource controller of changes in access to the resource, such as when a conductor is assigned or released. The conductor does not interact with the resource directly; instead, the local site manager co-ordinates all resources at the conducting site as a group on behalf of the conducting user.

Site Control

This subsystem is the interface to the participant, convener, and conductor for requesting conference management services. A single instance of the Site Control subsystem is associated with a particular local site management subsystem, as indicated by the one-to-one relationship between the two subsystems.

The **site control** interface object presents the user with a set of services to control and request the current state of a conference. The **site manager control** object converts the user request into the appropriate requests for the conference manager of the local site management subsystem.

3.6.2 Communication between Components

Communication between the user and the system and between subsystems is held in messages. Different types of messages are represented by data values, shown on figure 3.6 and figure 3.7 as a small circle with an arrow. There are four types of messages: **request**, **confirm**, **indication** and **response**.

The request message is a service request from a user to the service provider. A confirm message is returned by the service provider to the user, and indicates the success of the request. An indication message is sent by a service provider and is initiated by a previous service request. A response returns parameters requested by the service provider in a previous indication message.

The timethread in figure 3.6 shows an example of the end-to-end behaviour of

a convener requesting a list of the current conferences and the messages sent to carry out this service request. The timethread touches the different subsystems representing the time order in which functionality contained within the subsystem is invoked. However, not all service requests made by a user follow this model. The timethread in figure 3.7 summarizes the messages for a user requesting to create a conference.

A single service request for a user may result in any number of messages between the components of the system. A request by the user to the site control subsystem may result in any number of requests and confirms to the local site management subsystem. Similarly, a request by the local site management subsystem may result in any number of request messages to the inter-site management subsystem. The messages generated for a service request depend on the protocol definition for each service provider.

3.7 System Architecture

Figure 3.8 shows the architectural components of TCC. Each subsystem of the analysis model is mapped to a software component within each site in the entire system.

3.7.1 System Components

A number of design decisions are required for the components shown in figure 3.8. The design decisions are summarized below.

Inter-Site Communications Manager

This process handles all communication of a local site with remote sites. Its functionality maps directly to the Inter-Site Management subsystem in the analysis model. The functionality is distributed over all sites participating in a conference.

Figure 3.6: Participant Requests List of Conferences

Figure 3.7: Convener Requests to Create Conference

Figure 3.8: System Architecture

The functionality of this process is implemented with the Network Services Infrastructure (NSI) as described in section 2.4. There is a single instance of this component at each site.

Local Site Manager

This process holds the state of all conferences that the local site is participating in and state information related to the role a site is playing within each conference. Its functionality maps directly to the Local Site Management subsystem in the analysis model. This functionality is distributed over all sites participating in a conference, with peer-level Local Site Managers communicating via the Inter-Site Communication and Management process.

The functionality of this process is implemented using a subset of the services of the Generic Conference Control (GCC) standard. There is a single instance of the Local Site Manager at a site.

Site Controller

This process presents the interface to the user for accepting conference management requests and handles indications from the Local Site Manager. These indications can be presented to the user for a response, or they can be handled by the Site Controller process according to a specific policy. The functionality of this process maps directly to the Site Control subsystem in the analysis model.

The user interface is text-based. Users type in commands through a command-line interface, and text messages are printed on the standard display. However, the user interface could be implemented as a Graphical User Interface (GUI), similar to the Desktop Area Network (DAN). There is a single instance of the Site Controller at each site.

Resource Controller

This process presents the interface to the user to control a device located at the same or a different site and maps requests from the user into low-level interactions with the resource. The controller for a resource is distributed across all sites in the conference; a resource controller that offers services to interact with a device requires a resource controller for that device at every site. The functionality of this process maps directly to the resource control subsystem in the analysis model.

The user interface is text-based. Users type in commands directed to the resource through a command-line interface and status messages from the resource are printed on the standard display. However, the user interface may be a GUI specific to the services offered by the resource. There may be multiple instances of the resource controller at each site, one for each resource located at the site.

3.7.2 Features of the Architecture

The design model in figure 3.8 displays a number of features which meet the requirements for TCC.

Distributed Architecture

Functionality is distributed across all sites participating in a conference. To allow for the dynamic management of connections between sites, functionality must be distributed across all sites. A site may establish a connection to a site to hold a conference, and tear down the connection once the conference is completed. The connection does not need to be maintained outside the conference context.

By using standardized communications services and protocols, sites with a different implementation of the standard may join in a conference without the prior exchange of specialized software for a specific conferencing environment.

Policy and Mechanism

The policy of a conferencing environment is implemented entirely in the Site Controller. The Site Controller translates requests for functionality specific to the conferencing environment into a set of requests for general services from the Local Site Manager. Therefore, the Local Site Manager contains the mechanism for conference control requests, while the Site Controller contains the policy.

State Synchronization

The state of a conference is maintained by the convener site. All requests for the conference state are directed to the convener site, making it the centralized repository of the conference state. The administrator site maintains the list of all conferences in progress. All requests for a list of conferences are directed to the administrator site, making it the centralized repository for all meta-conference information.

Sites are informed of changes to the conference state with the multiple-source message ordering services of MCS. A state update message is sent to a centralized site, then retransmitted to all other sites. Thus, all sites receive asynchronous notifications to changes in a conference state in the same order. All sites will thus interpret the changes to the conference state in the same manner as all other sites in the conference.

3.7.3 Fulfillment of Requirements

The architecture of TCC meets the original requirements for the system. The fulfillment of these requirements can be summarized as follows:

- *The administrator site informs users of new conferences and of conferences in progress.* Users are informed of new conferences as they are created and can request a list of conferences in progress by joining the administrative confer-

ence. The administrator site acts as a centralized repository of all information for these services.

- *The convener site supports conferences with open membership, but also allows the membership to be controlled.* User requests to join an advertised conference are directed to the convener site, which the convener can accept or reject. The convener site acts as a centralized repository for all information for services.
- *The site controller restricts the scope of changes to the system for implementing different conference management policies.* Since the conference management policy is implemented in the site controller only, changes to the system for different conference management policies is restricted to the site controller.
- *A separate resource controller for each resource allows different access policies for the resources to be specified.* The changes to the system as new resources are introduced to the conferencing environment are limited to the implementation of a controller for that resource. A user can also request exclusive access to a set of resources with the services for conference conductorship.

3.7.4 Goals of Other Architectures

Requirements

The conference control systems described in section 2.1 have a different set of goals and requirements than that required for the OTP environment, and can be summarized as follows:

- *A conference control architecture must scale to conferences with any number of sites.* Participants may be located anywhere on the Internet, and conferences may be among any number of participants.

- *A conference control architecture must allow conferences to continue if a connection to a site within the conference is lost.* Connections within the Internet cannot be assumed to be reliable. Therefore, a connection to a participating site may be lost at any time. The failure may be at any Internet node since packets may be routed through any node regardless of its participation in the conference. A conference should be able to continue despite these communications failures.

Implications

The above requirements are reflected in the design of other conference control systems. Because the connections between sites are unreliable, no single site may be responsible for the state information for the conference. A single site cannot be delegated to order communications to all other sites, since communications with the centralized site may be lost at any time. Furthermore, a centralized approach for ordering communications between instances of a distributed application does not scale to any number of sites.

Maintaining consistent state among a set of processes in which one of the processes may fail at any time is a problem common to all distributed applications. Many algorithms have been developed; for example [9, 28, 19]. However, each of these approaches increase the number of messages between processes, and the implementation of such algorithms can be complex.

Applicability to OTP

The above requirements are not applicable to OTP. Conferences within the OTP environment are limited in size, both by video signal restrictions and social problems introduced by larger meetings [39]. Also, the use of connections provided by the telephone company provide a high degree of reliability for communications links. These links are directly between the participating sites, rather than routed through

a packet-switched data network.

Centralizing conference state information and implementing message ordering with a centralized site meets the requirements for TCC, but limits the complexity of the architecture and protocol. Thus, a broad range of services are provided to users, while the implementation complexity is minimized.

The GCC and MCS standards define a broad range of conference control services. These standards are defined for a reliable environment, where the failure of the A/V connections to a site in a conference result in the conference being held at a different time, rather than the continuation of the conference without the participation of the lost site. This conferencing environment is similar to the OTP environment. Therefore, the GCC and MCS standards are well suited for use in the OTP environment.

Chapter 4

Design and Implementation

This chapter describes the design model for TCC by discussing how the site controller, resource controller, and the GCC server are designed. The details of how these system components interact are given. The implementation environment is described. The class library, which includes a non-preemptive threading package and frameworks for developing the site controller and resource controllers, is also detailed.

4.1 Design Model

Figure 4.1 shows the major components of the design model for TCC.

The *GCC server* process and the *NSI server* process map to the local site manager component and the inter-site communications manager component. The site controller and the resource controller map to the components of the same name.

4.1.1 GCC Server

Figure 4.2 shows the internal design of the GCC server.

The *local site* process manages a message sent by a client which contains a GCC event. The GCC event may be either a request for a service or a response to an

Figure 4.1: High-Level Design Model

Figure 4.2: Design of GCC Server

earlier indication. The local site process then calls the appropriate function call of the *GCC Services and Protocol* module with the parameters specified in the message.

The *remote site* process instructs the *NSI event processing* module to process an event from the NSI server. The NSI event processing module converts the NSI event, which may be a confirm or indication, into a function call with the appropriate parameters. The function calls are defined in the GCC services and protocol module. The NSI event processing module also composes the message for either a request or response event from a corresponding function call with its parameters, and sends this event to the NSI server.

The GCC services and protocol module calls an appropriate confirm or indication button when it generates the appropriate confirm or indication event as part of its protocol. These buttons convert the confirm or indication parameters into an appropriate ASCII message for the GCC event, and sends this message to the appropriate client or set of clients.

4.1.2 GCC Services and Protocol Module

The GCC services and protocol module provides the set of GCC services according to a protocol. Figure 4.3 shows the GCC services and protocol module in more detail. Each data object represents an instantiation of a class.

The *GCC request* and *GCC response* buttons are used to request a service or respond to a previous indication. A separate button exists for each service, where the parameters for the service are provided in the service call. A request button instantiates a single instance of the request class, where each type of service request refers to a request class inherited from a base request class. A response button instantiates a single instance of the response class, where each type of service response refers to a response class inherited from the base response class.

The *network* object takes a message of any length from a request or response object and breaks it into a number of packets to be sent using the NSI send service.

Figure 4.3: GCC Services and Protocol Module

It takes packets and recombines them into messages at the receiving site. The network object also creates an appropriate request or response object, depending on the type of message received, and hands the message to the newly created request or response object. The network object also checks whether the message's destination site is the same as the local site and is intended only for the local site. If so, it bypasses sending the message through NSI by creating an appropriate request or response object and hands the message directly to it. A request or response object does not have to concern itself with checking the message's destination.

The request and response classes manage the protocol behaviour of a request or response. Request and response objects encapsulate two roles for each request and response: they handle requests and responses generated by the local or the remote site. A request from a remote site may cause the GCC indication button to be pushed, while a response from a remote site may cause the GCC confirm button to be pushed.

There is a separate instance of the *conference* class for each conference in progress at the site. This object encapsulates all behaviour for setting the conference up joining a conference in progress, maintaining its associated state information, and tearing the conference down. Setting up and tearing down the conference requires a series of NSI requests and confirms for communications resources.

The conference object assumes different roles, depending on the role of the site in the conference. For a participant site, it just notes that the site is participating in the conference. For the convener site, it tracks participants and applications (in this case, resource controllers) involved in the conference and returns this information to any participant requesting it. It also validates any requests to join the conference, where a *slave attach* indication is a participant's request to join a conference in progress. For the convener site of the administrative conference, it tracks all conferences in progress and returns their identity to any participant of the administrative conference.

The administrative and ordinary classes are inherited from the conference object, which indicates the special status of the administrative conference.

The *application* and *participant* classes hold the identity of an application or participant in a conference. Each class is instantiated once for each application or participant; they are instantiated only for the conference convener. If a participant site requests this information, the request is forwarded from the participant site to the convener site.

The *conductor* object holds the identity of the conductor for a conference, and is responsible for grabbing and releasing the conductor token for a conference. Grabbing a token requires an NSI request followed by a confirm.

4.1.3 Site Controller

Figure 4.4 shows the internal organization of the site controller.

The *site control* process parses a conference control command provided by the user from the keyboard, and calls the appropriate function of the *conference control policy* module with the user-supplied parameters. The *site manager* process informs the *GCC event processing* module that a GCC event has been received, and instructs it to process the event. The *event dispatcher* is not shown in this diagram. The *GCC event processing* module parses GCC events that are received and composes GCC events based on a service call and its parameters. This module allows control over the client and the server side of a connection, and guarantees that the GCC server will never receive an event that it does not understand.

The *request/confirm* mechanism matches a confirm with the appropriate request, so that the conference control policy module can sequentially request GCC services without having to match confirms with responses. The waiting place inside the site controller is implemented using a semaphore as the synchronization mechanism between the site control and site manager processes.

The *conference control policy* module implements the policy of a conferencing

Figure 4.4: Design of Site Controller

site. Functionality in the policy module may be translated into several service requests to the GCC server. The conference control policy module also handles indications from and responses to the GCC server, by calling the appropriate function in the GCC event processing module. Finally, the conference control policy module informs the user of events by printing messages to the display.

The conference control policy module must be provided by the user, and depends on the set of conferencing services the user wishes to implement at that site. All other functionality of the site controller is hidden within an object-oriented framework, as described in section 4.4.2.

4.1.4 Resource Controller

Figure 4.5 shows the internal organization of a resource controller.

The *resource control* process parses a resource control command, as provided by the user from the keyboard, and calls the appropriate function of the *resource control operations* module with the user-supplied parameters. The *resource driver* process handles an external event provided by a resource, such as a physical device, and calls the appropriate function of the resource controller operations module with the resource status.

The *remote site* process instructs the NSI event processing module to parse the event and call the appropriate function of the resource control operations module with the appropriate parameters. The *local site* process instructs the *GCC event processing* module to parse the event and call the appropriate function of either the resource controller operations module or the request/confirm mechanism with the specified parameters.

The *resource control operations* module provides a set of services to control a resource, such as a set of services to handle the low-level control of a physical device with a device driver. The resource control operations module communicates with peer-level resource controllers at other sites directly through NSI. It is informed

Figure 4.5: Design of the Resource Controller

about changes in conference state with a GCC indication event. It may also request the current conference state through the request/confirm matching mechanism module.

The resource control operation module is specified by the user, and depends on the functionality of the resource the user wishes to implement at the site. All other functionality of the resource controller is hidden within an object-oriented framework, as described in section 4.4.2.

4.2 Implementation Environment

The implementation environment for TCC is the UNIX operating system, specifically SunOS version 4.1. There are several features that UNIX provides which simplify the implementation of TCC:

- Unix supports multiple processes at the operating system level. The separation of processes is enforced by the operating system; if a process attempts to access memory outside of its designated memory area, the operating system forcibly terminates the process before it can do any damage to other processes or to the operating system. The operating system also schedules processes by selecting the next available process to run according to a set of criteria applied to the process.
- The only permitted way for processes to communicate is via inter-process communication (IPC) primitives supplied by the operating system. One of the many IPC mechanisms is TCP/IP sockets, which implement buffered asynchronous messaging between processes in the Internet domain. Messages may be sent between any two processes located on any machine connected to the Internet.

- Sockets also provide support for dynamic client/server connections. A server provides a socket at a well-known address. Clients request to attach to the server through this well-known address, and the server may allow or deny clients access to the server. If the client is allowed access, a socket through which the server may send messages to the client is created. The connections between the client and the server is fully bi-directional.

4.3 Implementation of the Design Model

4.3.1 Inter-Process Communication

The Network Services Interface (NSI) Server is a distinct Unix process. The site controller, resource controller, and local site manager are also implemented as separate UNIX processes. Messages, which hold the details of events, are asynchronously sent between processes using TCP/IP sockets.

The site controller and resource controllers are clients of the GCC server, which is a client of the NSI server. Resource controllers are also clients of the NSI server. The support for client/server connectivity as provided by UNIX with TCP/IP sockets is used to manage the connections between clients and a server.

4.3.2 Format of Messages

Messages are composed of a set of ASCII characters. The message length is included as a set of ASCII digits at the start of the message. There are a few reasons why ASCII characters are convenient for the content of messages:

- *ASCII messages are portable.* Since the ASCII format is a standardized representation of characters, ASCII character values represent the same information across different implementation platforms.

- *ASCII messages are readable.* Messages can be understood directly by the user, without requiring translation from special message formats.
- *ASCII messages are interpretable.* ASCII messages can be interpreted and processed using a text interpreter language.

Using ASCII characters for the content of messages is not an efficient representation for volume data. However, it is an acceptable method for the messages sent between TCC components since they are short and do not carry a large amount of data.

4.3.3 Processing of Events

Once events are received from the process' waiting place, each one is processed according to its event source and type. Events from other components are read from the socket and interpreted using TCL [46].

TCL (Tool Command Language) is an interpreter which is embedded in programs as the basic command interpreter. It provides a parser for a simple textual command language and a C interface for programs to augment the textual command language with program-specific command primitives. TCL provides a flexible front-end for programs. Users are not restricted to a specific set of commands, but may write *scripts*, or programs in the textual command language, which call the program's command primitives.

The scripting language has many characteristics of procedural programming languages, such as variable substitution, conditionals, and loops. It is also extensible; users can define more complicated procedures from less complicated primitives.

Although TCL scripts may be of any length or complexity, the use of TCL in TCC is limited to the marshalling and unmarshalling of parameters held in ASCII messages that are sent between the software components. The behaviour of TCC is implemented entirely in a compiled language. However, using TCL for the processing

of messages between the software components provides the possibility of extending the behaviour of TCC by writing scripts, which do not require recompiling the software components. However, this feature of TCL as it is used in TCC has not yet been explored.

4.3.4 Messages from Clients

A client's connection to the GCC and NSI server is managed by an Application Programmer's Interface (API), shown by the GCC and NSI event processing modules in the design diagrams. The API converts a service request or response, implemented by a procedure call with the service request or response parameters, into an ASCII message containing the event. This message is then sent to the server. Messages from the server are converted from an ASCII message into a confirm or indication procedure call with the specified parameters, where the behaviour for the procedure call is specified in the code for the client. The API also hides all the details of managing the connection to the server.

If a client uses the API to communicate with a server, all messages sent to the server are guaranteed to be in the correct format. The server assumes that all clients are trustworthy. It is possible for an untrustworthy client to communicate directly with the server without using the API, and to submit a message that the server cannot complete, such as a TCL script that loops indefinitely. In this case, the server will not respond to any further messages from clients.

4.3.5 Specification of Event Behaviour

In the design diagrams of section 4.1, processes are actually non-preemptive threads, rather than UNIX processes.

There are several advantages to using a threading package for the internal organization of the GCC server, site controller, and resource controller. The behaviour

of the thread can be specified sequentially, where synchronization with other threads is embedded in its behaviour using semaphores.

This approach allows the behaviour for handling an event or series of events to be specified in a sequential manner in the code for a thread. This improves traceability from the requirements model through to the implementation model, since the code for a thread can map directly to end-to-end behaviour specified in the requirements model, such as the behaviour annotated by a timethread.

The use of non-preemptive threads simplifies source code organization, since it allows behaviour related to the same service, which is a logical sequence of events, to be grouped together in the same section of code. Otherwise, the behaviour of the logical service would require the code to be distributed throughout the code space, with separate code fragments for each event.

A threading package also allows sets of asynchronous events to be translated to synchronous calls. When a client requests a service from the GCC server, an asynchronous message is sent to the server. Later, when the service request is completed, an asynchronous message is sent back to the client from the GCC server. Other asynchronous messages, relating to other service requests made by the client, may be interspersed between the asynchronous request and reply message for the same service. Using threads in the client simplifies the matching of asynchronous request and reply messages, so that a client may make a synchronous call for a service in a style similar to a Remote Procedure Call (RPC) mechanism.

4.3.6 Event-Driven Software Model

The site controller, resource controller, NSI server and GCC server follow the model of figure 4.6.

All events are received from all sources at a single waiting place within the Unix process. After the event is processed to completion, the next event is received from the waiting place. If no event is available, the process waits until an event arrives.

Figure 4.6: Model for Event-Driven Software

The mechanism for the waiting place is supplied by the Unix operating system through the **select** operating system call. A set of file descriptors are supplied as the parameters of the **select** statement. It returns the file descriptor's identity on which an event has been received and is ready for processing. A timeout can also be specified as one of the **select** statement's parameters, which specifies a maximum waiting time for any event.

When an event is not available at any of the process' file descriptors, the Unix operating system marks the process as waiting for input and swaps the process out. Thus, if events are not available on any of the process' event sources, the process does not consume any system resources.

Since the **select** statement waits on a set of file descriptors, it can be triggered by any Unix event which operates with file descriptors, such as a user typing in input from the keyboard, characters being read from a file, or messages arriving on a socket.

The details of how a component manages the events and then distributes them to other entities within the component are not given in the design diagrams of section

4.1. However, figure 4.7 shows the event handling in more detail for the GCC server.

Figure 4.7: Event Handling for the GCC Server

In figure 4.2, events from the site controller, resource controller, and NSI server are shown as asynchronous jabs to the local site and remote site threading pools. Actually, a single thread waits on a waiting place (with the **select** command) for a GCC event or an NSI event from the site controller, resource controller, or NSI server, and creates a new local site or remote site thread depending on the type of event.

The *event dispatcher* thread encapsulates the state machine of figure 4.6. It

accepts new client connections and remembers the set of current clients. The event dispatcher creates a new local site or remote site thread and adds it to the local site or remote site pool of threads. The event is then given to the new thread which handles the event to completion. During the processing of this event, the thread may signal other, previously created threads at synchronization points among the threads. When the processing of the event is completed, the thread is destroyed.

The event dispatcher thread has a lower priority than all other threads. The threads within the component process the current event as much as possible before processing a new event from another component.

4.4 Implementation Model

TCC is implemented with approximately 21K lines of source code. The complexity is managed by organizing the source code into a class library and a number of modules. The class library is implemented in an object-oriented language, in this case, C++. Figure 4.8 shows the class library and modules for TCC. The classes for the GCC module are not shown in this diagram. The shaded classes indicate where clients of the GCC server inherit and provide functionality for a specific type of client.

4.4.1 Threading Classes

The non-preemptive threading package is implemented with the *thread* class, the *semaphore* class, and the *thread manager* class.

The thread class encapsulates the current state of a thread such as its context, stack, and priority. Context-switching among threads is managed with the UNIX `setjmp` and `longjmp` operating system calls. Behaviour of a thread is specified by inheriting a class from the thread class and providing the code in a virtual method.

The semaphore class is the synchronization mechanism among threads; an ordered queue holds the identity of threads in highest to lowest priority order. Threads

Figure 4.8: Class Library and Modules

with the same priority are queued in first-in-first-out order. The *ordered* class component implements the ordered queue.

The thread manager remembers the identity of the current thread and manages context-switching of the current thread with the most eligible ready-to-run thread. It also manages the dynamic creation and destruction of threading objects.

The stack for a thread is allocated from the free store, where the stack's size is indicated on creation of a thread object. If a thread does not have enough stack space, unpredictable behaviour will result, since the stack will overrun its designated space. The operating system cannot check for this condition, since it is unaware of how threads operate. Indicating a stack size that is far too large will cause the process to consume far more system resources than it requires and will affect the system's performance. Thus, determining the correct stack size is an important issue for threads implemented at the user level.

The thread manager helps determine the correct stack size by writing a known mask value to every location of the stack on creation of the thread object. When the thread object is destroyed, the thread manager checks the stack for the mask value and determines the high-water mark of stack usage by the thread. It then logs this usage information to a file, which is later used to adjust the stack sizes to more appropriate values. Once the correct stack values are determined, this feature is disabled so that its overhead is not present in the final system.

The currently scheduled non-preemptive thread relinquishes control of the processor. Issues such as protection of shared variables by mutual exclusion, and the guaranteeing that only one, non-reentrant call is made to the operating system at a time, are simplified with a non-preemptive threading package.

Some operating systems, such as Sun Solaris, provide support for threads, including stack bounds checking and re-entrant operating system calls [51]. For a discussion of the issues related to the implementation of threads, see [52].

4.4.2 Frameworks

The user must specify the conference control policy in the site controller and the resource control services in the resource controller. Object-oriented frameworks for the site and resource controllers helps to manage the complexity of these GCC server clients; the user specifies the policy or operation in a place holder specified by the framework.

The *TccApp* class is the framework for resource controllers, and the *Tnode* class is the framework for the site controller. The *Tclient* class holds behaviour common to the *Tnode* and *TccApp* classes, namely presenting the user with a prompt and accepting commands from the keyboard. The *TcommandApp* is a specialized framework for a resource controller specifically for the control of physical devices.

4.4.3 Components

Several classes simplify the complexity of clients of the GCC server. The *Tinterpreter* class encapsulates a TCL interpreter which understands a set of keywords, with the behaviour of each keyword specified in the *Tkeyword* class. Commands to control physical devices are specified in the *Tcommand* class. Information about a conference which a client requires is held in the *Tconference* class. The request/confirm matching mechanism is encapsulated in the *Trequest* class. The *Tagent* class encapsulates some basic behaviour for constructing an agent pool of threads. The *container* class manages collections of objects, and the *ordered* class manages a prioritized queue of objects.

4.4.4 Modules

The *logging* module is used for tracing events in all TCC components. It appends events, along with the time that they occur, to a logging file. The *client* and *server* modules manage the client and server sides of a TCP/IP connection. The *tcc-client*

is the TCC event processing API.

The *cman-driver* and *dboz-driver* handle the low-level interaction with a camera control device and a VCR control device. The device control defined as part of the NSI was not implemented by the time that TCC was implemented; functionality for low-level device control was implemented directly within applications instead.

Chapter 5

Testbed

This chapter describes the validation of the conference control architecture by applying it to the OTP environment. The testbed hardware and software configuration is described. A set of test cases, which implement a number of typical scenarios for the deployment of the software, are applied and the results described. Finally, a number of observations about the strong and weak points of the architecture, as observed during the tests, are described.

5.1 Configuration of Testbed

A testbed for the verification of TCC behaviour was set up at the OTP site in Ottawa. It consists of three logical nodes, all within the Ottawa site.

5.1.1 Hardware

Figure 5.1 shows the hardware configuration of the testbed. Two workstations are connected by a local area network. A VCR is connected to one workstation and a camera panning device is connected to the other via RS232 connections. The A/V switch is connected to the workstation with the camera panning device.

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

PRECISIONSM RESOLUTION TARGETS

Figure 5.1: Hardware Configuration of the Testbed

A camera is seated in the camera panning device, which can then point the camera in any direction. It also controls the zooming in and out of the camera.

A camera and monitor is at each of the two workstations. A third camera and monitor is placed in the conference room, where a camera is seated in the camera panning device. The camera and monitor audio and video connections are connected to the A/V switch.

5.1.2 Software

Figure 5.2 shows the software configuration of the testbed. The software to manage two distinct logical nodes are located on one workstation, and the software for the third logical node is located on the other workstation.

Each logical node consists of a node controller, a GCC server, an NSI server, a resource controller for the VCR, and a resource controller for the camera panning device. Low-level device control was implemented directly in these resource controllers since the device control services of the NSI were not yet implemented at the time of implementing TCC and the testbed.

The name of the logical node which controls the camera panning device in the conference room is named *conf*. The VCR is located at the logical node *orwell*, and the IIIF controller is located at the logical node *squik*. The node *orwell* is the administrator, that is, the convener of the administrative node.

The control of the A/V switch is managed by IIIF. One of the logical nodes has a resource controller which forwards requests to IIIF based on the current state of a conference. This is shown in figure 5.2 as the *IIIF controller*.

The *VCR controller* and the *camera pan controller* have one instance located at each node. These instances communicate using the services of the NSI server directly. The instance of the resource controller at the node on the workstation where the device is connected controls the device through the RS232 connection. All instances of these resource controllers present a command-line interface to the

Figure 5.2: Software Configuration of the Testbed

user for entering commands to control the device and displaying status messages as they are returned to the user.

5.2 Test Scenarios

The following scenarios show the results of each action, as shown on the display for each of the components for each of the logical nodes.

5.2.1 Meta-Conference Management

Many of the requirements for TCC relate to meta-conference management. A user at a node must be able to discover which conferences are in progress and may request to join to the conference. However, the convener of the conference must be able to allow or deny membership to nodes within the conference.

The node controller for the administrative node is first initialized. The node controller joins the node to the administrative conference as part of the initialization for that node. The node controller for the administrative node informs the user when another node has joined the administrative conference. The administrative node automatically approves each node's request to join the administrative conference.

The user at the administrative node then asks to create a new conference. The user input is shown after the prompt **Node Controller>**.

```
Initialization in progress...
Initialization completed...
conf has just joined the administrative conference.
squik has just joined the administrative conference.
Node Controller> create conference gerald
Conference conference successfully created.
```

The nodes *conf* and *squik* display a message notifying the user upon creation of a new conference, so that the node controllers at all nodes participating in the administrative conference have been informed of the new conference with an asynchronous

notification.

```
Initialization in progress...
Initialization completed...
Conference conference has been created by orwell.
```

Each node may also request which conferences are in progress at any time.

```
Node Controller> conferences
Conferences currently in progress:
ADMIN
conference
```

The node controller for *conf* and *squik* may then request to join a conference in progress.

```
Node Controller> join conference gerald
```

At this point, the convener of the conference, in this case *orwell*, is informed of another node requesting to join the conference. The administrator then explicitly allows or disallows the requesting node into the conference.

```
gerald at conf is requesting to join conference conference.
gerald at squik is requesting to join conference conference.
Node Controller> allow conference conf
Node Controller> allow conference squik
```

When the convener of a conference allows or disallows the node in the conference, the node is informed of the results of its request to join the conference.

```
Successfully joined conference conference.
```

When a node joins a conference, the IIIF controller determines which node has joined and then instructs IIIF to set up the appropriate connections. In this case, IIIF connects the A/V signals between node *orwell* and *conf*, and nodes *orwell* and *squik*, together.


```
gerald at orwell has announced on conference 2.  
gerald at conf has announced on conference 2.  
request to hook up to different location...  
orwell and conf  
IIIF command: connect indef node sprout node wroom  
IIIF command: connect indef node wroom node sprout  
request to hook up to different location...  
orwell and squik  
IIIF command: connect indef node sprcut node chris  
IIIF command: connect indef node chris node sprout
```

The convener can also terminate the conference.

```
Node Controller> terminate conference  
conference successfully terminated.
```

All other node controllers are informed of the conference being terminated.

```
Conference conference has been terminated by the convener.
```

The IIIF controller also tears down the connections between all nodes.

```
IIIF command: disconnect  
conference 2 has been terminated.
```

5.2.2 Conference Management Policy

One of the requirements for TCC is the ability to change the policy of conferences control with a minimum set of changes to its components. For example, the policy of requiring a node to explicitly join to a conference may not be suitable for automatically starting up a conference and co-ordinating all the devices for use in the conference.

The conference control policy of TCC was changed to have nodes automatically enter a conference when it is created by another node and to have the audio and video connections to the new node setup automatically without requiring intervention by the users at that node. This test demonstrates the flexibility of implementing

different conference management policies in TCC and the ability of TCC to manage all devices and connections between nodes without intervention by users. The changes affected only the node controller at each node.

The convener of the conference, in this case *orwell*, creates the conference, and automatically approves other nodes attempting to join the conference.

```
Node Controller> create conference gerald
Conference conference successfully created.
conf is requesting to join conference conference.
Automatically allowing the node into the conference.
squik is requesting to join conference conference.
Automatically allowing the node into the conference.
gerald at squik has just joined conference conference.
```

The node controllers at *squik* and *conf* automatically request to join the conference when it is created. All node controllers at all nodes within the new conference announce the presence of the newly-joined nodes in the conference.

```
Conference conference has been created by orwell.
Joining to conference conference automatically.
gerald at conf has just joined conference conference.
gerald at squik has just joined conference conference.
```

The IIF controller establishes the connections between the nodes once they have joined the conference.

5.2.3 Device Access Policy

Another requirement for TCC is the ability to specify a different type of access policy for different devices.

A node may grab a device, which is a request for exclusive access to the device by that node. This access is not the same as conducting a conferencing; the mechanism for grabbing a device is managed by the peer-level device controllers at each node, not by the GCC server.

For example, a node may request exclusive access to the VCR. If the device has

not yet been grabbed by a node, any node may control it. The node then grabs the device and may continue controlling the device, where status messages from the device are printed on the display for the node's device controller.

```
VCR> play
Vcr vcr1 returned status 0 06 C00200 +000006.
VCR> stop
Vcr vcr1 returned status 0 02 C00200 +000018.
VCR> grab vcr1
Vcr vcr1 has been grabbed by a node.
VCR> play
Vcr vcr1 returned status 0 06 C00200 +000018.
VCR> fast_rewind
Vcr vcr1 returned status 0 83 400200 +000027.
Vcr vcr1 indicated status 0 72 300200 -000002.
VCR> release
Vcr vcr1 has been released.
```

The other nodes may also control the device, until it is grabbed by the first node. They are then denied access to the device. They are informed when the device is released.

```
VCR> play
Vcr vcr1 returned status 0 06 C00200 +000006.
Vcr vcr1 has been grabbed by a node.
VCR> play
Vcr vcr1 is currently grabbed by another node.
VCR> stop
Vcr vcr1 is currently grabbed by another node.
Vcr vcr1 has been released.
```

This policy of grabbing and releasing devices is implemented within the device controller. Therefore, the policy can be different for each device.

When a node requests to conduct a conference, the node controller and all device controllers at all other nodes are informed of the conference conductorship. Similarly, all nodes and resource controllers are informed when conductorship is released.

```
Node Controller> conduct conference
Assigned conductor for conference conference successfully.
Node Controller> release conference
Released conductorship for conference conference successfully.
```

The other node controllers report the conference conductorship.

```
conf is now conducting conference conference.
Conductorship for conference conference has been released.
```

However, each device may interpret what conference conductorship means for that particular device. For example, the VCR device controller interprets it as allowing exclusive access to the device by the node that is conducting the conference. Other instances of the device controller at other nodes are not permitted to access the device while the conference is being conducted by another node.

```
Conductorship for conference has been granted to another node.
VCR> play
Conference is being conducted by another node.
Conductorship for conference has been released.
VCR> stop
Vcr vcrl returned status 0 02 C00200 +000020.
```

The camera pan device does not allow exclusive access to the device by the conductor, although a different policy may be implemented within the resource controller for the camera panning device.

The IIIF controller interprets conference conductorship by reconfiguring the A/V connections between the nodes so that the user at the conducting node is on the display of all participating nodes.

```
conf is conducting the conference.
IIIF command: connect indef node wroom node sprout
IIIF command: connect indef node wroom node chris
```

5.2.4 Flexibility of Resource Controllers

One of the requirements for TCC is that it must support the dynamic configuration of resource controllers, without affecting the state of conferences in progress.

This requirement was tested by terminating and restarting the resource controllers for the VCR and camera panning device during the course of the conference. When the resource controllers were terminated, the state of the conferences in which the devices were being used was not affected, although the device was no longer available to the conference. When the resource controller was restarted, it could be used by nodes for conferences in progress.

Chapter 6

Conclusions

6.1 Research Summary

TCC provides a software architecture for conference control which has been successfully applied to the OTP environment to provide a cohesive multipoint video conferencing service. It supplies a set of services to users which support the way in which conferences are held within OTP. The architecture is flexible; it can easily be adapted to conferencing environments with a conference control policy that differs from the OTP environment. TCC also supports any type of audio and video data transmission medium between sites; for example, the OTP environment uses base-band analog connections, dedicated digital connections, and non-dedicated telecommunications connections.

There are several areas where TCC has improved video conferencing within OTP: it provides meta-conference management services, it implements a conference management policy consistent with the way conferences are held within OTP, it allows access to resources despite their location or implementation, and it supports dynamic configuration of resources.

Meta-Conference Management

A user at a site can inquire what conferences are in progress at any time, and may request to join any conference in progress without having to be explicitly invited. The membership of the conference is controlled; the convener of the conference may allow or deny membership to any requesting user.

The meta-conference management services of TCC support the way meetings are held within the OTP environment. Spontaneous interactions may be held among users at different sites at any time, provided that the communications resources are available. Users are able to see what conferences are in progress and request to join a conference in progress without having to be explicitly invited to join the conference.

IIIF allows users to announce their accessibility to other users by locking a door, closing a door, or opening a door icon associated with the user. The GCC standard also allows the accessibility to a conference to be specified; the convener of a conference can specify the conference as locked so that only users that specify the password and know the specific conference name may enter the conference. However, the accessibility to conferences has not yet been implemented in the services of TCC.

Conference Management Policy

The implementation of a conference management policy is limited to changes in the node controller. TCC has been modified to support both the policy of users at sites requesting to join a conference explicitly and sites being joined to a conference automatically. The minimal changes required to TCC to support these two different policies are limited to the node controller.

The GCC server provides a general set of services for conference management, which can be used as the mechanism for implementing a wide range of conference management policies. Using a standardized set of services and protocol for the GCC server allows different implementations of conferencing systems to inter-

operate without the prior exchange of specialized software. The services of the GCC server are implemented according to an emerging standard. Although the specific services and protocol of GCC have not yet been finalized, the implementation of the standard demonstrates that it provides a range of services which can support different conference management policies.

Although the intent of the GCC standard is to specify a mechanism for any type of conferencing policy, the specific services of GCC still imply a particular conferencing environment. The convener of a conference is a centralized repository for the state of a conference. It also is the centralized communications entity that orders all messages broadcast to all participants. A centralized architecture for distributed applications imposes a limit on the number of communicating entities because of the throughput restrictions of the centralized entity. It also assumes that the centralized entity is highly reliable. Thus, the GCC standard assumes conferences that are limited in the number of sites and have highly reliable communications between sites, a model that is not suitable for conferencing over the Internet.

Thus, although TCC attempts to provide a mechanism for conference control that will support different conferencing environments, the adaptation of the GCC standard has imposed a particular conferencing environment for TCC.

The services provided to support the roles of a site within a conference, such as the services for conductorship of a conference, were found to be less important for conferences where verbal and visual cues can be used instead. The ATM connection between Ottawa and Toronto provides a video connection with no delay; social protocols can be used reliably in this case. The services to support the role of a site within a conference were found to be more important in situations where delays in video communication introduce problems with normal social protocol between users in a conference.

Access to Resources

TCC allows users at any site in a conference to access devices at any other site within the conference, according to the access policy implemented in the resource controller. TCC also allows access to logical resources regardless of the specific implementation. Each resource controller can specify a different policy for access by peer resource controllers at other sites, and a different policy for access by the conducting site.

Currently, access to devices is implemented with a separate device controller at each site communicating with the device controller at the site where the device is located. However, this approach does not scale to a large number of devices. Another approach that would scale better is to have a generalized device controller client at each site which can interact with any type of device controller server located at the site where the physical device is located. Each new device would only require a device controller server to be implemented, rather than both a server and a client.

Messages are sent to resource controllers with no timing guarantees. If a delay is introduced while controlling certain devices, the user may be frustrated by the device not responding quickly enough to commands. For example, if a user specifies a VCR to stop at a certain time, the delay in stopping the VCR should not be noticeable by the user, otherwise it will be stopped in a different spot than the user desired. Introducing a method to prioritize messages sent between applications by the NSI server may provide a more timely response of devices to a user's commands.

TCC does not yet provide a method for resource discovery by sites. As new resources become available, the services of the new resource should be announced to all sites so that the resource can be specified for use within a conference.

Dynamic Resource Configuration

A client/server relationship between the GCC server and applications allows dynamic changes in the specific set of resources at a site without affecting the state of conferences in progress. The only effect of a resource controller being terminated is that the resource is not available to sites until its resource controller has been restarted.

Clients may make many requests to the server and receive the responses to the requests in a different order. The requests and replies may also be interspersed with asynchronous notifications of changes to the conference state. Clients must match requests with responses and separate notifications from the responses.

This burden placed on the client is managed with a lightweight threads package within the client's process space. Another approach is to have the GCC server return responses to the clients in the same order that clients make the requests and to have asynchronous notifications returned to the client on a different TCP/IP connection. A standard RPC mechanism may then be used for clients making requests to the server, where the client waits for asynchronous notifications and the response from the server for the request. This approach may simplify the code in the client's process space.

6.2 Future Work

Although TCC provides a set of video conferencing services suitable for the OTP and other environments, there are still several areas which require further work.

The immaturity of the GCC services and protocol standard prevented full investigation into the issues of inter-operability between different conferencing systems. The GCC server should be changed to implement the services and protocol of the final accepted and approved GCC standard to demonstrate that TCC can inter-operate with other conferencing systems developed outside of the OTP environment.

The interface of the node controller and resource controllers is text-based. The use of graphical interfaces for device control has been investigated during the development of the DAN. The user interfaces developed for the DAN should be extended to TCC for the node controller and for resource controllers.

Currently, TCC has been developed for an automated conference room environment. Only one node controller may request conferencing services, where the node controller would be located on a workstation in the conference room. However, TCC should be extended to support many media spaces located within a site where each user at their media space has their own node controller. Users at a media space within one site may contact any number of users located at media spaces at other sites.

These changes to TCC will provide a video conferencing service that will support any number of media spaces at each site and will inter-operate with other independently implemented video conferencing services at other sites.

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.GCC: Generic conference control for audiovisual and audiographic terminals and multipoint control units. ITU Telecommunications Standardization Sector, 2-5 May 1994.
- [5] M. Ahmed and K. Tesink. Definitions of managed objects for ATM management version 8.0 using SMIV2. Internet RFC 1695, August 1994.
- [6] S. Armstrong, A. Freier, and K. Marzullo. MTP: An atomic multicast transport protocol. Technical Report X9100359, Xerox Webster Research Center, March 1991.
- [7] S. Armstrong, A. Freier, and K. Marzullo. Multicast transport protocol. Internet RFC 1301, February 1992.
- [8] R. Atkinson. Default IP MTU for use over ATM AAL5. Internet RFC 1626, May 1994.
- [9] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47-76, February 1987.
- [10] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(8):272-314, August 1991.

- [11] 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.
- [12] C. Brazdziunas. IP support for ATM services. Internet RFC 1680, August 1994.
- [13] 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.
- [14] W. Buxton. Integrating shared task and person spaces. In *Proceedings of Graphics Interface '92*, Vancouver, British Columbia, May 1992.
- [15] S. Casner. First IETF internet audiocast. *ACM SIGCOMM Computer Communication Review*, 22(3):92-97, July 1992.
- [16] S. Casner, C. Lynn, and P. Park. Experimental internet stream protocol: Version 2 (ST-II). Internet RFC 1190, October 1990.
- [17] S. Casner, K. Seo, W. Edmond, and C. Topolcic. N-way conferencing with packet video. In *The Third International Workshop on Packet Video*, Morristown, NJ, March 1990.
- [18] S. Casner, K. Seo, W. Edmond, and C. Topolcic. N-Way conferencing with packet video. In *The Third International Workshop on Packet Video*, Morristown, New Jersey, 22-23 March 1990.
- [19] J. Chang and N. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3):251-273, August 1984.
- [20] S. Deering. Host extensions for IP multicasting. Internet RFC 1112, August 1989.
- [21] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85-110, May 1990.
- [22] C. Elliott. High-quality multimedia conferencing through a long-haul packet network. Technical report, BBN Systems and Technologies, Cambridge, MA, April 1993.

- [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] Institute for the Future. Ten years of teleconferencing: Lessons from failures, hints for success. Technical report, Institute for the Future, Menlo Park, California, 1978.
- [27] Institute for the Future. A survey of permanently installed teleconferencing rooms in North America. Technical report, Institute for the Future, Menlo Park, California, 1981.
- [28] H. Garcia-Molina and A. Spauster. Ordered and reliable multicast communication. *ACM Transactions on Computer Systems*, 9(3):242-271, August 1991.
- [29] M. Handley and I. Wakeman. CCCP: Conference control channel protocol- a scalable base for buliding conference control applications. Dept. of Computer Science, University College London.
- [30] Multiparty multimedia session control internet engineering task force (IETF) working group. Mailing List: confctrl@isi.edu.
- [31] I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Publishing, Reading, MA, 1992.
- [32] V. Jacobson and S. McCanne. Using the LBL network whiteboard. UNIX Manual Pages, Lawrence Berkeley Laboratory, Berkeley, CA.
- [33] V. Jacobson and S. McCanne. vat: X11-based audio teleconferencing tool. UNIX Manual Pages, Lawrence Berkeley Laboratory, Berkeley, CA.
- [34] 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.
- [35] Bellcore Information Networking Research Laboratory. Touring machine system. *Communications of the ACM*, 36(1):68-77, January 1993.

- [36] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
- [37] C. Leedom. LLNL multicast distribution version 1.2. Technical report, Lawrence Livermore National Laboratory, January 1993.
- [38] M. Macedonia and D. Brutzman. MBone provides audio and video across the internet. *IEEE Computer*, 27(4):30, April 1994.
- [39] 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.
- [40] D. Mills. Network time protocol (version 3): Specification, implementation and analysis. Internet RFC 1305, March 1992.
- [41] G. Moore. Social science models and strategies for telepresence user studies. Technical Report OTP-93-01, Ontario Telepresence Project, Toronto, Canada, 1993.
- [42] 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.
- [43] J. Ott. The notion of conferences in the telecooperation project. Extended abstract. Technische Universitat Berlin, Germany, November 1993.
- [44] J. Ott. Service interoperability in teleconferences. Extended abstract. Technische Universitat Berlin, Germany, February 1994.
- [45] 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.
- [46] J. Ousterhout. TCL: an embeddable command language. In *Winter 1990 USENIX Conference*, Washington, D.C., January 1990.
- [47] 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.

- [48] E. Schooler and S. Casner. A packet-switched multimedia conferencing system. *ACM SIGOIS Bullentin*, 1(1):12-22, January 1989.
- [49] 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.
- [50] R. Stults. Media space. Technical report, Xerox Palo Alto Research Centre, 1986.
- [51] Sun Microsystems, Mountain View, CA. *SunOS 5.2 Guide to Multi-Thread Programming*, May 1993.
- [52] A. Tanenbaum. *Modern Operating Systems*, chapter 12, pages 507-522. Prentice Hall, Englewood Cliifs, NJ, 1992.
- [53] T. Turetti. H.261 software codec for videoconferencing over the internet. Technical Report 1834, INRIA, Sophia-Antipolis, France, January 1993.
- [54] L. Zhang, S. Deering, and D. Estrin. RSVP: a new resource ReSerVation Protocol. *IEEE Network*, pages 8-19, September 1993.

Appendix A

Node Controller User's Guide

This document describes the framework for the implementation of a node controller client, and describes how the functionality of the OTP node controller was implemented using this framework.

Class Declaration

The framework for the node controller is implemented as a C++ class. The declaration of the class is found in the file `Tnode.h`, which is as follows:

```
class Tnode : public Tclient {
    container<Trequest> requests;
    void GetResponse(void);
public:
    // Requests: Synchronous request/reply mechanism
    boolean ConferenceCreate(char *ConferenceName, int *ConferenceID,
                           char **Reason);
    boolean ConferenceJoin(int ConferenceID, char **Reason, GCCid *NodeID);
    boolean ConferenceQuery(char **Reason,
                          int *Conferences,
                          conference_specifier **Conference);
    boolean ConferenceAnnouncePresence(int ConferenceID,
                                       char *Location, char *Participants,
                                       char **Reason);
    boolean ConferenceDisconnect(int ConferenceID, char **Reason);
```

```

boolean ConferenceTerminate(int ConferenceID, char **Reason);
boolean ConferenceRosterInquire(int ConferenceID, char **Reason,
                                int *Nodes, int *AnnouncedNodes,
                                roster_entry *Roster[]);
boolean ConductorAssign(int ConferenceID, char **Reason);
boolean ConductorRelease(int ConferenceID, char **Reason);
boolean ConductorInquire(int ConferenceID, char **Reason,
                          boolean *Conducted, GCCid *NodeID);
boolean ApplicationInvoke(int ConferenceID, char *Application,
                           char **Reason);
boolean ApplicationRosterInquire(int ConferenceID, char **Reason,
                                 int *Applications,
                                 application_entry *Roster[]);

// Indications: use TCC responses directly to reply to these
virtual void ConferenceCreatedIndication(int ConferenceID, char *Name,
                                          GCCid NodeID);
virtual void ConferenceDisconnectIndication(int ConferenceID,
                                             GCCid NodeID,
                                             char *Reason);
virtual void ConferenceTerminateIndication(int ConferenceID,
                                           char *Reason);
virtual void ConductorAssignIndication(int ConferenceID, GCCid NodeID);
virtual void ConductorReleaseIndication(int ConferenceID);
virtual void ApplicationEnrollIndication(int ConferenceID, GCCid NodeID,
                                          char *Application);
virtual void ApplicationUnenrollIndication(int ConferenceID,
                                           GCCid NodeID,
                                           char *Application);
virtual void AnnouncePresenceIndication(int ConferenceID,
                                         GCCid NodeID,
                                         char *Location,
                                         char *Participants);
virtual void ConferenceRosterInquireIndication(int ResponseID,
                                                int ConferenceID,
                                                GCCid NodeID);
virtual void ApplicationRosterInquireIndication(int ResponseID,
                                                int ConferenceID,
                                                GCCid NodeID);
virtual void ConferenceQueryIndication(int ResponseID, GCCid NodeID);
virtual void ConferenceJoinIndication(int ResponseID, int ConferenceID,

```

```

GCCid NodeID);
virtual void ConferenceCreateIndication(int ResponseID, char *Name,
GCCid NodeID);

// event handlers
void OtherEvent(int fd);
virtual int TccEvent(int tcc_sap);
virtual void UserEvent(int fd);

// get/set attributes
Trequest *GetRequest(int RequestID);

// Creation/Invokation
Tnode(void);
void Invoke(void);
};

```

The user implements the functionality of a specific node controller by inheriting the **Tnode** class and providing functionality in the virtual methods as specified in the above class declaration. For example, a user's node controller specifies how to handle indications from the server in the virtual methods provided for indications. If the user does not override the virtual method, the default behaviour for the method is invoked instead.

The node controller is multi-threaded. Confirms from the GCC server are matched with the original request, in a fashion similar to an RPC mechanism, on a thread-by-thread basis. All currently outstanding requests are held in the **requests** container. A user can look up a particular request by using the **GetRequest** method. A user can use semaphores for synchronization between threads as implemented in the **threads** class.

Initialization

The user's node controller is initialized and invoked by declaring an instance of the user's framework and then calling the **Invoke** method.

When a node controller is invoked, it registers as a node controller with the GCC server for that node. If it cannot identify itself as a node controller with the server or if the server does not respond within some timeout value, the node controller terminates with an error.

Synchronous Requests/Confirms

The node controller requests the functionality of the GCC server with the synchronous request methods. The thread that makes the request contacts the GCC server with the request, and then blocks until the confirm for that request is received from the GCC server. If the GCC server cannot be contacted, the node controller terminates with an error.

The synchronous requests allow the end-to-end behaviour of a particular function in the node controller to be specified in the same area of code, preventing the implementation of the end-to-end behaviour being scattered in code fragments throughout the code space.

```
boolean ConferenceCreate(char *ConferenceName, int *ConferenceID,  
                        char **Reason);
```

The **ConferenceCreate** method sends a request to create a conference with the name **ConferenceName** to the GCC server. When the node controller receives the confirm for this operation from the GCC server and the administrator allowed the conference to be created, this method returns **TRUE**, else it returns **FALSE**. The ID of the newly-created conference is returned in **ConferenceID**. If a conference cannot be created, the reason for the failure is held in **Reason**. When a conference is successfully created, the site with the node controller that issued the request becomes the convener of the newly created conference.

```
boolean ConferenceJoin(int ConferenceID, char **Reason, GCCid *NodeID);
```

The **ConferenceJoin** method sends a request to join a conference with the identifier **ConferenceID**. If the convener allows this site to join the conference, the identifier for the site within this conference is returned in **NodeID**, otherwise the reason for the failure to join the conference is held in **Reason**.

```
boolean ConferenceQuery(char **Reason, int *Conferences,  
                        conference_specifier **Conference);
```

The **ConferenceQuery** method sends a request to query the conferences in progress to the GCC server. If the request is approved by the administrator, the number of conferences is returned in **Conferences** and the details of each conference are held in the set of conference specifiers **Conference**. If the request is denied, the reason is held in **Reason**.

```
boolean ConferenceAnnouncePresence(int ConferenceID,  
                                   char *Location, char *Participants,  
                                   char **Reason);
```

The **ConferenceAnnouncePresence** method sends a request to announce the site on the conference with the identifier **ConferenceID**. The location is given in **Location** and the list of participants at the site is given in **Participants**. If the request to announce the site fails, the reason is given in **Reason**.

```
boolean ConferenceDisconnect(int ConferenceID, char **Reason);
```

The **ConferenceDisconnect** method sends a request to disconnect this site from the conference with the identifier **ConferenceID**. If the request fails, the reason is given in **Reason**.

```
boolean ConferenceTerminate(int ConferenceID, char **Reason);
```

The **ConferenceTerminate** method sends a request to terminate the conference with the specified identifier **ConferenceID**. Only the convener site of a conference may issue this request. If the request fails, the reason is given in **Reason**.

```

boolean ConferenceRosterInquire(int ConferenceID, char **Reason,
                                int *Nodes, int *AnnouncedNodes,
                                roster_entry *Roster[]);

```

The `ConferenceRosterInquire` method sends a request to the GCC server to ask for the list of current participants in the conference with the identifier `ConferenceID`. The current number of sites is returned in `Nodes`, the number of sites that have announced their presence in `AnnouncedNodes`, and the details of each node is returned in the set of roster entries `Roster`. If the request fails or the convener does not approve the request, the reason is returned in `Reason`.

```

boolean ConductorAssign(int ConferenceID, char **Reason);
boolean ConductorRelease(int ConferenceID, char **Reason);

```

The `ConductorAssign` and `ConductorRelease` methods send a request to assign or release this site as the conductor of conference `ConferenceID`. If the request fails, the reason is given in `Reason`.

```

boolean ConductorInquire(int ConferenceID, char **Reason,
                          boolean *Conducted, GCCid *NodeID);

```

The `ConductorInquire` method sends a request for the identity of the conference conductor to the GCC server. If the request fails, the reason is returned in `Reason`. The status of conductorship and the identity of the conductor are returned in `Conducted` and `NodeID` respectively.

```

boolean ApplicationInvoke(int ConferenceID, char *Application,
                           char **Reason);

```

The `ApplicationInvoke` method sends a request to invoke the specified application `Application` on the conference `ConferenceID`. If the request fails, the reason is returned in `Reason`.

```

boolean ApplicationRosterInquire(int ConferenceID, char **Reason,
                                  int *Applications,
                                  application_entry *Roster[]);

```

The **ApplicationRosterInquire** method sends a request to the GCC server for the list of applications participating in the conference **ConferenceID**. If the request fails, the reason is given in **Reason**. Otherwise, the number of applications currently in the conference is returned in **Applications** and the set of application entries with the details of each application is returned in **Roster**.

Indications

The virtual methods for indications map to indications from the GCC server and are mapped directly onto the indications specified in the client API. Therefore, these methods are not explained in detail.

The advantage of providing a set of virtual methods for these indications is that the user does not have to override the method if the indication will not be used, whereas with the client API, the user must specify the behaviour for each indication regardless of whether it will be used or not.

Event Handling

The node controller waits on a set of file descriptors with the **select** statement. The file descriptor for the socket connection to the GCC server is also one of these file descriptors. The user can also specify additional file descriptors, such as the keyboard or a socket for X-protocol events. The **TcClient** class provides functionality to specify these additional file descriptors.

The method **UserEvent** is called when an event is received on one of the file descriptors in the aggregate set which is not the socket connection to the GCC server or the keyboard. If an event is received from the keyboard, the **KeyboardEvent** method is called as defined in the **TcClient** class.

When an event is received from the GCC server, the **TccEvent** method is called.

The default behaviour of this method is to call the appropriate virtual method for indication messages, or to match the confirm with the original request in the case of confirm messages.

OTP Node Controller

The declaration for the node controller for OTP as specified in the file `tpnode.cc` is as follows:

```
class TpNode : public Tnode {
    Tinterpreter cli;
    int NewID;
    Tstring error;
    Tstring location;
    container<request> requests;
    container<conference> conferences;
    conference *admin_conference;
    conference *GetConference(int id);
    conference *GetConference(char *Conference);
    void RemoveConference(conference *this_conference);
    void initialize(void);
public:
    // Telepresence node control behaviour
    // This behaviour is taken directly from the analysis model.
    // This is the set of services that TP requires.
    void CreateConference(char *Conference, char *Participants);
    void JoinConference(char *Conference, char *Participants);
    void AllowJoin(char *Conference, char *Location);
    void DisallowJoin(char *Conference, char *Location);
    void LeaveConference(char *Conference);
    void TerminateConference(char *Conference);
    void QueryConferences(void);
    void QueryParticipants(char *Conference);
    void ConductConference(char *Conference);
    void ReleaseConductorship(char *Conference);
    void QueryConductor(char *Conference);
    void UseResource(char *Conference, char *Resource);
    void QueryResources(char *Conference);
```



```

// Indications
// Specify the TP behaviour for indications.
void ConferenceCreatedIndication(int ConferenceID, char *Name,
                                GCCid NodeID);
void ConferenceDisconnectIndication(int ConferenceID, GCCid NodeID,
                                    char *Reason);
void ConferenceTerminateIndication(int ConferenceID, char *Reason);
void ConductorAssignIndication(int ConferenceID, GCCid NodeID);
void ConductorReleaseIndication(int ConferenceID);
void ApplicationEnrollIndication(int ConferenceID, GCCid NodeID,
                                 char *Application);
void ApplicationUnenrollIndication(int ConferenceID, GCCid NodeID,
                                   char *Application);
void AnnouncePresenceIndication(int ConferenceID,
                                GCCid NodeID,
                                char *Location,
                                char *Participants);
void ConferenceRosterInquireIndication(int ResponseID,
                                       int ConferenceID,
                                       GCCid NodeID);
void ApplicationRosterInquireIndication(int ResponseID,
                                       int ConferenceID,
                                       GCCid NodeID);
void ConferenceQueryIndication(int ResponseID, GCCid NodeID);
void ConferenceJoinIndication(int ResponseID, int ConferenceID,
                              GCCid NodeID);
void ConferenceCreateIndication(int ResponseID, char *Name,
                                GCCid NodeID);

// Other methods
TpNode(char *Location);
void Invoke(void);
friend class InitThread;

// Keyboard entry
void KeyboardEvent(char *event);
};

```

When the node controller is invoked from the command line, a description of the location that it is being invoked from must be specified as the first argument on the

command line.

The node controller is instantiated and invoked as follows:

```
int main(int argc, char *argv[]) {
    if(argc != 2) {
        printf("Usage: tpnode location, where location is text string.\n");
        exit(1);
    }

    TpNode node_controller(argv[1]);
    node_controller.Invoke();
}
```

When the telepresence node controller is invoked, it presents a prompt to the user. The user can then type in a number of commands with optional arguments. The commands are as follows:

- **create name participants:** The user requests to create a conference with the specified name and the specified list of participants at this site. The method **CreateConference** implements the end-to-end behaviour for this command, where a number of GCC server requests are required to implement this request.
- **join name participants:** The user requests to join the specified conference with the specified list of participants at this site. The method **JoinConference** implements the end-to-end behaviour for this command.
- **allow name location:** Allow the location into the specified conference. The user at the specified location asked to enter the conference at an earlier time. Only the convener site of a conference allows or disallows locations in a conference. Implemented by the **AllowJoin** method.
- **disallow name location:** Disallow the location into the specified conference. The location asked to join the conference at an earlier time. Implemented by the **DisallowJoin** method.

- **leave name:** The user requests to leave the specified conference. The method **LeaveConference** implements the end-to-end behaviour of this command.
- **terminate name:** The user requests to terminate the specified conference. The method **TerminateConference** implements the end-to-end behaviour of this command.
- **conferences name:** The user requests the names of all the conference currently in progress. The **QueryConferences** method implements the end-to-end behaviour of this command.
- **participants name:** The user requests the names of the participants in the specified conference. The **QueryParticipants** method implements the end-to-end behaviour of this command.
- **conduct name:** The user requests to conduct the specified conference, as implemented by the **ConductConference** method.
- **release name:** The user requests to release conductorship of the specified conference, as implemented by the **ReleaseConductorship** method.
- **conductor name:** The user requests the identity of the conductor for the specified conference, as implemented by the **QueryConductor** method.
- **use name resource:** The user requests to use the specified resource on the specified conference, as implemented by the **UseResource** method.
- **resources name:** The user requests the list of current resources in the conference, as implemented by the **QueryResources** method.

The behaviour specified by the Telepresence node controller for the indications is to display a message to the user that the specified indication has occurred. In the case of indications requiring a response, the appropriate response is called.

Appendix B

Resource Controller User's Guide

This document describes the framework for the implementation of a resource controller client, and gives the details of the OTP VCR device controller as an example of resource controller clients implemented for the OTP environment.

Class Declarations

The framework for the resource controller is implemented as a C++ class. The declaration of the class is found in the file `TccApp.h` as follows:

```
class TccApp : public Tclient {
    container<Tconference> conferences;
protected:
    int channels;
    char channel_id[256];
    void RegisterChannel(int channel_id);
public:
    // conference management
    virtual Tconference *CreateConference(int id);
    virtual void NewConference(Tconference *conference);
    virtual void OldConference(Tconference *conference);
    Tconference *GetConference(int id);

    // event handlers
    void OtherEvent(int fd);
```

```

virtual int TccEvent(int tcc_sap);
virtual int NsiEvent(int nsi_sap);
virtual void UserEvent(int fd);

// request/reply management
virtual void IncomingRequest(int id, int channel, int user,
                             char *request);
virtual void IncomingReply(int id, int channel, int user,
                           char *reply);

// Create and Invoke the TCC application
TccApp(char *name);
void Invoke(void);
};

```

The class `Tconference` holds all the information related to a particular conference, and all the indications are handled in this class. An instance of the `Tconference` class is created when a new conference is created. The event handler methods are similar to the event handler methods for the node controller client.

The file `Tconference.h` has the declaration for the `Tconference` class, as follows:

```

class Tconference {
    int result;
    phase_t phase;
    state_t state;
    char channel[256];
    int channels;
    semaphore attach_state, join_state, enroll_state, unenroll_state;
    boolean VerifyChannels(char *channel, int channels);
protected:
    int id;
    int user;
    virtual int Setup(void);
    virtual int Teardown(void);
public:
    Tconference(int id, char *channel, int channels);
    virtual ~Tconference(void);

    // attributes
    int GetId(void);

```

```

int GetUser(void);
boolean Connected(void);

// NSI event handlers for the specific conference
virtual void AttachConfirm(boolean result, char *reason);
virtual void DetachIndication(char *reason);
virtual void JoinConfirm(boolean result, char *reason,
                        int user, char *channel, int channels);
virtual void LeaveIndication(char *reason, char *channel, int channels);
virtual void ReceiveIndication(int user_id, int channel,
                              char *data, int size);
virtual void SequencedReceiveIndication(int user_id, int channel,
                                       char *data, int size);
virtual void TokenOperationConfirm(int token, NSI_Token_Result result,
                                  char *reason);
virtual void TokenTestConfirm(int token, NSI_Token_State state,
                              char *reason);

// TCC event handlers for the specific conference
virtual void ConferenceInquireConfirm(boolean result, char *reason,
                                     int nodes, int announced,
                                     roster_entry roster[]);
virtual void ConductorInquireConfirm(boolean result, char *reason,
                                     boolean conducted_mode,
                                     GCCid conductor_node);
virtual void ApplicationEnrollConfirm(boolean result, char *reason);
virtual void ApplicationUnenrollConfirm(boolean result, char *reason);
virtual void ApplicationInquireConfirm(boolean result, char *reason,
                                       int applications,
                                       application_entry application[]);
virtual void ConferenceAnnouncePresenceIndication(GCCid announcing_node,
                                                  char *location,
                                                  char *participants);
virtual void ConferenceDisconnectIndication(GCCid disconnecting_node,
                                           char *reason);
virtual void ConferenceTerminateIndication(char *reason);
virtual void ConductorAssignIndication(GCCid conducting_node);
virtual void ConductorReleaseIndication(void);
virtual void ApplicationInvokeIndication(int node_id);

// incoming request/request handlers

```

```

    virtual void IncomingRequest(int channel, int user, char *request);
    virtual void IncomingReply(int channel, int user, char *reply);
};

```

The **Tconference** class specifies all the client behaviour for indications. When the resource controller client receives an indication that invokes the resource on a conference, a new instance of the **Tconference** class is created. The user overrides the virtual methods to specify behaviour for each indication, where the method that maps to the appropriate indication is called for the appropriate instance of **Tconference**.

VCR Device Controller

The VCR device controller for OTP is an example of a resource client. The class declarations, as found in `vcrapp.cc`, are as follows:

```

class VcrApp : public TcommandApp {
public:
    VcrApp(void) : TcommandApp("VCR") { }
    void Initialize(void);
    void ReadConfigFile(void);
    void UserEvent(int fd);
    void IncomingReply(int id, int channel, int user, char *reply);
    Tconference *CreateConference(int id);
};

class VcrConference : public Tconference {
public:
    int conducting;
    VcrConference(int id, char *channel_id, int channels) :
        Tconference(id, channel_id, channels) { }
    void ConductorReleaseIndication(void);
    void ConductorAssignIndication(GCCid conducting_node);
};

```

The class **TcommandApp** is inherited from the class **TccApp**. It implements some additional functionality for having a remote instance of an application execute a

command entered by a different instance of the application and simplifies the implementation of distributed resource controllers.

The only indications that the VCR controller handles are those for assigning and releasing the conductor.

When the VCR controller is invoked, it presents a prompt to the user. The user may type in any one of a number of VCR commands and parameters. The user should just hit TAB at the prompt for a list of all possible commands, or enter **help** at the prompt.

When the VCR controller is invoked, it reads the list of all VCRs from a config file called **vcr.config**, which specify the machine locations of all VCRs at all OTP sites and the RS232 port for communicating with the VCR.

When the user types in a command, it is translated into a low-level command for controlling the VCR. The command is then issued. When the VCR completes the command, it returns the status on the RS232 port. The status is read and displayed on the screen for the user.

The user can also grab the device. When a site grabs a device, no other site can issue a command to the device. Furthermore, the site that has grabbed a device is also informed of changes to the status of the VCR caused by users controlling the VCR directly, for example, the play or stop button is pressed on the VCR.

Appendix C

GCC Server User's Guide

This document describes how to invoke the GCC server, and how to use the functionality of the client Application Programmer's Interface (API).

Starting the Server

The server is started by the command `TCC_server 1` for the administrator server and `TCC_server 2` and upwards for the participant servers for the administrative domain. Each server must have a unique number which is used to determine the port number of the server. The administrator has a port number of 5702, the participant with id 1 has a port number of 5703, and so forth.

Client API

The GCC server accepts commands in ASCII format where the client submits an ASCII string holding the name of the command and the parameters particular to that command. The client API converts a function call and its parameters into the appropriate outgoing ASCII message and sends this message to the server. It also converts ASCII messages from the server into call-back functions that must be

provided by the client, specifically confirm and indication functions.

Although it is possible to submit commands to the GCC server in ASCII format without using the client API, the functionality of the GCC server will be described in terms of the client API. The node controller and applications both use the same client API to interact with the GCC server.

Many of the client API calls return a positive integer on success or a -1 on failure. A failure reported by the client API indicates a problem with communicating the request to the GCC server; for example, the connection to the server may be lost. If a client function call is successful, the positive integer returned is an identifier for the request. When a confirm message is received from the server, the request ID associated with the confirm is the same value as the identifier of the original request. The call-back functions implemented in the client do not have return values.

Indications which require a response from the client also have an identifier associated with them. The client, when responding to these indications, must supply the identifier of the indication so that the server can associate the response with the original indication.

The GCC server is multi-threaded on a per-client basis. This means that the server may confirm requests from a client in a different order than the client originally issued the requests. It is the responsibility of the client to match confirms with the original request.

The complexity of the clients is hidden within a framework for the client, which may be either a node controller or an application. The low-level calls to the server are hidden within the framework. However, the low-level calls with the server are described here for completeness.

Initialization

```
int TCC_register(void);  
int TCC_deregister(int sap);
```

These calls establish and tear-down the connection to the GCC server by the client. The **TCC_register** call returns a SAP to the server, which is used in all calls to specify the particular connection to the server. There can be any number of client connections to the GCC server within the same UNIX process. The SAP is implemented in UNIX as a file descriptor, on which the client must wait for events from the GCC server.

A client uses the shell variables **TCC_SERVER** and **TCC_PORT** to identify the server and the TCP/IP port number of the conference control server that the client is connecting to.

Processing Messages from the Server

```
int TCC_process_event(int sap);
```

This call processes a message received from the GCC server by the client. The client must make this call when it is aware of a message being received from the GCC server. This function reads the message from the SAP and calls the appropriate callback function to specify which message was received. A return value of -1 indicates an error in handling the message and a possible problem with the connection to the server.

Client Identification

```
int TCC_node_controller_assign_request(int sap);  
void TCC_node_controller_assign_confirm(int sap,  
                                     boolean Reject,  
                                     char *RejectReason);
```

A node controller client must identify itself as a node controller before it can use any of the functionality provided by the server. The GCC server will only allow one node controller at a time; if a client requests to be a node controller while one already exists, the request is denied.

```

int TCC_node_controller_release_request(int sap);
void TCC_node_controller_release_confirm(int sap,
                                         boolean Reject,
                                         char *RejectReason);

```

A node controller that previously identified itself to the server uses this request to inform the server that it is no longer the node controller. If the node controller is terminated without performing this request, it is automatically released so that another client may ask to be the node controller.

```

int TCC_application_identify_request(int sap, char *name);
void TCC_application_identify_confirm(int sap,
                                       boolean Reject,
                                       char *RejectReason);

```

An application client must identify itself as an application before it can use any of the functionality provided by the server. Any number of clients may identify themselves as applications to the server. However, each application must have a unique name from all other applications at that node. If an application attempts to identify itself with the same name as another application, the request is denied.

```

int TCC_application_unidentify_request(int sap);
void TCC_application_unidentify_confirm(int sap,
                                         boolean Reject,
                                         char *RejectReason);

```

An application that previously identified itself to the server uses this request to inform the server that it is no longer available to participate in conferences. If a client application terminates without performing this request, it is automatically unidentified so that another client may identify itself as the same application.

Creating a Conference

```

int TCC_conference_create_request(int sap, char *ConferenceName);
void TCC_conference_create_confirm(int sap, int RequestID,

```

```

        int ConferenceID,
        boolean Reject,
        char *RejectReason);

```

This request is performed by a node controller. If this request is performed by an application or a client that has not yet identified itself, the request is denied. The request is forwarded to the administrator, and the status of the request depends on the administrator's response.

The domain identifier of the newly-created conference is returned in the confirm function. Applications use this identifier to attach to the specific domain for the conference.

The node from which this request originates must be a participant of the administrative conference. A node may request to join the administrative conference with the `TCC_conference_join_request` function.

```

void TCC_conference_create_indication(int sap, int response_id,
                                     char *ConferenceName,
                                     GCCid RequestingNode);
int TCC_conference_create_response(int sap, int response_id,
                                   int ConferenceID, boolean Reject,
                                   char *RejectReason);

```

The node controller at the administrator node receives this indication when a participant of the administrative conference requests to create a conference. The requesting node is the identifier of the participant on the administrative conference; each participant in a conference is assigned a unique identifier upon joining the conference.

The node controller at the administrator node uses the response function to indicate whether the node's request is granted or rejected. If the request is rejected, the administrator indicates the reason in the `RejectReason` string.

```

void TCC_conference_created_indication(int sap, int ConferenceID,
                                       char *Name,
                                       GCCid CreatorNode);

```

When a conference is successfully created, all nodes currently participating in the administrative conference are informed of the new conference with this indication, so that nodes do not have to poll to determine if a new conference has been created.

If the administrator node requests to create a conference, the request is automatically granted without giving an indication to the administrator.

Joining a Conference

```
int TCC_conference_join_request(int sap, int ConferenceID);
void TCC_conference_join_confirm(int sap, int RequestID,
                                int ConferenceID, GCCid NodeID,
                                boolean Reject, char *RejectReason);
```

A node controller uses this request to join a previously created conference. The identifier of the conference is specified in the request. If access to the conference is permitted by the convener of the conference, the confirm indicates that the request is not rejected. If rejected by the convener, the reason is held in the `RejectReason` string. The ID of the participant on the new conference, where the id of a node on a particular conference is on a per conference basis, is returned in the confirm.

```
void TCC_conference_join_indication(int sap, int response_id,
                                    GCCid ConferenceID,
                                    GCCid RequestingNode);
int TCC_conference_join_response(int sap, int response_id,
                                boolean Reject, char *RejectReason);
```

The node controller at the convener node of a conference receives this indication from a future participant of the conference. The ID of the requesting node on the administrative conference is given. The convener node grants or denies access with the response function.

Querying Conferences in Progress

```
int TCC_conference_query_request(int sap);
```

```

void TCC_conference_query_confirm(int sap, int RequestID,
                                boolean Reject,
                                char *RejectReason,
                                int CurrentConferences,
                                conference_specifier Conference[]);

```

A node controller or an application makes this request to ask for a list of conferences currently in progress. The confirm holds the number of conferences in progress and the details of each conference, which include the unique identifier and name for each conference. The node must be a participant of the administrative conference before this request can be made.

```

int TCC_conference_query_response(int sap, int response_id,
                                boolean Reject, char *RejectReason);
void TCC_conference_query_indication(int sap, int response_id,
                                    GCCid RequestingNode);

```

The node controller at the administrator node receives this indication when a participant of the administrative conference requests a list of conferences in progress. The administrator allows or denies the list of conferences in progress to the requesting node with the response function.

Conference Roster Management

```

int TCC_conference_announce_presence_request(int sap,
                                             int ConferenceID,
                                             char *Location,
                                             char *Participants);
void TCC_conference_announce_presence_confirm(int sap, int RequestID,
                                              int ConferenceID,
                                              boolean Reject,
                                              char *RejectReason);

```

The node controller of a participant node in a conference uses this function to announce its presence to all other nodes participating in the conference. Once the node is announced on the conference, its identity is added to the conference roster.

```

void TCC_conference_announce_presence_indication(int sap,
                                                int ConferenceID,
                                                GCCid AnnouncedNode,
                                                char *Location,
                                                char *Participants);

```

All participant nodes of a conference receive this indication when another participant announces its presence in the conference. The ID of the announcing node is also given. A response is not expected for this indication.

```

int TCC_conference_roster_inquire_request(int sap, int ConferenceID);
void TCC_conference_roster_inquire_confirm(int sap, int RequestID,
                                           int ConferenceID,
                                           boolean Reject,
                                           char *RejectReason,
                                           int NodesInConference,
                                           int AnnouncedNodes,
                                           roster_entry Roster[]);

```

The node controller or an application at a node in a conference uses this request to ask for the list of participants in a conference. The confirm function gives the number of nodes in the conference, whether they have announced or not, and the identity of each node which includes its identifier on the conference, its location, and its participants in the conference.

```

void TCC_conference_roster_inquire_indication(int sap, int response_id,
                                              int ConferenceID,
                                              GCCid RequestingNode);
int TCC_conference_roster_inquire_response(int sap, int response_id,
                                           boolean Reject,
                                           char *RejectReason);

```

The node controller at the convener node of a conference receives this indication when a participant node requests the identity of all other nodes in the conference. The node controller can allow or deny the request for the list of participants with the confirm function.

Conductor Management

```
int TCC_conductor_assign_request(int sap, int ConferenceID);  
void TCC_conductor_assign_confirm(int sap, int RequestID,  
                                int ConferenceID,  
                                boolean Reject, char *RejectReason);
```

The node controller at a node participating in a conference may request to be the conductor of the conference with this request. If another node is currently the conductor, the request is refused. If conductorship is successfully granted, all applications registered at that node are informed with an indication.

```
void TCC_conductor_assign_indication(int sap, int ConferenceID,  
                                    GCCid ConductingNode);
```

All applications and node controllers at all nodes participating in a conference receive this indication when conference conductorship is assigned to another node. The node that requested conductorship does not receive this indication.

```
int TCC_conductor_release_request(int sap, int ConferenceID);  
void TCC_conductor_release_confirm(int sap, int RequestID,  
                                  int ConferenceID,  
                                  boolean Reject, char *RejectReason);
```

The node controller at the node that previously requested to be the conductor uses this request to release conductorship. If the node requesting to release conductorship is not the conductor, the request is rejected, otherwise the request is accepted. The requesting node is notified with the confirm. If conductorship of a conference is successfully released, all applications at the node are informed with an indication.

```
void TCC_conductor_release_indication(int sap, int ConferenceID);
```

When the conducting node releases conductorship, all participant nodes in the conference are informed with an indication. All applications and node controllers at all other nodes in the conference receive this indication.

```

int TCC_conductor_inquire_request(int sap, int ConferenceID);
void TCC_conductor_inquire_confirm(int sap, int RequestID,
                                   int ConferenceID,
                                   boolean Reject, char *RejectReason,
                                   boolean ConductedMode,
                                   GCCid ConductorNode);

```

The node controller and applications at all nodes in a conference may request the current identity of the conductor with this request. The node that requests the identity of the conductor must be joined to the conference to get the identity. The ID of the conducting node is returned in the confirm message.

Application Management

```

int TCC_application_invoke_request(int sap, int ConferenceID,
                                   char *ApplicationKey);
void TCC_application_invoke_confirm(int sap, int RequestID,
                                   int ConferenceID,
                                   boolean Reject, char *RejectReason);
void TCC_application_invoke_indication(int sap, int ConferenceID,
                                       GCCid NodeID);

```

A participant node can request an application to participate in a particular conference. When the request is made, the user must specify the conference identifier and the name of the application. The name of the application corresponds to the name an application gives when it identifies itself to the server. An application with the specified name at all other nodes in a conference are notified of the request to be invoked on a conference with the indication function.

```

int TCC_application_enroll_request(int sap, int ConferenceID);
void TCC_application_enroll_confirm(int sap,
                                   int ConferenceID,
                                   boolean Reject, char *RejectReason);
void TCC_application_enroll_indication(int sap, int ConferenceID,
                                       GCCid EnrollingNode,
                                       char *ApplicationKey);

```

Once an application receives an indication to be invoked on a conference, it enrolls for the specified conference. If an application receives a request to be invoked on a conference and does not enroll for the conference, the application cannot participate in the specified conference.

All applications and node controllers at all other nodes are informed of the application enrolling in the conference with the indication.

```
int TCC_application_unenroll_request(int sap, int ConferenceID);
void TCC_application_unenroll_confirm(int sap,
                                     int ConferenceID,
                                     boolean Reject, char *RejectReason);

void TCC_application_unenroll_indication(int sap, int ConferenceID,
                                         GCCid UnenrollingNode,
                                         char *ApplicationKey);
```

If an application wants to leave a conference, it requests to leave with this request. All other applications and node controllers at all nodes in the application are informed of the application leaving with the indication.

```
int TCC_application_roster_inquire_request(int sap, int ConferenceID);
void TCC_application_roster_inquire_confirm(int sap, int RequestID,
                                           int ConferenceID,
                                           boolean Reject,
                                           char *RejectReason,
                                           int Applications,
                                           application_entry Roster[]);
```

An application or the node controller at a node can request the list of applications currently participating in a conference. If the request is approved by the administrator, the list of applications participating at each node in the conference, along with the node identifier, is returned to the user. Otherwise, the user is informed that the request is rejected and the reason for the rejection is given.

```
void TCC_application_roster_inquire_indication(int sap, int response_id,
                                              int ConferenceID,
```

```
                                GCCid RequestingNode);  
int TCC_application_roster_inquire_response(int sap, int response_id,  
                                boolean Reject,  
                                char *RejectReason);
```

The node controller at the convener node of a conference is informed of a request by a participant node for the application roster. The user at the convener node may approve or deny the request, with the reason for the denial specified in the **RejectReason** string.

END

3 1 - 0 5 - 9 5

FIN

A camera is seated in the camera panning device, which can then point the camera in any direction. It also controls the zooming in and out of the camera.

A camera and monitor is at each of the two workstations. A third camera and monitor is placed in the conference room, where a camera is seated in the camera panning device. The camera and monitor audio and video connections are connected to the A/V switch.

5.1.2 Software

Figure 5.2 shows the software configuration of the testbed. The software to manage two distinct logical nodes are located on one workstation, and the software for the third logical node is located on the other workstation.

Each logical node consists of a node controller, a GCC server, an NSI server, a resource controller for the VCR, and a resource controller for the camera panning device. Low-level device control was implemented directly in these resource controllers since the device control services of the NSI were not yet implemented at the time of implementing TCC and the testbed.

The name of the logical node which controls the camera panning device in the conference room is named *conf*. The VCR is located at the logical node *orwell*, and the IIIF controller is located at the logical node *squik*. The node *orwell* is the administrator, that is, the convener of the administrative node.

The control of the A/V switch is managed by IIIF. One of the logical nodes has a resource controller which forwards requests to IIIF based on the current state of a conference. This is shown in figure 5.2 as the *IIIF controller*.

The *VCR controller* and the *camera pan controller* have one instance located at each node. These instances communicate using the services of the NSI server directly. The instance of the resource controller at the node on the workstation where the device is connected controls the device through the RS232 connection. All instances of these resource controllers present a command-line interface to the