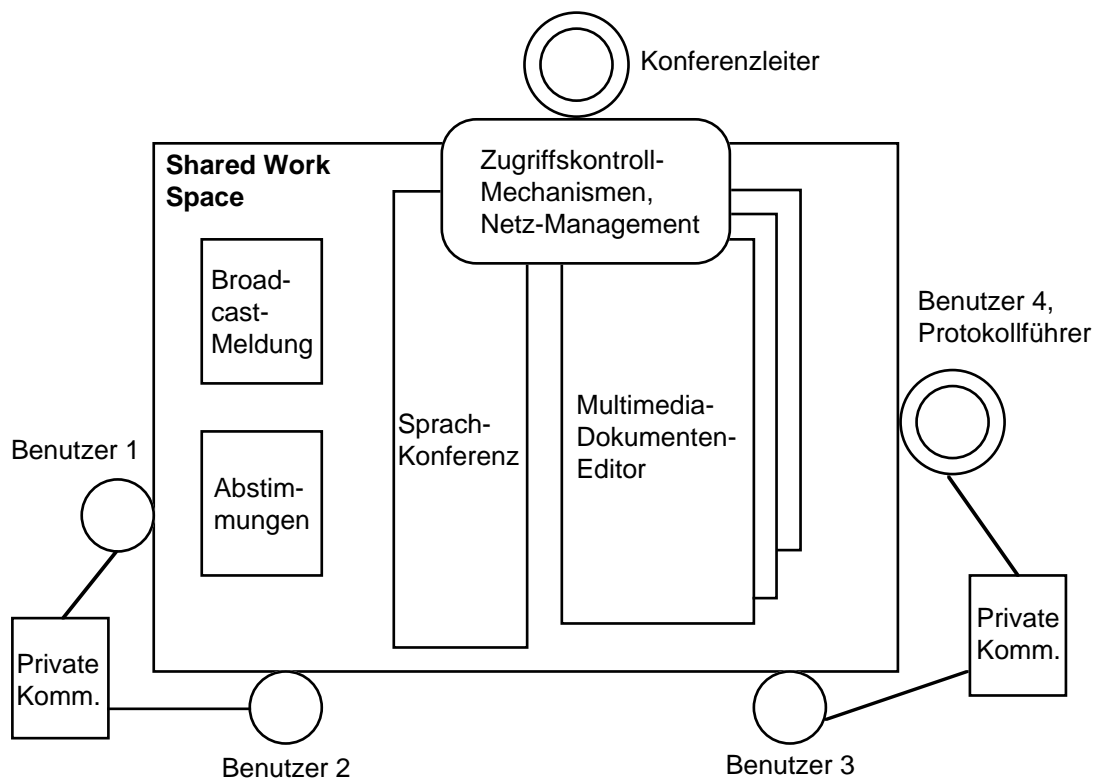


# Implementationsbeschreibung

## zum ZBF-Projekt 224 Z:

### MultimETH



# Inhaltsverzeichnis

Inhaltsverzeichnis .....	2
1. Beschreibung des Gesamtsystems .....	3
1.1. Das Konferenzmodell.....	3
1.2. Die technische Realisierung des Prototyps .....	4
2. Beschreibung der funktionalen Einheiten.....	7
2.1. Kommunikationssystem.....	7
2.2. Benutzerschnittstelle und Konferenzverwaltungssystem .....	9
2.3. Dokumentenverwaltung und -konvertierung.....	11
2.4. Multimedia-Mehrbenutzer-Editor.....	15
2.5. Netzwerk-Administration und Sicherheitsaspekte.....	20
2.6. Sprachkonferenz-Einrichtung.....	25
3. Allgemeine Anmerkungen und Gesamtbeurteilung.....	30
4. Literatur.....	31
4.1. Studentische Arbeiten an der ETH Zürich.....	31
4.2. Andere relevante Arbeiten .....	32

# 1. Beschreibung des Gesamtsystems

## 1.1. Das Konferenzmodell

Das MultimETH-System ist eine komplexe verteilte Applikation, die es räumlich und/oder örtlich verteilten Benutzergruppen ermöglicht, Sprach- und Datenkonferenzen zu administrieren und durchzuführen, sowie multimediale Dokumente zu bearbeiten und einen verteilten Dokumentenraum zu verwalten. In einer MultimETH-Konferenz betreiben mehrere Teilnehmer eine synchrone n:m-Mehrwegkommunikation, wobei jedoch auch die Untermengen 1:1- (Zwiesgespräch) und 1:n-Kommunikation (Vorführung) möglich ist. Das Konferenzmodell umfasst also zunächst die Funktionalität, die während der Dauer einer Konferenz benötigt wird. Zusätzlich wird jedoch auch der Zeitraum vor oder nach einer Konferenz einbezogen, da beispielsweise asynchron zu einer Konferenz eingeladen werden kann, wobei bereits zum Konferenzbeginn nötige Information (z.B. gemeinsam zu editierende Dokumente) vorab an die Teilnehmer verteilt werden kann. Ebenso kann eine Mitschrift einer Konferenz zur späteren Erarbeitung eines Verlaufsprotokolls erstellt oder ein gemeinsam editiertes Dokument abgespeichert bzw. anderen Benutzern zur Verfügung gestellt werden. Die auf dem Deckblatt dieses Berichts gezeigte Abbildung zeigt die Struktur des Konferenzmodells und seine Komponenten in Form einer Momentaufnahme einer gerade aktiven Konferenz.

Innerhalb einer solchen Konferenz existieren mehrere ausgezeichnete Rollen, die einzelnen Teilnehmern zugeordnet sind. Die Rolle des Konferenzleiters muss immer von genau einem Teilnehmer übernommen werden, während andere Rollen, beispielsweise die des Protokollführers, optional sind. Alle Teilnehmer sind über ihren Namen und, sofern ihnen eine Rolle zugeordnet ist, zusätzlich auch über die Bezeichnung der Rolle identifizierbar.

Alle einer Konferenz Teilnehmer haben sowohl Zugriff auf ihren *lokalen Datenraum* (private work space) als auch auf einen *gemeinsamen Datenraum* (shared work space), in dem sich die für alle Teilnehmer zugreifbaren Daten befinden. Daten im gemeinsamen Datenraum sind entweder Dokumente, die gemeinsam editiert werden können, oder flüchtige Kommunikationseinheiten wie Sprache, Daten in Dialogfenstern, etc., während der lokale Datenraum private Dokumente bzw. das jeweils lokale Dateisystem umfasst.

Interaktionen zwischen den Teilnehmern beruhen auf einer gemeinsamen "Sicht" auf den gemeinsamen Daten-/Sprachraum und Modifikationsmöglichkeiten auf den gemeinsamen Daten

durch alle Teilnehmer (WYSIWIS, “What you see is what I see” [Stefik 86]).<sup>1</sup> Im Einzelfall können jedoch die Teilnehmer auch gemeinsame Daten lokal auf verschiedene Weise repräsentieren (beispielsweise können Reservationen anderer Teilnehmer in einem gemeinsam bearbeiteten Dokument von jedem Teilnehmer unabhängig “offen” oder in ikonisierter Form dargestellt werden).

Eine wichtige Komponente des Konferenzmodells ist die Bereitstellung von Zugriffskontrollmechanismen auf gemeinsame Ressourcen. Hierbei kann davon ausgegangen werden, dass sowohl Objekte in einem Dokument bzw. ganze Dokumente als auch Kommunikationskanäle als Ressourcen angesehen werden, die jeweils “unteilbar”, d.h. von jeweils einem Benutzer reservierbar oder allen Benutzern ohne explizite Zugriffskontrolle zugänglich sind, wobei der Zuteilungsmechanismus für die jeweilige Ressource für verschiedene Typen von Ressource differieren kann.

## **1.2. Die technische Realisierung des Prototyps**

Der auf UNIX-Rechnern und unter dem X-Windows-Fenstersystem implementierte MultimETH-Prototyp besteht aus einer Menge von Hardware-Komponenten, von kooperierenden Prozessen und zugehörigen Kommunikationseinrichtungen, die nachstehend im Überblick abgebildet und erläutert werden. Kapitel 2 dieses Berichts geht im Detail auf die verschiedenen Komponenten des Systems ein bzw. verweist auf die entsprechenden studentischen Arbeiten. Die Berichte zu diesen Arbeiten sind im Literaturverzeichnis erwähnt und werden zusammen mit dem vorliegenden Bericht abgegeben.

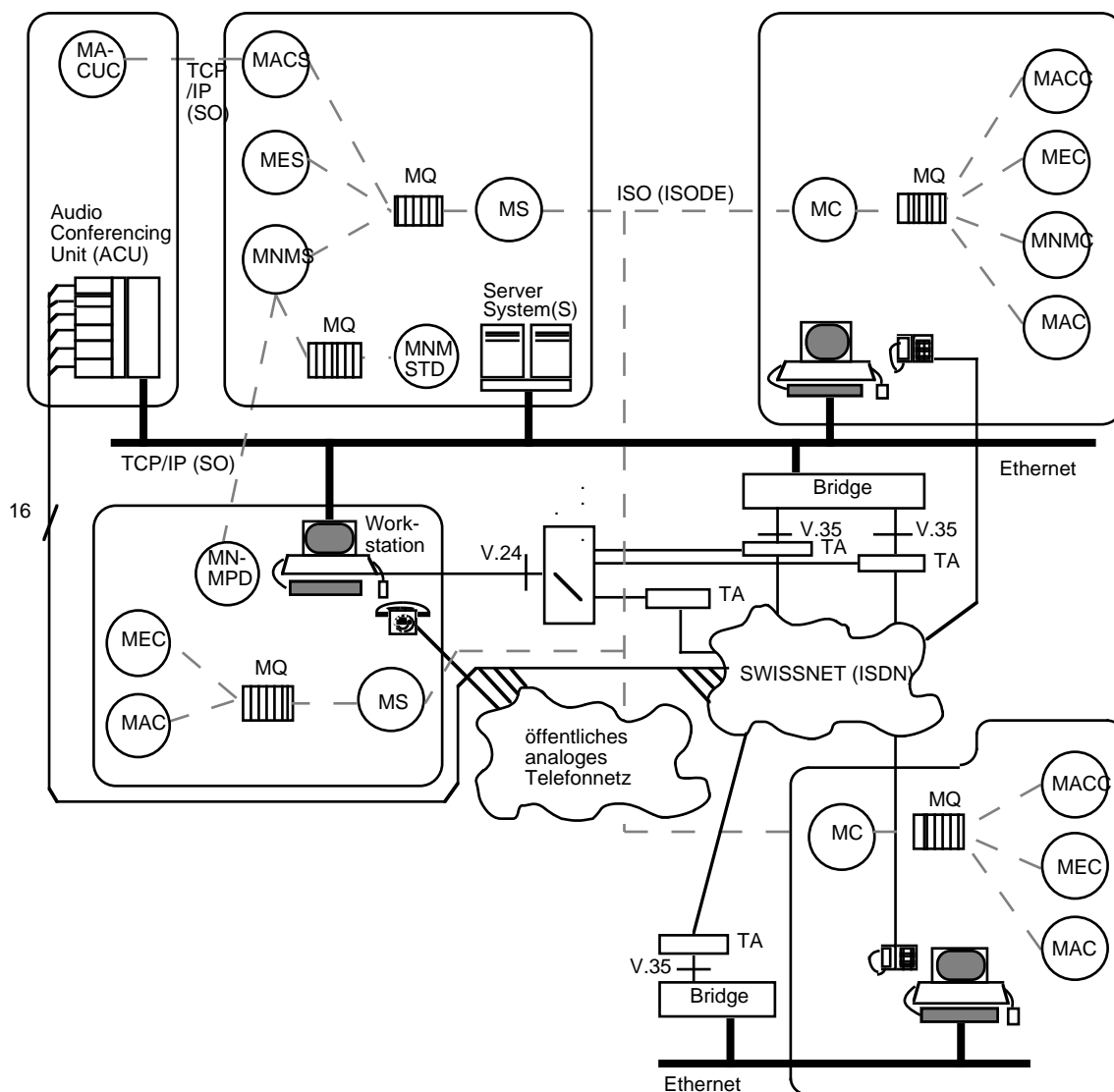
In der nachfolgenden Abbildung 1 kennzeichnen durchgezogene Linien physikalische Verbindungen (Ethernet, Swissnet, V.24- und V.35-Leitungen, analoges Telefon), während nicht durchgezogene Linien konzeptionell darüberliegende Dienste verschiedenen Typs (OSI-Dienste, TCP/IP, Message Queue) darstellen, welche mit Hilfe der physikalischen Verbindungen oder über lokale Interprozess-Kommunikation realisiert werden. Kreise bezeichnen Prozesse auf den beteiligten Rechnersystemen, alle anderen Komponenten sind entsprechend gekennzeichnet.

Das “Herzstück” des MultimETH-Systems besteht aus einem Server-System, auf welchem die verschiedenen Server-Prozesse (Daten-Konferenzsystem, Sprach-Konferenzsystem, Editor, Netzwerk-Konfiguration und -Management) angesiedelt sind. Der MultimETH-Server als zentrale Instanz der Server-Architektur kommuniziert einerseits über die OSI-Dienstelemente ROSE und ACSE (über, möglicherweise via ISDN miteinander verbundene, LANs) mit den

---

<sup>1</sup> Durch die Einbindung von Sprachinformation in das Konferenzsystem muss genau genommen nicht mehr nur von einem Konzept des “What you see is what I see”, sondern eher von einem “What you perceive is what I perceive” Konzept gesprochen werden.

MultimETH-Klienten-Systemen, andererseits über eine "UNIX System V Message Queue" mit den anderen Server-Prozessen auf dem Server-System (siehe [Stevens 90]).



- Glossar:
- SO: Socket
  - MQ: Message Queue
  - MS: MultimETH (Basis) Server
  - MC: MultimETH (Basis) Client
  - MAC: MultimETH Andrew Client
  - MES: MultimETH Editor Server
  - MEC: MultimETH Editor Client
  - MACS: MultimETH Audio Conferencing Server
  - MACC: MultimETH Audio Conferencing Client
  - MACUC: MultimETH Audio Conferencing Unit Controller
  - MNMS: MultimETH Network Management Server
  - MNMC: MultimETH Network Management Client
  - MNMPD: MultimETH Network Management Port Daemon
  - MNMSTD: MultimETH Network Management SNMP Trap Daemon

Abbildung 1: Übersicht über den Aufbau von MultimETH

Die Audio-Konferenz-Einheit gehört logisch zum Server-System, ist aber als eine eigenständige Hardware-Einheit realisiert, welche ebenfalls über ein LAN mittels TCP/IP mit dem Server-System verbunden ist. Sie wird vom Server-System mittels eines eigenen Server-Prozesses

konfiguriert und überwacht. Pro MultimETH-Server kann im jetzigen Prototyp aus Komplexitätsgründen jeweils nur eine Konferenz aktiv sein.

In einem Klienten-System sind als zentrale Komponente ein MultimETH-Klienten-Prozess sowie weitere Klienten-Prozesse für das Konferenzmanagement, das gemeinsame Editieren, die Teilnahme an der Sprach-Konferenz sowie ggf. für das Netzwerk-Management vorhanden. Analog zum Server-System kommunizieren alle Klienten-Prozesse über eine Message Queue mit dem MultimETH-Klientenprozess. Damit ist festgelegt, dass die Kommunikation zwischen allen Servern und ihren zugehörigen Klienten (mit Ausnahme des Netzwerk-Managements und der Ansteuerung der Audio-Konferenz-Einheit) immer über den MultimETH-Server und den MultimETH-Klienten, d.h. über eine OSI-Verbindung, führt.

Für die Konfiguration der LAN-Kopplung über ISDN verfügt entweder das Server-System, oder, wie in der Abbildung gezeigt, ein Klienten-System, über zusätzliche Hard- und Software. Einerseits muss eine Reihe von Terminal-Adaptern und ISDN/LAN-Bridges vorhanden sein (deren Kontroll- und Konfigurationseingänge über einen programmierbaren V.24-Switch an einen Daten-Eingang des Arbeitsplatzrechners angeschlossen sind), andererseits muss ein entsprechendes Software-Paket die Konfiguration sowie die Überwachung der LAN-Kopplung erlauben. Die Netzwerk-Überwachung via SNMP fügt dem MultimETH-System zusätzliche Prozesse für das Bereitstellen bzw. das Entgegennehmen und Auswerten der entsprechenden Information hinzu. Diese Information wird dem Netzwerk-Management-System über eine zusätzliche Message Queue zur Verfügung gestellt.

Im folgenden Kapitel werden die einzelnen funktionalen Einheiten detaillierter beschrieben. Für ein genaues Verständnis ihrer Funktionsweise sowie die details der Realisierung ist jedoch die Lektüre der Dokumentation der entsprechenden studentischen Arbeiten unerlässlich.

## 2. Beschreibung der funktionalen Einheiten

### 2.1. Kommunikationssystem

Das Kommunikationssystem von MultimETH basiert auf der Verwendung eines OSI-Protokoll-Stacks im allgemeinen, und der Verwendung zweier Dienstelemente der Schicht 7, namentlich ROSE (Remote Operations Service Element, ISO 9072) und ACSE (Association Control Service Element, ISO 8649/8650), im besonderen. Als Topologie wurde aus Performance- und Komplexitätsgründen eine sternförmige Architektur, d.h. die Verbindung mehrerer Klienten über einen zentralen Server-Prozess, vorgesehen. Hierzu wurde ein erster Prototyp der "protocol engine" im Jahr 1988 entworfen und unter Verwendung des ISODE-Protokoll-Programmiersystems realisiert [Imfeld 89]. Das ISODE-Paket erlaubt es, die Operationen sowie die zugehörigen Datenstrukturen für die Übergabe von Argumenten, Fehler- und Resultatwerten in der normierten Beschreibungssprache ASN.1 (Abstract Syntax Notation One, ISO 8824/8825) zu spezifizieren. Mittels spezieller ISODE-Compiler ist es daraufhin möglich, die entsprechenden Programmfragmente für die Darstellung der Datentypen und Operationen in der Programmiersprache "C" sowie die Prozeduren für das Generieren, das Entgegennehmen sowie das syntaktische Überprüfen der entsprechenden Protokolldateneinheiten zu generieren. Eine entsprechende Definition eines Moduls mit einer Operation, einem Argumenttyp, einem Resultattyp sowie drei Fehlertypen kann z.B. die folgende Form haben:

```
MultimETH DEFINITIONS ::=
BEGIN

startConf          OPERATION
                   ARGUMENT User
                   RESULT Confirmation
                   ERRORS { noPermission, tooManyUsers, wrongState }
                   ::= 1

tooManyUsers       ERROR ::= 1
wrongState         ERROR ::= 2
noPermission       ERROR ::= 11

Confirmation       ::= NULL

User ::= [APPLICATION 1]
      SET {
        host          [0] IA5String OPTIONAL,
        name          [1] IA5String,
        password      [2] IA5String,
        fullname      [3] IA5String OPTIONAL,
        organis       [4] IA5String OPTIONAL,
        telephone     [5] IA5String OPTIONAL,
        mailaddress   [6] IA5String OPTIONAL,
        reason        [7] IA5String OPTIONAL,
        prefname      [8] IA5String OPTIONAL }

END
```

Das initiale Protokoll und die zugehörige Implementation des Prototyp-Systems wurde in weiterführenden Arbeiten stark erweitert und umgestaltet (siehe insbesondere [Lubich 89]). Da-

mit wurde der von ISODE zur Verfügung gestellte Protokollstack wie in Abbildung 2 gezeigt um ein eigentliches, in ASN.1 formal beschriebenes “Multimedia Specific Application Service Element” erweitert.

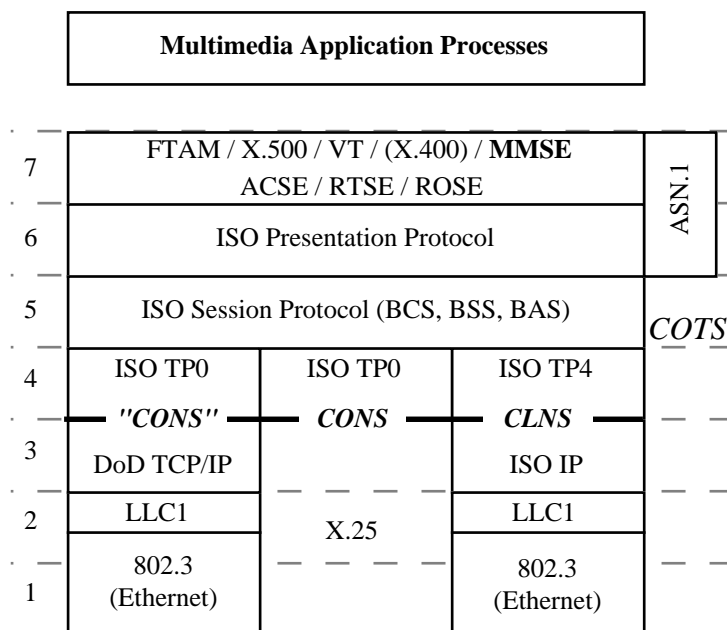


Abbildung 2: Der MultimETH-Protokollstack

Zum jetzigen Zeitpunkt umfasst das Protokoll 73 Operationen, 18 Fehlertypen und 62 Datentyp-Definitionen für die Argument- oder Resultatübermittlung. Während das ISODE-Paket das Aufrufen von entfernten Operationen eines Server-Prozesses durch einen Klientenprozess bereits gut unterstützte, musste die Protokollmaschine noch wesentlich erweitert werden, um:

- a) die gleichzeitige Verbindung mehrerer Klienten zu einem Server zu ermöglichen, und
- b) auch asynchrone Aufrufe von Operationen durch den Server bei einem oder mehreren Klienten zuzulassen (da die Klienten über den Server kommunizieren, muss auch der Server zu jedem Zeitpunkt eine Operation, z.B. die Anzeige einer Mitteilung, bei einem Klienten auslösen können).

Da die vorhandenen Software-Werkzeuge es lediglich gestatteten, die statischen Teile des MultimETH-Protokolls, also die Operationen und Datentypen, darzustellen und entsprechenden Code zu generieren, musste die eigentliche “protocol engine” von Hand entworfen und implementiert werden. Aufgrund der Komplexität des Protokolls war es jedoch nicht möglich, einen formalen Beweis für die Korrektheit des Entwurfs bzw. der Implementation zu liefern. Aus diesem Grund wurde in zwei weiteren studentischen Arbeiten ein Testwerkzeug erstellt und angewendet, welches eine “random state exploration” des MultimETH-Prototyps ermöglichte (siehe [West 87]). Hierbei wird aus einer Ablauf-Spezifikation für alle möglichen Operationen ein Testsystem generiert, welches einen Benutzer des Systems simuliert und zufällig Operationen auswählt und auf das getestete System anwendet. Das Testsystem vergleicht für jede Opera-

tion das erwartete mit dem tatsächlich beobachteten Verhalten und meldet Inkonsistenzen. Durch die Zufälligkeit der Auswahl der Operationen sowie durch eine entsprechend häufige Verwendung des Testsystems (mehrere tausend Operationen in einem Testlauf) besteht eine grosse Wahrscheinlichkeit, dass auch komplexe Fehlerzustände mindestens einmal erreicht und angezeigt werden können. Die Anwendung dieses eigens entwickelten Validierungssystems führte zur Aufdeckung und Behebung einer Reihe von Fehlern im MultimETH-System. Die entsprechende Diplomarbeit wurde 1991 mit einem Preis des Schweizer Automatik Pool (SAP) ausgezeichnet (siehe [Lichtin 90]).

## **2.2. Benutzerschnittstelle und Konferenzverwaltungssystem**

Der ursprüngliche MultimETH-Prototyp verfügte über zwei Benutzerschnittstellen, eine rein textuelle Basis-Schnittstelle mit zeilenorientierter Eingabe sowie eine auf dem proprietären Fenstersystem SunView der Firma Sun Microsystems aufbauende graphische Schnittstelle [Helbing 89]. Die textuelle Schnittstelle diente dabei einerseits der (eingeschränkten) Interaktion mit dem System über nicht grafikfähige Terminals, andererseits wurde sie als Ein-/Ausgabemedium für den in 2.1 beschriebenen Protokollvalidator verwendet.

Bei der für das ZBF-Projekt 224 Z durchgeführten Erweiterung und Neu-Konzipierung des MultimETH-Systems wurde die Verwendung der SunView-basierten Benutzerschnittstelle zugunsten einer portableren und auf einer grösseren Palette von Arbeitsplatzrechnern zur Verfügung stehenden X-Window-basierten Benutzerschnittstelle aufgegeben. Hierbei wurde vorgesehen, das MultimETH-System von vorneherein so offen zu gestalten, dass auch die Einbindung neuer Benutzerschnittstellen zu einem späteren Zeitpunkt möglich sein sollte. Daher wurde die Benutzerschnittstelle als eigenständiger Prozess konzipiert, die über eine bidirektional verwendete Message Queue mit dem MultimETH-Klientenprozess kommuniziert. Hierfür wurde die textuelle Benutzerschnittstelle so ausgebaut, dass sie auch multimediale Daten verarbeiten, d.h. in textueller Form darstellen und weiterleiten kann. Damit ist es möglich, eine neue Benutzerschnittstelle für MultimETH ohne wesentliche Änderungen am MultimETH-Basissystem zu realisieren. Zudem können auf diese Weise auch Teile der Funktionalität in weitere Prozesse ausgelagert werden (wie z.B. der Editor-Klient und der Management-Klient), ohne dass dies aufwendige Änderungen im MultimETH-Basissystem erfordert. Abbildung 3 zeigt diesen Zusammenhang zwischen den Benutzerschnittstellen-Prozessen und dem MultimETH-Basissystem.

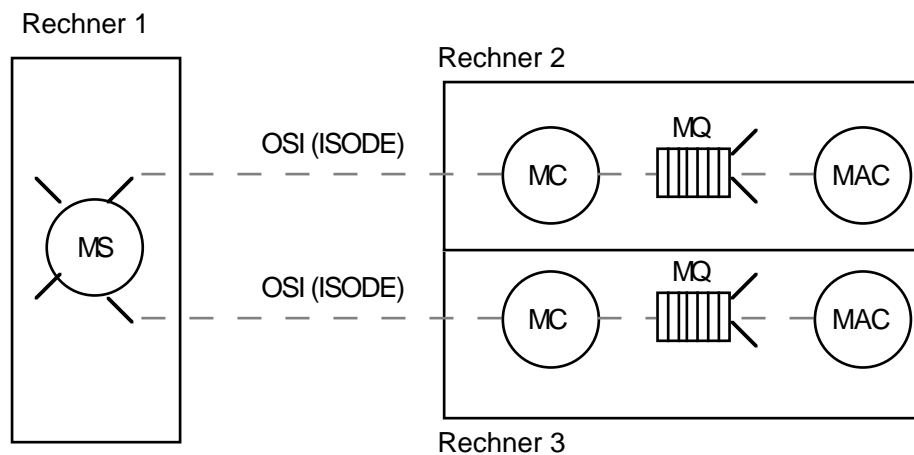


Abbildung 3: Zusammenhang zwischen den MultimETH-Prozessen

Während dieses Vorgehen beim MultimETH-Server keine Änderungen bedingte und die Änderungen beim MultimETH-Klienten auf das Umstellen der textuellen Kommunikation über eine Message Queue anstelle von Standard-Input/Output des UNIX-Prozesses beschränkt waren, ergab sich ein prinzipielles Problem auf Seite des Benutzerschnittstellen-Prozesses.

Fast alle der heute verfügbaren und in [Aebi 91] untersuchten graphischen Benutzerschnittstellen sind im Bezug auf ihre Ereignisbehandlung benutzergesteuert. Dies bedeutet, dass Anwendungen, welche eine entsprechende Benutzerschnittstelle beinhalten, keine Kontrolle über den Prozessablauf haben. Stattdessen wird beim Start der Anwendungen die Kontrolle an eine Endlos-Schleife (Event Loop) im Fenstersystem übergeben. Prozeduren der Anwendung können in dieser Ereignis-Behandlung als “Callback”-Prozeduren installiert werden, die dann ausgeführt werden, wenn ein vorher an diese Prozedur gebundenes Ereignis auftritt. Selektiert der Benutzer beispielsweise ein Objekt, oder betätigt er die Maustaste, wird dieses Ereignis vom Fenstersystem bearbeitet und ggf. eine in der Anwendung für diesen Fall definierte Prozedur ausgeführt. Die eigentliche Applikation entspricht hierbei also eher einer “Toolbox”, die eine Reihe von Prozeduren bereitstellt. Sollen nun asynchron Daten von einem anderen Prozess oder über eine Netzwerk-Verbindung empfangen werden, muss die Anwendung den Event Loop so erweitern, dass nicht nur ein Warten auf Benutzerinteraktionen, sondern auch auf die Ankunft von “externen” Daten über eine entsprechende Schnittstelle im Event Loop zyklisch oder ereignisgesteuert ausgeführt wird (z.B. wenn ein Server einem Klientenprozess eine Operation zur Anzeige einer Dialogbox sendet).

Die untersuchten Fenstersysteme behandeln ihren Event Loop jedoch weitgehend intern, so dass die Definition und Verwendung neuer Event-Typen o.ä. nicht möglich ist. Die einzige vorhandene Schnittstelle nach aussen bildet in den meisten Fenstersystemen die File-Deskriptor-Schnittstelle von UNIX, hinter der auch Sockets, d.h. Kommunikationsendpunkte für die lokale und entfernte Datenkommunikation, angelegt werden können. Werden Daten auf einer solchen Socket-Verbindung empfangen, wird ein entsprechender Event erzeugt, und eine zuvor

vom Programmierer definierte Callback-Prozedur der Anwendung aufgerufen. Diese Methode ist jedoch nicht brauchbar für jede Art der Kommunikation, die nicht über die Socket-Schnittstelle stattfinden, wie z.B. die in MultimETH verwendete Kommunikation über Message Queues und OSI-Dienstzugriffspunkte.

Aus den in [Aebi 91] skizzierten Lösungen für dieses Problem wurde für MultimETH die Variante der Verwendung eines Zeitgebers (Timer) ausgewählt. Die meisten Fenstersysteme stellen einen Timer zur Verfügung, der beim Aufruf auf eine initiale Zeitspanne initialisiert und dann vom Fenstersystem kontinuierlich dekrementiert wird. Erreicht der Timer den Nullwert, so wird ein entsprechender Event ausgelöst. Die Behandlungsprozedur für diesen Event enthält dann die Überprüfung der Kommunikationsschnittstelle und die allfällige Bearbeitung der eingetroffenen Daten. Für MultimETH wurde ein Timer vorgesehen, welcher einmal beim Systemstart initialisiert wird. Fällt der Timer-Wert auf Null, so wird einerseits die Message Queue oder OSI-Kommunikationsschnittstelle auf neu angekommene Daten überprüft und ggf. eine entsprechende Verarbeitung vorgenommen, andererseits wird als letzte Aktion der Timer-Behandlung der Timer wieder neu gestartet. Auf diese Weise kann durch entsprechende Einstellung des Timers ein guter Mittelweg zwischen "busy waiting" und einem event-gesteuerten Mechanismus beschränkt werden.

Beim Ersatz der SunView-Benutzerschnittstelle durch ein X-Windows-basierte Lösung wurde die von der Carnegie Mellon Universität entwickelte Andrew-Fensterumgebung ausgewählt, welche eine objektorientierte Programmierschnittstelle für multimediale Objekte, einen "Interface Builder" usw. zur Verfügung stellt (siehe auch Bericht zum Projekt). Aufbauend auf Andrew wurde zunächst der oben beschriebene Timer-Mechanismus als Grundlage für einen Benutzerschnittstellen-Prozess implementiert. Als Folgearbeit wurde dann die Kommunikation zwischen MultimETH-Klienten-Prozess und Benutzerschnittstellen-Prozess über eine Message Queue (siehe auch Abbildung 3) sowie die graphische Benutzerschnittstelle des gesamten MultimETH-Systems entworfen und implementiert.

### **2.3. Dokumentenverwaltung und -konvertierung**

Zur Verwaltung von Dokumenten im privaten Arbeitsbereich eines Benutzers bzw. im gemeinsamen Arbeitsbereich einer Konferenz wurde für MultimETH ein eigenes Navigations- und Bearbeitungswerkzeug (Browser) entworfen und implementiert ([Caronni 93]). Mittels dieses Werkzeugs kann ein Benutzer im jeweils baum-strukturierten Dokumentenraum sowohl auf dem Klienten- als auch auf dem Server-System seine Position anzeigen und verändern, Listen von Dokumenten und ihre zugehörigen Attributen (Zugriffsrechte, Besitzer usw.) anzeigen, sowie ein Dokument oder eine Gruppe von Dokumenten auswählen, um dann eine Operation (Löschen, Kopieren usw.) darauf auszuführen. Der Browser stellt den Dokumentenraum in einem separaten Fenster der Benutzerschnittstelle so dar, dass jeweils zwei Sichten auf den Dokumentenraum verfügbar sind. Da beide Sichten (= Unterfenster) jeweils unabhängig

voneinander manipuliert werden können, können z.B. zwei verschiedene Verzeichnisse im privaten oder gemeinsamen Arbeitsbereich oder aber jeweils eine Sicht auf den gemeinsamen und eine Sicht auf den gemeinsamen Arbeitsbereich angezeigt werden. Abbildung 4 zeigt beispielsweise vor dem Hintergrund des MultimETH-Hauptfensters die Verwendung des Browsers in einer Konferenz, um sowohl den Inhalt des privaten, als auch des gemeinsamen Arbeitsbereiches in Kurzform anzuzeigen.

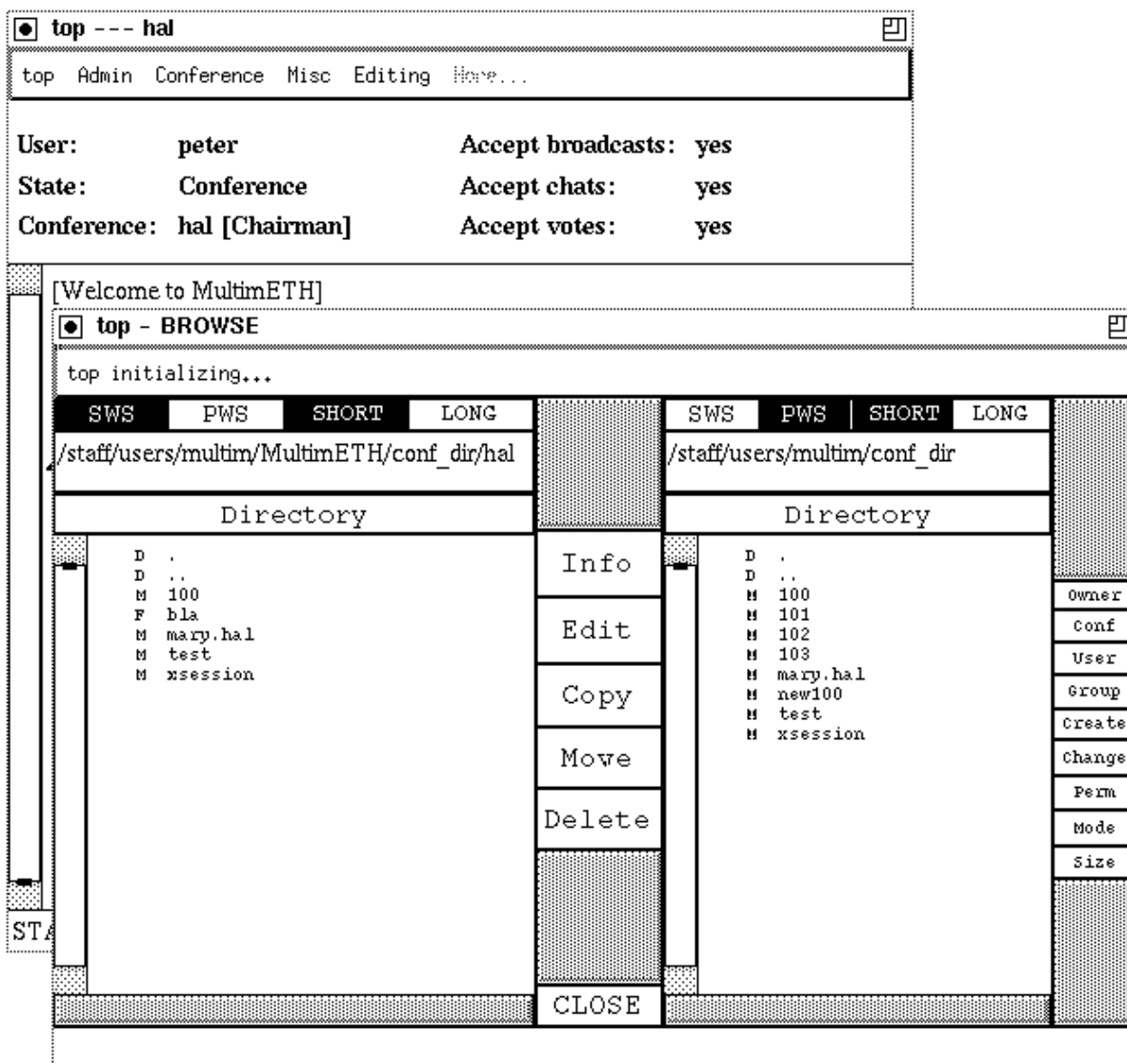


Abbildung 4: Der MultimETH-Browser

Durch die Abbildung des MultimETH-Dokumentenraums auf UNIX-Dateisysteme ergab sich die zusätzliche Schwierigkeit der unterschiedlichen Semantik der MultimETH-Zugriffsrechte und der UNIX-Zugriffsrechte auf Dateien/Directories bzw. der Einführung von Dateitypen (MultimETH-Dokument, Text-Datei, Directory, usw). Aus diesem Grund ist es im Browser nötig, die entsprechenden Attribute getrennt voneinander anzeigen und auswerten zu können. Bestimmte Operationen sind dementsprechend nur auf bestimmten Dateitypen möglich, während andere Operationen auf allen Dateitypen zulässig sind. Im Fall des lokalen, privaten Arbeitsbereiches werden die MultimETH-Attribute für den Zugriff auf ein Dokument nicht

überprüft, es müssen jedoch beispielsweise zum Löschen eines Dokuments im lokalen Arbeitsbereich die entsprechende UNIX-Zugriffsrechte vorhanden sein. Die zu einem Dokument gehörigen Attribute und ihre Semantik werden im folgenden Abschnitt über den MultimETH-Editor erläutert.

Zusätzlich zur Einbindung des unter MultimETH verwendeten UNIX-Dateisystems und die daraus resultierende Öffnung von MultimETH "nach aussen" wurde im Projektzeitraum auch ein Konverter zwischen dem MultimETH-Dokumentenformat und dem RTF-Dokumentenformat entworfen und implementiert ([Ruess 93]). Wie bereits im Abschlussbericht dargelegt, war die zunächst vorgesehene Konvertierung zwischen dem MultimETH-Dokumentenformat und ODA aus Gründen der übermässigen Komplexität sowie der Nicht-Verfügbarkeit von ODA-Produkten nicht möglich und auch nicht unbedingt sinnvoll.

Bei der Realisierung des MultimETH/RTF-Konverters lagen die wesentlichen Schwierigkeiten:

- a) in der nicht vollständig verfügbaren Beschreibung bzw. der kontinuierlichen Weiterentwicklung von RTF,
- b) in der Abbildung bzw. Auslassung von wechselseitig nicht oder nicht in dieser Form vorhandenen Objekten,
- c) in der verschiedenen Mächtigkeit der Dokumentenformat-Beschreibung sowie
- c) in der verschiedenen Codierung multimedialer Objekte.

Aufgrund dieser Schwierigkeiten, die jedoch typisch für die Konvertierung komplexer Dokumentenformate sind, wurde der MultimETH/Andrew-Konverter soweit entwickelt, dass eine bidirektionale Konvertierung einfacher Dokumente mit textuellem Inhalt, Tabellen, Rastergraphiken, Stilinformationen usw. unterstützt wird. Hierbei wird die Konvertierung so transparent wie möglich gehalten. Nachfolgend werden drei typische Probleme bei der Konvertierung und die jeweils zugehörige Lösungen vorgestellt, um die Komplexität und die generellen Schwierigkeiten einer solchen Konvertierung aufzuzeigen:

- 1) In MultimETH ist es möglich, dass Reservierungen in Dokumenten auch zwischen Editingsitzungen im Dokument erhalten bleiben. Hierfür wird die Reservierungsinformation im Dokument gespeichert. Da RTF-verarbeitende Einzelplatz-Editoren wie z.B. Microsoft Word (als Referenzversion diente Version 5.0 auf Macintosh) Reservierungen prinzipiell nicht kennen, kann bei der Konvertierung entweder diese Reservierungsinformation gelöscht, oder in ein anderes Objekt konvertiert werden, so dass bei einer allfälligen Rück-Konvertierung diese Information wieder in ein korrekt arbeitendes Reservationsobjekt umgewandelt werden kann. Im Fall des MultimETH-Konverters bleibt die Reservierungsinformation erhalten, indem sie in einem Microsoft Word-Dokument teilweise "versteckt", teilweise in tabellarischer Form angezeigt wird. Bei einer Rück-Konvertierung

wird diese Information wieder korrekt in ein Reservationsobjekt konvertiert, sofern sie nicht im RTF-fähigen Editor modifiziert worden ist.

- 2) Angaben über logische Zugehörigkeit und Darstellung von Objekten, z.B. Kennzeichnung einer Zeile als "Überschrift erster Ordnung" oder Darstellung eines Wortes in einer Zeile in Kursivschrift, können entweder in Form einer verborgenen Einleitung zu einem Dokument ("stylesheet") oder verstreut an den jeweiligen relevanten Orten im Dokument ("ab jetzt gesamter Text im Kursivdruck") vorliegen. Zudem kann solche Information entweder absolut ("ab jetzt Schriftgröße 14 verwenden") oder relativ ("ab jetzt Schriftgröße um 2 Punkte erhöhen") vorkommen. Werden nun Objekte z.B. mittels "cut & paste"-Operationen in oder zwischen Dokumenten verschoben, muss diese Information gesucht und ggf. ebenfalls kopiert werden. Zu Schwierigkeiten kommt es immer dann, wenn die beiden beteiligten Dokumentenformate, wie im Fall von MultimETH und RTF, verschiedene Strategien verwenden werden. In der Implementierung des Konverters wurde grosse Mühe auf die korrekte, bi-direktionale Konvertierung von Stilelementen gelegt. Im Einzelfall kann zwar Stilinformation verloren gehen, jedoch ist sichergestellt, dass keine Inhaltsobjekte des Dokumentes verloren geht.
- 3) Die interne Darstellung von Raster-Graphiken (Bitmaps) bei MultimETH basiert auf einer Bitmap-Codierung der verwendeten Andrew-Programmierungsumgebung. Demgegenüber beruht die Darstellung von Graphiken in im RTF-interpretierenden Editor MsWord auf dem Macintosh auf dem Macintosh-spezifischen QuickDraw-Format, welches nicht die eigentlichen Graphikdaten, sondern Graphik-Direktiven zur Darstellung der Graphik enthält. Jedoch ist es möglich, die MultimETH-Darstellung ebenfalls in RTF darzustellen, da RTF auch diese Form der Datenrepräsentation unterstützt. Der MultimETH-Konverter erlaubt also die Übernahme von Bitmaps von MultimETH z.B. in MsWord, bietet jedoch keine Konvertierung in die Gegenrichtung. Da bei Konvertierung eines multimedialen Dokuments somit Daten in einer Konvertierungsrichtung verloren gehen können, wird bei der Konvertierung des Dokuments ein textueller Hinweis auf diesem Umstand anstelle der eigentlichen Graphik in das Dokument integriert.

Der MultimETH-Dokumentenkonverter kann somit nur eine eingeschränkte Funktionalität zur Verfügung stellen. Obwohl Erweiterungen und Ergänzungen des Konverters sicher denkbar sind, werden sich die grundlegenden Schwierigkeiten bei der Dokumentenkonvertierung nicht vollständig umgehen lassen. Der vorliegende Konverter ist jedoch ein wichtiges Element der Anbindung von MultimETH an die "Aussenwelt". Insbesondere bietet er ausreichende Funktionalität für die intendierte Benutzung von MultimETH zur gemeinsamen Erstellung von Dokumenten, zum Importieren von einfachen externen Dokumenten, sowie zum "Exportieren" von MultimETH-Dokumenten an Editoren, die besser als MultimETH für ein qualitativ hochwertiges Schlusslayout und "Desktop Publishing" geeignet sind.

## 2.4. Multimedia-Mehrbenutzer-Editor

Der im Rahmen des MultimETH-Projektes entworfene und realisierte Mehrbenutzer-Multimedia-Editor ist ein eigenes verteiltes System, bestehend aus einem Editor-Server und einem Editor-Klienten. Der Editor-Server wird dabei auf dem MultimETH-Server-System angesiedelt, während auf jedem MultimETH-Klienten-System ein Editor-Klient bereitgestellt werden kann. Sowohl der Editor-Server als auch der Editor-Klient sind als eigenständige Prozesse realisiert, beruhen jedoch auf demselben Programm, welches im Server- oder im Klienten-Modus gestartet wird. Die Kommunikation zwischen Editor-Server und Editor-Klienten wird transparent über das MultimETH-OSI-Transportsystem abgewickelt. Hierfür wurden die ROSE-Spezifikation des MultimETH-Protokolls um je eine generische Editor-Operation vom Klienten zum Server und vom Server zum Klient erweitert. Die editor-spezifischen Daten werden vom MultimETH-Server und MultimETH-Klienten transparent behandelt, d.h. uninterpretiert in der jeweiligen Message Queue abgelegt. Auf dem Server-System wurde zusätzlich ein lokales Kommunikationsprotokoll zwischen MultimETH-Server und Editor-Server entworfen und implementiert, welches ebenfalls die Message Queue verwendet. Abbildung 5 zeigt diese Erweiterung des MultimETH-Systems:

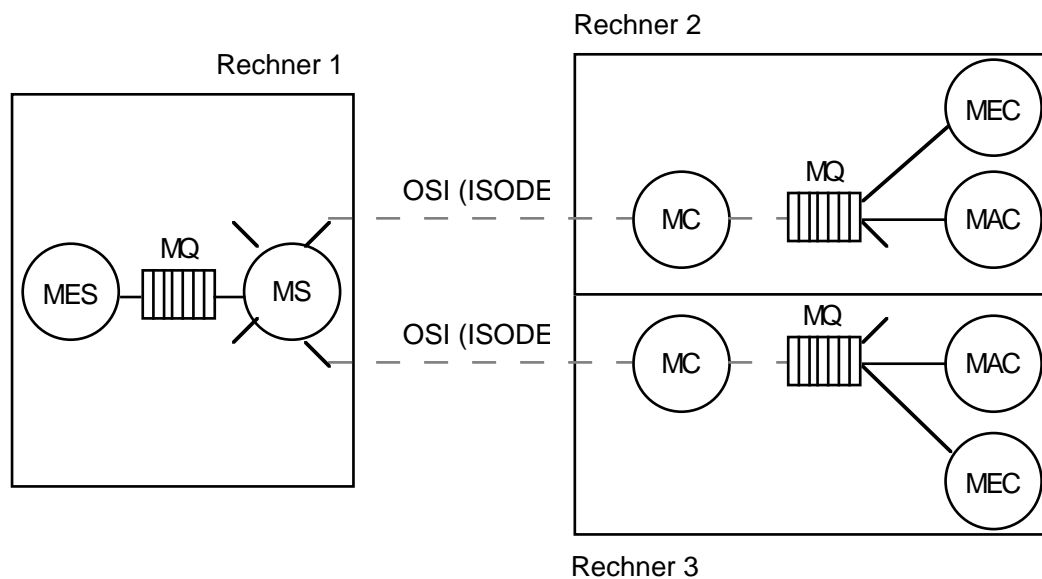


Abbildung 5: Integration der Editor-Kommunikation in das MultimETH-System

Da der MultimETH-Server nun ebenfalls zwei Schnittstellen zu überwachen hat (OSI-Schnittstelle und Message Queue), musste der ursprüngliche Server-Code so abgeändert werden, dass ein abwechselndes Überprüfen beider Schnittstellen (anstelle des in ISODE ursprünglich vorgesehenen ausschliesslichen Wartens auf das Eintreffen einer Remote Operation) stattfand.

Der eigentliche Editor wurde aus software-technischen Gründen zunächst als eigenständige Server/Klient-Implementation über TCP/IP realisiert (siehe [Almesberger 92] und [Wild 92]), und erst zu einem späteren Zeitpunkt auf die Kommunikation über Message Queues umgestellt. Er

ist jedoch weitgehend unabhängig vom darunterliegenden Kommunikations-Subsystem. Das Kommunikations-Modul des Editors beruht auf einem eigens entwickelten "Remote Method Call"-Paket, welches es erlaubt, in einer objektorientierten Umgebung, wie sie der Editor darstellt, Methoden (d.h. nach aussen sichtbare Prozeduren) auch über Prozess- und Systemgrenzen hinweg aufzurufen. Zudem ist es leicht möglich, den Aufruf neuer Methoden in das Kommunikations-Modul zu integrieren. Methoden-Aufrufe werden in Form von Zeichenketten codiert und über ein einfaches Message Passing der jeweils anderen Instanz oder einer Reihe anderer Instanzen geschickt. Auch Resultatwerte oder Fehler werden auf diese Weise versandt.

Die für den Benutzer sichtbare Funktionalität ist eng an den mit dem Andrew-System ausgelieferten multimedialen Editor "ez" (meist wie "easy" ausgesprochen) angelehnt. Ebenso ist das Dokumentenmodell von MultimETH wesentlich durch das Andrew-Dokumentenmodell beeinflusst. Demzufolge war auch die Entscheidung, das ursprüngliche, baumstrukturierte Dokumentenmodell durch ein objekt-orientiertes Modell zu ersetzen, sowohl durch theoretische als auch durch pragmatische Überlegungen bezüglich der vorhandenen Softwarebasis beeinflusst.

Ein wesentlicher Gesichtspunkt bei der Definition des Dokumentenmodells bildete die Realisierung des gleichzeitigen Mehrbenutzer-Zugriffs bzw. dessen Synchronisation. Im ursprünglich entworfenen Dokumentenmodell für MultimETH wurde von einem baum-strukturierten Dokument ausgegangen, um durch explizites Reservieren von Unterbäumen eine klare Trennung von aktiven Editier-Bereichen zu gewährleisten (siehe [Lubich 89]). Diese hierarchische Form der Reservierung über Kapitel, Unter-Kapitel, Abschnitte usw. entsprach jedoch nicht dem durch die Verwendung der objektorientierten Andrew-Programmierungsumgebung gegebenen Dokumenten- und Objekt-Modell. Wie in [Almesberger 92] dargelegt, wurde daher ein auf dem Reservieren von kontinuierlichen Regionen basierendes Verfahren entworfen und implementiert. Hierbei wird eine vom Benutzer ausgewählte beliebig grosse Region aus dem Originaldokument entfernt und als neues Objekt ("jnote") mit entsprechend veränderten Attributen wieder in das Dokument eingefügt. Dieses Vorgehen der Behandlung der Reservation als separates Objekt erleichtert auch die Darstellung eines solchen Objektes im Editor als separates Fenster im Dokument.

Eine Reservation wird jeweils durch Selektieren eines Dokumentteils und Senden einer Reservierungsanfrage an den Editor-Server beantragt. Der Server prüft die Anfrage auf Konflikte und verweigert ggf. die Ausführung. Im Erfolgsfall wird das Original-Dokument beim Server modifiziert, d.h. die selektierte Region wird aus dem Dokument entfernt, und innerhalb eines Reservationsobjekts wieder in das Dokument eingefügt. Zudem wird ein "Remote Method Call" an alle aktiven Editor-Klienten gesendet, der diese Modifikation bekanntmacht. Die bei allen Editor-Klienten gespeicherten Arbeitskopien des Dokuments werden entsprechend aktualisiert und angezeigt. Zu einem Zeitpunkt können beliebig viele Reservationen in einem Dokument vorhanden sein. Sie werden mit dem Dokument abgespeichert, bleiben also auch zwischen Editier-Sitzungen erhalten. Ein Benutzer kann zu einem Zeitpunkt mehrere, voneinander

unabhängige, Reservationen in einem Dokument besitzen. Abbildung 6 zeigt vor dem Hintergrund des MultimETH-Hauptfensters ein Beispiel eines MultimETH-Dokuments mit zwei aktiven Reservationen zweier Benutzer.<sup>2</sup>

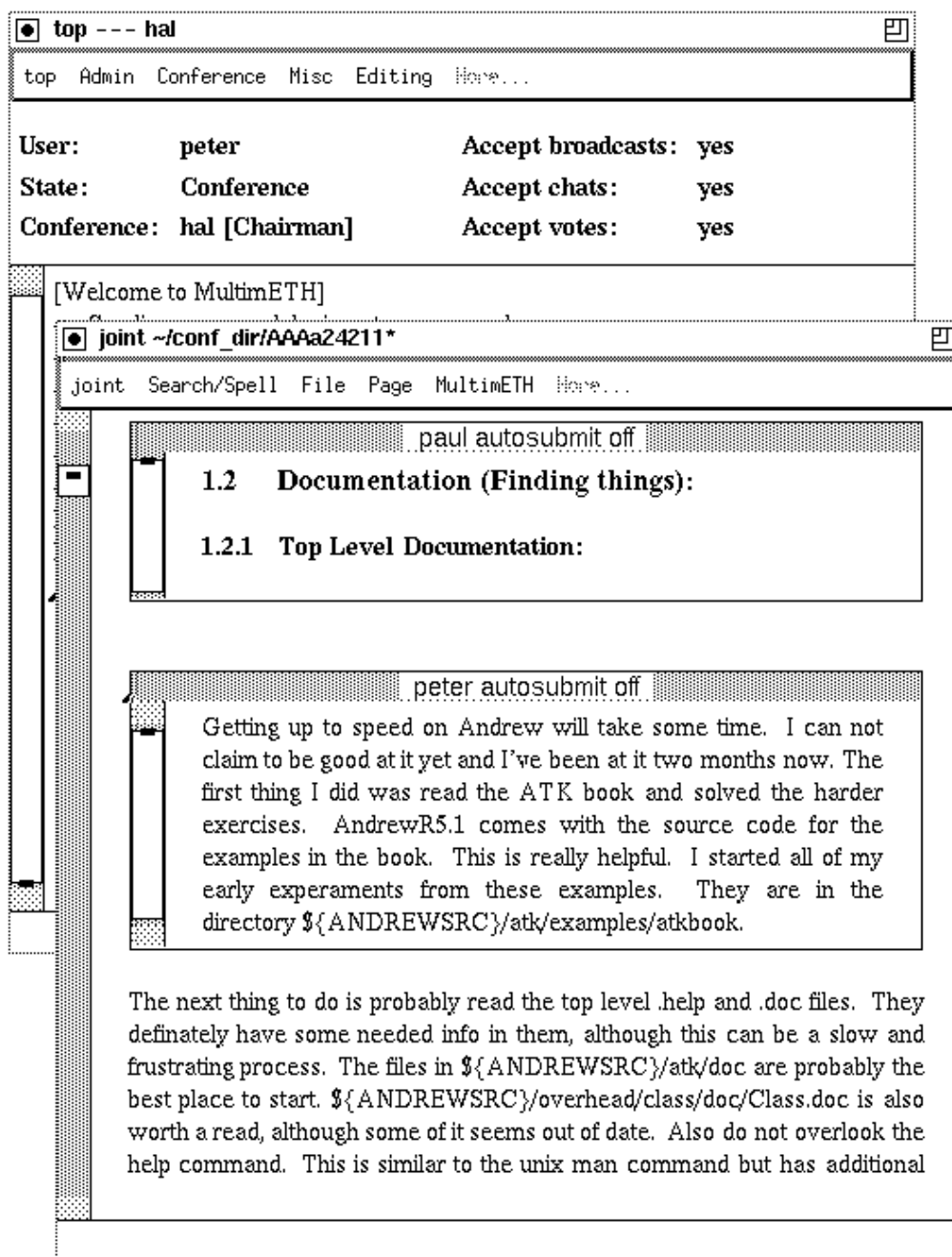


Abbildung 6: Beispiel von Reservierungen in einem MultimETH-Dokument

<sup>2</sup> Das Reservierungsfenster erscheint hier zu klein für eine effektives Editieren. Das Fenster kann jedoch einerseits vergrößert werden, andererseits ist es möglich, zu einer Reservierung ein separates, größeres Fenster mit identischem Inhalt anzuzeigen, in welchem dann wesentlich bequemer editiert werden kann.

Um einen Mittelweg zwischen dem Bedarf nach Anzeige der jeweils aktuellen Dokumentenversion und der Netzbelastung zu realisieren, wurde das Versenden von Änderungsoperationen innerhalb einer Reservation benutzergesteuert realisiert. Hierbei kann durch eine “submit”-Operation die Verteilung der aktuellen Version eines reservierten Bereiches an alle Editor-Klienten bewirkt werden. Die Operation kann dabei sowohl vom Besitzer der reservierten Region als auch von allen anderen Benutzern ausgeführt werden. Eine Erweiterung dieses Mechanismus zum automatischen Versenden der jeweils aktuellen Version (“autosubmit”) wurde ebenfalls implementiert (siehe [Bauer 93]). Die beiden anderen Operationen auf Reservationen sind “commit” und “unreserve”, welche beide die Reservation wieder aus dem Dokument entfernen, und eine Aktualisierung bei allen Klienten erzwingen. Während bei der “commit”-Operation ein Änderungsgrund abgefragt und die Änderung in das Dokument übernommen wird, setzt die “unreserve”-Operation die reservierte Region wieder auf ihren Originalzustand (vor der Reservation) zurück. Hierzu wurde das “jnote”-Objekt so implementiert, dass es jeweils drei Unterobjekte enthält, den Original-Inhalt, den aktuellen Inhalt und die letzte mittels “submit” übergebene Modifikation des Inhalts.

Weiterführende Arbeiten am MultimETH-Editor verfeinerten das Konzept der Zugriffsregelung und der Anzeige von Dokument-Attributen mittels eines entsprechenden Objekts in jedem MultimETH-Dokument (“Access Object”, siehe [Bauer 93]). Abbildung 7 zeigt diesen Teil eines MultimETH-Dokumentes im MultimETH-Editor.

Ein Access Object enthält neben dem Namen des Besitzers des Dokuments auch den Namen der Konferenz, dem das Dokument zugeordnet ist, sowie das Erstellungsdatum des Dokuments. Zudem enthält das Objekt eine Liste aller bei “commit”-Operationen angegebenen Änderungsbegründungen mit dem zugehörigen Benutzernamen und Änderungsdatum (“Change Log”).

Schliesslich enthält das Access Object Angaben über die Zugriffsrechte auf das Dokument, geordnet nach dem Besitzer des Dokuments (Owner), den Angehörigen einer Konferenz, der das Dokument zugeordnet ist (Conference), und allen anderen auf dem Server-System eingetragenen Benutzern (World). Die Zugriffsrechte, die diesen drei Gruppen zugeteilt werden können, sind Leserecht, Schreibrecht und Annotationsrecht. Das Leserecht erlaubt, ein Dokument ohne Änderungen zu editieren und zu kopieren, das Schreibrecht, ein Dokument zu editieren, die Attribute des Dokuments zu verändern und das Dokument zu löschen. Das Schreibrecht alleine erlaubt das Überschreiben und Löschen eines Dokumentes, es umfasst jedoch nicht implizit auch das Leserecht, das Dokument kann also in diesem Fall nicht editiert werden. Besitzen die Konferenzteilnehmer ein Recht nicht, welches aber für alle anderen auf dem Server-System eingetragenen Benutzer vorhanden ist (World), so gilt dieses Recht auch für die Konferenzangehörigen und den Besitzer des Dokumentes. Das Annotationsrecht ist für eine spätere Erweiterung vorgesehen, bei der der Dokumenteninhalte nicht verändert, jedoch schriftlich kommentiert werden darf. Dieses Zugriffsrecht wird momentan nicht ausgewertet. Abbildung 7

zeigt vor dem Hintergrund des MultimETH-Hauptfensters die Zugriffsrechte als Teil des Access Object für ein Beispieldokument.

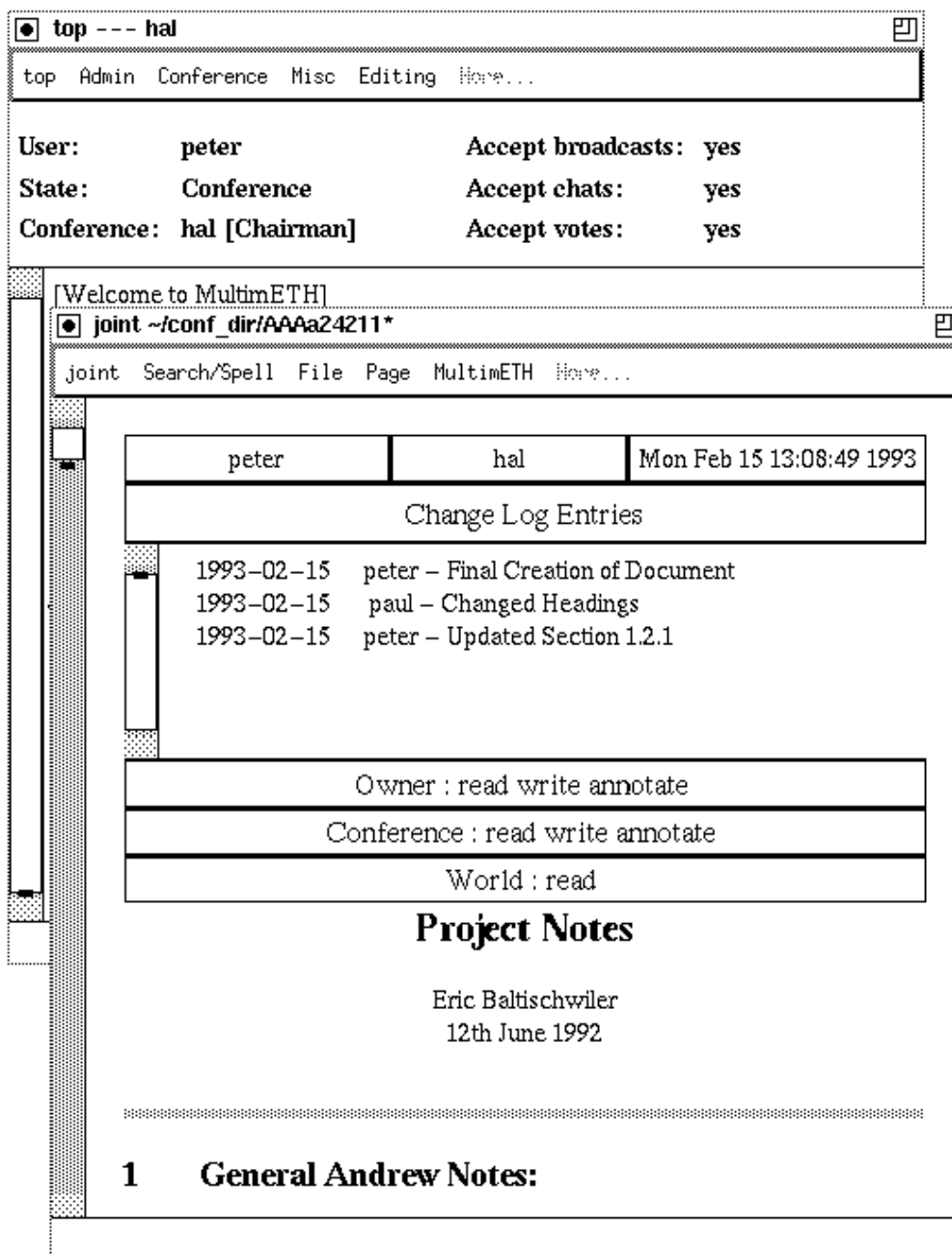


Abbildung 7: Anzeige der Attribute und des Änderungsprotokolls zu einem Dokument

Das Access Object dient jedoch nur zur Anzeige der Dokument-Attribute und wird durch Operationen z.B. im Browser (Setzen der Zugriffsrechte in einem speziellen Dialogfeld, Ändern des Besitzers, der Konferenz oder des Erstellungsdatums durch Kopieren) oder im Editor (Setzen des Erstellungsdatums durch erstmaliges Abspeichern, Erweitern des Änderungsprotokolls (Change Log) durch "commit"-Operation) implizit oder explizit verändert.

Für den MultimETH-Editor wurden im Projektzeitraum bereits mehrere Ergänzungen und Erweiterungen definiert und teilweise bereits in der Software vorbereitet. Diese Ergänzungen sind:

- 1) Annotationen, die dem Dokument, Objekten im Dokument oder auch an beliebiger Stelle im Dokument eingefügt werden können. Über diesem Mechanismus wäre es möglich, Teilnehmern ohne Schreibrecht auf den eigentlichen Dokumenteninhalte das schriftliche Kommentieren und Ergänzen des Dokumentes zu erlauben. Ein entsprechendes Zugriffsrecht wurde bereits definiert, jedoch würde die Bereitstellung der entsprechenden Funktionalität im Editor einen nicht unerheblichen Arbeitsaufwand bedeuten.
- 2) Darstellung der gerade aktuellen Version des bearbeiteten Dokumentes durch den Editor-Server in einem X-Fenster. Der Editor-Server (als X-Windows-Klient) versucht immer, ein X-Fenster anzulegen. Ist auf dem Server-System keine entsprechende DISPLAY-Variable definiert und gültig, kann der Editor-Server nicht gestartet werden. Hier sollte die Möglichkeit angeboten werden, dieses X-Fenster bei einem anderen X-Server (z.B. beim Sitzungsleiter) anzuzeigen, oder die Darstellung dieses Fensters zu unterdrücken. Es konnte hierfür jedoch im Projektzeitraum keine befriedigende Lösung gefunden werden.
- 3) Behandlung von "multiplen" Reservationen. Hier wäre das automatische Zusammenfügen von direkt aneinanderliegenden Reservationen des selben Benutzers u.U. sinnvoll, wird jedoch momentan nicht unterstützt. Zudem wäre es sinnvoll, dass die Reservation einer Region, in der bereits eine Reservation eines anderen Benutzers liegt, nicht zur Ablehnung des Reservationswunsches, sondern zur Reservation von zwei Teilregionen vor und hinter der anderen Reservation führt. Im bestehenden Prototyp wird jedoch ein Teil dieser Funktionalität dadurch zur Verfügung gestellt, dass eine Reservation des ersten Teils bis zur Fremd-Reservation ausgeführt wird.

## **2.5. Netzwerk-Administration und Sicherheitsaspekte**

Die Kommunikationsmöglichkeiten des MultimETH Konferenzsystems werden grundsätzlich durch die Kommunikationsmöglichkeiten des verwendeten OSI-Protokollstacks bestimmt. Diese beschränken sich deshalb bei MultimETH auf die Kommunikationsmöglichkeiten von ISODE, dem verwendeten OSI-Protokollstack, welcher die Netzwerk-Protokolle TCP/IP und X.25 unterstützt. Da aber die Verfügbarkeit, Konnektivität und Leistungsfähigkeit dieser beiden Protokolle ausserhalb der universitären Umgebung und zum Teil auch inner- und ausserhalb von Firmen nicht sehr gross ist, wurde für MultimETH die Verwendung weiterer Netzwerkinfrastrukturen vorgesehen. Bei der Evaluation ergab sich, dass das digitale Telefonnetz der PTT, welches auf dem ISDN Standard basiert, die gestellten Anforderungen, nämlich eine einfache Installation, eine grosse örtliche Verfügbarkeit, ausreichende Leistung und eine einfache Benutzung, erfüllt. Das digitale Telefonnetz der PTT wird in der Schweiz als SWISSNET bezeichnet. Zu Beginn des Projektes stand die Version SWISSNET 1+ zur Verfügung, mittlerweile wurde der Betrieb von SWISSNET 2 aufgenommen.

Die zusätzliche Verwendung von ISDN stellte das Projekt jedoch vor neue Aufgaben. Zum einen existierten bei Projektbeginn weder seitens der Industrie noch seitens der PTT Vorstellungen darüber, wie und mit welchen Hilfsmitteln ISDN (SWISSNET) verwendet werden soll. Zum anderen existierten kaum Hardwareausrüstungen und Software-Werkzeuge, mit welchen die Datenübertragung über SWISSNET erfolgen konnte. Daher wurde die Entscheidung getroffen, die Übertragung mittels "Remote Bridges" und Terminaladaptern gemäss Abbildung 8 zu realisieren, da beide Komponenten beim Projektstart verfügbar waren und – eine Erstellung der Datenverbindung von Hand vorausgesetzt – keine zusätzliche Software-Installation in den beteiligten Arbeitsplatzrechnern und Serversystemen nötig war.

Die gewählte Form der Datenübertragung basiert auf dem IEEE Standard 802.3, welcher die Übertragung von Daten über Ethernet beschreibt. Die Daten werden von der Remote Bridge auf der MAC (Media Access Control) - Schicht (OSI-Schicht 2) übertragen. So werden nicht einzelne Rechner, sondern LAN-Segmente miteinander verbunden, eine Verbindung, welche für die Protokolle oberhalb der MAC-Schicht völlig transparent ist. Deshalb ist für diese Übertragungsart eine initiale Konfiguration notwendig, welche die eindeutige Vergabe der Adressen vornimmt, die von Protokollen oberhalb der MAC-Schicht verwendet werden. Im vorliegenden Fall sind dies die IP-Adressen der Systeme auf den beiden LAN-Segmenten. Weiter müssen in dieser initialen Konfiguration die Bridges konfiguriert werden. Dabei geht es im wesentlichen darum, ihnen eine IP-Adresse (für SNMP-Gebrauch) zu geben, die seriellen V.35 Schnittstellen zu konfigurieren und den Bridging-Mode (static/learning) sowie allfällige Tabellen zu initialisieren. Im weitem müssen die SWISSNET-Nummern gegenseitig bekannt gemacht und die Terminaladapter auf den selben Übertragungsmodus eingestellt werden.

Um einem Teilnehmer aus einem entfernten LAN-Segment die Teilnahme an einer MultimETH-Konferenz zu ermöglichen, muss zur Laufzeit diese Verbindung aufgebaut werden. Dazu wird vom lokalen Terminaladapter der entfernte Adapter angerufen. Nimmt dieser den Anruf entgegen, ist die Verbindung erfolgreich erstellt.

Diese zusätzliche Ausrüstung muss während des Betriebs überwacht werden können, Fehler sollten erkannt, gemeldet und – wenn möglich – behoben werden können. Zudem sollte es möglich sein, die Geräte, soweit diese es zulassen, entfernt zu konfigurieren. Die verwendeten V.35-Terminaladapter können jedoch nur mit einem Terminal am "control port" konfiguriert werden. Sie müssen im allgemeinen aber nach der oben erwähnten initialen Konfiguration im Betrieb nicht mehr umkonfiguriert werden. Die Bridges können ebenfalls nur mit einem Terminal am "terminal port" konfiguriert werden. Nach der initialen Konfiguration sind zur Laufzeit mögliche Operationen zum Beispiel das Erweitern der IP-Alias-Tabelle, das Ein- oder Ausschalten eines physikalischen Ports der Bridge (Sperrung des Zugangs), das Ein- und Ausschalten der Multilink-Option der Bridge (zwei parallele Verbindungen zum selben Endpunkt) sowie das Löschen eines Eintrags in der ARP-Tabelle der Bridge (Address Resolution Protocol, wird für die Bridging-Funktionalität benötigt).



Der in Abbildung 8 gezeigte Aufbau wirft jedoch zusätzlich Sicherheitsfragen auf. Bei der gewählten Einrichtung hat grundsätzlich jeder beliebige Benutzer die Möglichkeit, sich die gleiche Ausrüstung zu beschaffen und sich an das andere LAN-Segmente anzukoppeln. Damit ist ein nicht zu vernachlässigendes Gefahrenpotential verbunden, können so zum Beispiel doch auf einfache Weise "denial of service"-Angriffe oder Identitätstäuschungen vorgenommen werden. Im MultimETH-System wurde deshalb zusätzlich ein "sicherer" LAN-Zugang entwickelt.

Ein beliebiger Terminaladapter ist als "incoming-TA" ausgezeichnet. Nur die ISDN-Nummer dieses Anschlusses wird bekanntgemacht und es werden nur auf diesem Terminaladapter Anrufe entgegengenommen. Die anderen Terminaladapter, welche für die eigentliche LAN-Kopplung verwendet werden, sind so konfiguriert, dass sie kein Anrufe entgegennehmen. Dies stellt sicher, dass Verbindungen nicht unkontrolliert erstellt werden können.

Bei einem ankommenden Anruf auf dem "incoming-TA" wird die Funktionalität von ISDN ausgenutzt, dass die rufende Nummer angezeigt wird. Diese wird vom *MNMPD (MultimETH Network Management Port Daemon)*, welcher die Steuerleitungen verwaltet, gelesen und der Anruf wird danach abgewiesen. In der MultimETH-Benutzer-Datenbank wird nach einem Benutzer gesucht, welcher mit der soeben empfangenen ISDN-Nummer registriert ist. Ist ein solcher vorhanden, so wird dieser Teilnehmer auf der gefundenen Nummer von einem aus einer Gruppe von Terminaladaptern zurückgerufen, welche an die Bridge angeschlossen sind. Damit ist die Verbindung der LAN-Segmente zustande gekommen. Ist kein Benutzer mit der empfangenen ISDN-Nummer registriert, so wird nicht zurückgerufen. Damit ist ein gesicherter, auf der Eindeutigkeit und der Fälschungssicherheit der ISDN-Nummer basierender, Netzzugang gewährleistet.

Die Realisierung der oben beschriebenen Funktionalität im Rahmen von MultimETH wurde in einer ersten Arbeit [Paul/Uhlm 91] in Angriff genommen. In dieser Arbeit wurde eine erste Spezifikation des Management-Tools vorgenommen, welche in den Folgearbeiten [Ruess 92] und [Hächler 93] Verwendung fand. In einem zweiten Schritt wurde in der Arbeit [Ruess 92] die Ansteuerung der Bridges und der Terminaladapter realisiert. Dabei stand die Implementation des sicheren LAN-Zugangs sowie der damit verbundenen Dialback-Funktionalität im Vordergrund. So entstand in dieser Arbeit eine erste Version des *MNMPD*.

Der *MNMPD* verwaltet alle "control ports" der Bridge und der Terminaladapter. Durch den eingesetzten V.24 Switch, welcher mittels eines CAMTEC MiniPAD realisiert wurde, wurde erreicht, dass mehrere Bridges und mehrere Terminaladapter installiert werden können. Somit ist diese Konfiguration beliebig skalierbar. Gleichzeitig wurde die Topologie jedoch zusätzlich komplexer, was die Realisierung erheblich erschwerte. In der nachfolgenden Arbeit [Hächler 93], in welcher des MultimETH Netzwerk Management in der vorliegenden Form entwickelt wurde, wurde der *MNMPD* erweitert, sowie der *MNMS (MultimETH Network Management*

Server), der *MNMC* (*MultimETH Network Management Client*) und der *MNMSTD* (*MultimETH Network Management SNMP Trap Daemon*), entwickelt.

Der *MNMS* ist für die Bereitstellung der oben definierten Funktionalität verantwortlich. Dabei bedient er sich einerseits des *MNMPD* zum Zugriff auf die Bridge und die Terminaladapter, andererseits erhält er SNMP-Traps vom *SNMPTD*. Der *SNMPTD* wurde so realisiert, dass der *snmpd* der verwendeten SNMP-Implementation so abgeändert wurde, dass dieser die empfangenen Traps nicht dem UNIX *syslogd* weiterreicht, sondern in eine Message Queue schreibt, welche dann vom *MNMS* gelesen werden kann.

Die Benutzerschnittstelle (*MNMC*) wurde analog zu den anderen Klientenprozessen von MultimETH mit dem Andrew-Toolkit erstellt, und hat somit das selbe "look and feel" wie die anderen Teile des MultimETH Konferenzsystems. Die Benutzerschnittstelle ist zustandslos und kommuniziert mit dem Server über die ISODE Verbindung des MultimETH-Konferenzsystems. Die Operationen, welche der Benutzer ausführt, werden über diese Verbindung dem Management-Server gesendet, welcher die nötigen Aktionen durchführt und ein allfälliges Resultat oder eine Fehlermeldung auf dem selben Weg an die Benutzerschnittstelle zurücksendet. Das Hauptfenster der Benutzerschnittstelle mit den Haupt-Menues und dem ausgeklappten Information Menu ist in Abbildung 9 dargestellt.

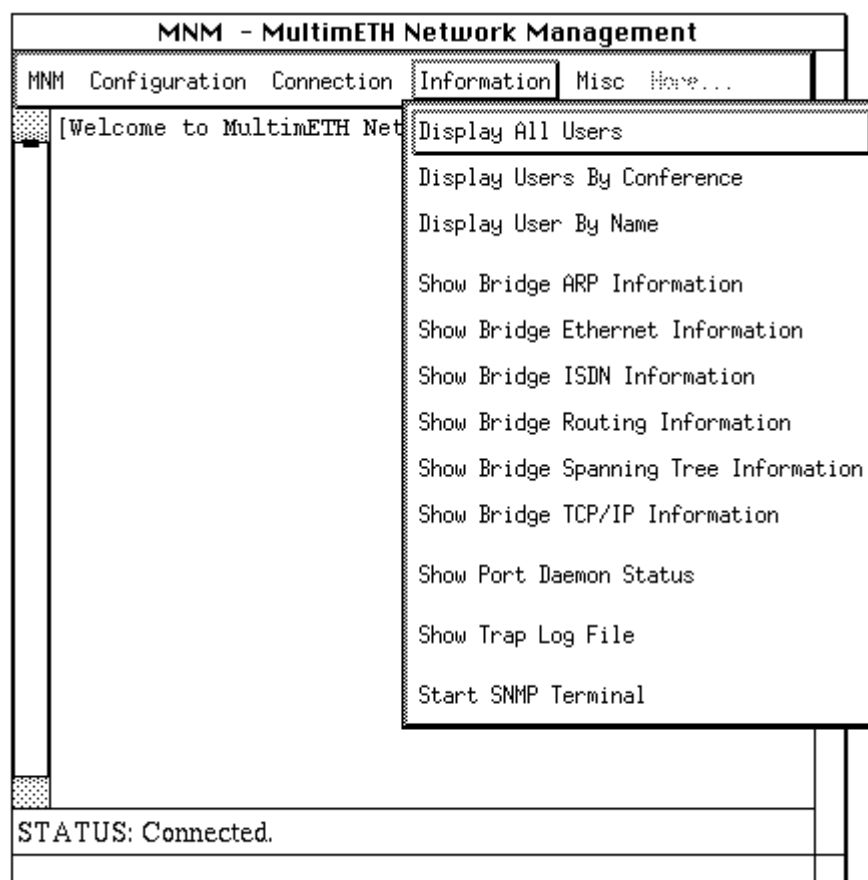


Abbildung 9: Das Hauptfenster des MultimETH Network Management -Tools mit ausgeklappten Information-Menu

Das so bereitgestellte *MNM (MultimETH Network Management)* stellt dem Konferenzleiter (Chairperson) einer MultimETH Konferenz ein komfortables Software-Werkzeug zur Verfügung, um die eingesetzten TAs und Bridges zu überwachen und zu steuern. Das MNM ist somit eine sinnvolle Ergänzung zum bestehenden MultimETH Konferenzsystem und vereinfacht die Kontrolle der Kommunikation mit entfernten Konferenzteilnehmern erheblich.

Da diese LAN-Kopplung auch von anderen Anwendern (z. B. von Macintosh-Benutzern, welche z.B. Appleshare über Ethernet betreiben) gebraucht werden kann, stellt dieses Management-Tool eine exemplarische Lösung für eine solche Managementaufgabe dar. Diese könnte auch ausserhalb von MultimETH auf eine andere Plattform portiert und dort verwendet werden.

## **2.6. Sprachkonferenz-Einrichtung**

Während für Applikationen wie Filetransfer oder elektronische Post eine reine Datenverbindung ausreichend ist, ist für eine Applikation aus dem Bereich "joint editing", zu welchem das MultimETH-System gehört, ein zusätzliches Übertragungsmedium mit unmittelbarem Zugriff und einfacher Benutzung nötig. Hierfür stehen neben der Datenübertragung grundsätzlich die Medientypen Audio und Video zur Verfügung.

Für das MultimETH-System wurde das Medium "Audio" gewählt, da es als Ergänzung für das verteilte Editieren gut geeignet und sehr einfach zu bedienen ist. Insbesondere ist das mit der sprachlichen Interaktion zwischen Menschen verbundene "soziale Protokoll" weitgehend intuitiv bekannt und dementsprechend einfach und unmittelbar einsetzbar. Mit dieser Erweiterung um eine Sprachkonferenz-Komponente wird MultimETH der gestellten Anforderung, eine möglichst realitätsnahe Umgebung für die Unterstützung elektronischer Arbeitsgruppentreffen zur Verfügung zu stellen, weitgehend gerecht.

Bei Projektbeginn musste festgelegt werden, wie die Sprache in das bestehende Konferenzsystem eingebettet werden soll. Hierfür standen grundsätzlich zwei verschiedene Möglichkeiten offen, nämlich die Einbettung und Übertragung der Sprache über die bestehende Datenverbindung, oder die Trennung von Sprach- und Datenübertragung. Für die erste Variante wäre die Integration der Sprache einerseits in den OSI-Protokollstack und andererseits die Audio-Ein- und Ausgabe bei den Teilnehmern an ihren Arbeitsplatzrechnern nötig. Diese Realisierung wäre jedoch tief in die Problematik der Protokolle für integrierte Netze und Schnittstellen und Anschlüsse an solche Netze vorgestossen. Keine dieser Problematiken lag aber im Zielbereich von MultimETH, daher wurde dieser Ansatz nicht weiter verfolgt.

Stattdessen wurde die zweite Lösungsmöglichkeit, d.h. die Trennung der Sprach- und Datenübertragung, im Prototyp realisiert. Obwohl die Benutzerschnittstelle der Konferenzapplikation beide Möglichkeiten für den Benutzer vereint, wird für die Sprachkonferenz auf die bestehende Infrastruktur der analogen und digitalen Telefonie zurückgegriffen. So ist für die Teilnahme an

der Sprachkonferenz keine zusätzliche Hardware nötig und der OSI-Protokollstack musste ebenfalls nicht abgeändert werden.

Für die Realisierung der Sprachkonferenz parallel zur Datenübertragung wurde eine sternförmige Topologie gewählt. Im Zentrum bedient eine Sprachkonferenzeinheit (SE) die Teilnehmer an einer MultimETH-Konferenz. Hierfür muss für jeden aktiven Teilnehmer in dieser SE ein Telefon-Interface zur Verfügung stehen. Die Abbildung 10 zeigt das Blockschaltbild der SE. Es ist ersichtlich, dass sowohl Teilnehmer aus dem analogen als auch aus dem digitalen Telefonnetz der PTT an einer Konferenzschaltung teilnehmen können.

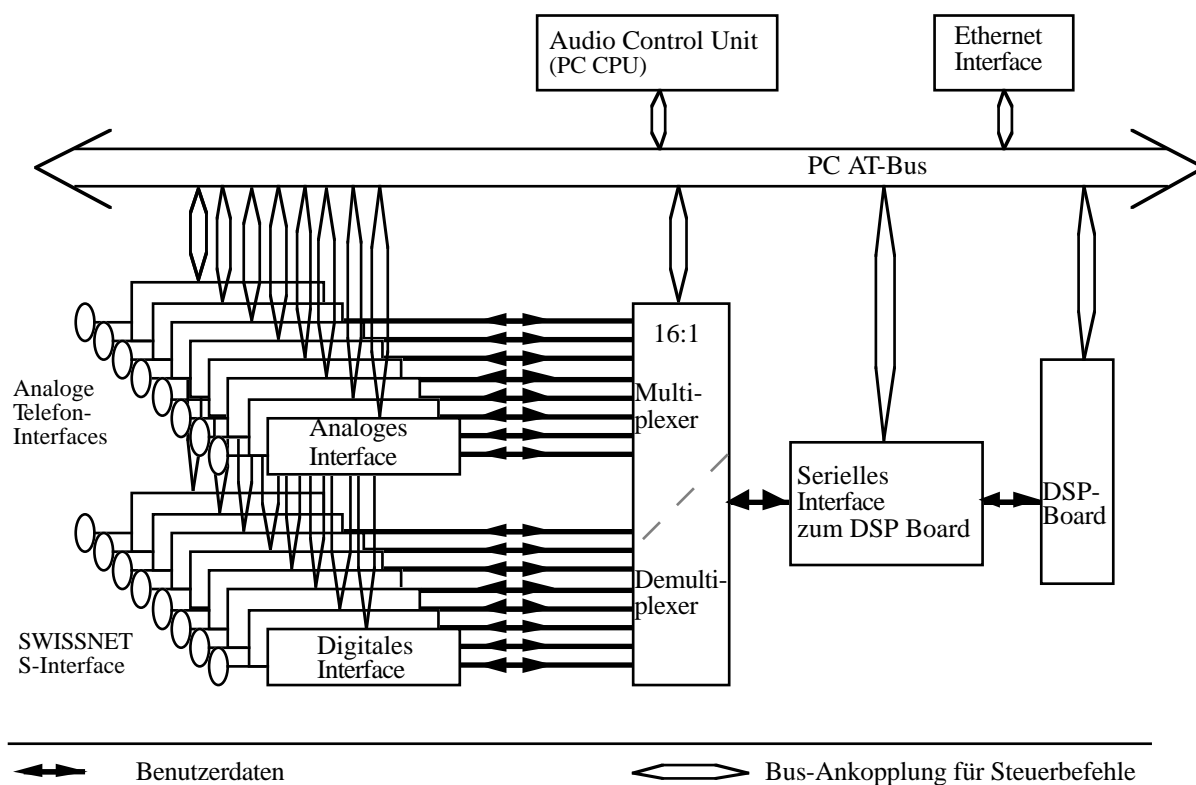


Abbildung 10: Das Blockschaltbild der Sprachkonferenzeinheit

Die Funktionsweise der SE kann wie folgt zusammengefasst werden. Die hereinkommenden Sprachdaten werden entweder von den analogen Interfaces digitalisiert oder von den SWISSNET-Interfaces bereits digital empfangen. Diese digitalen Sprachdaten werden einem Multiplexer zugeführt, welcher die Sprachdaten im Zeitmultiplex-Verfahren auf eine schnelle serielle Leitung multiplext. Über das serielle Interface wird dieser Datenstrom dem digitalen Signalprozessor (DSP) zugeführt. Dieser mischt die einkommenden Sprachdaten gemäss einer Matrix, welche vom Konferenzleiter angegeben werden kann. Diese Matrix gibt an, welcher Teilnehmer die Sprachdaten welches anderen Teilnehmers empfangen kann, und wie die einzelnen Datenkanäle verstärkt werden sollen. Dies ergibt die eigentliche Konferenzschaltung. Die so berechneten Ausgangsdatenströme für die einzelnen Teilnehmer werden vom DSP über das serielle Interface dem Demultiplexer zurückgesendet, welcher diesen Datenstrom entspre-

chend aufteilt. Über die entsprechenden Telefon-Interfaces gelangen die Sprachdaten zu den Endausrüstungen der Teilnehmer zurück. In diesem Ablauf ist nur die SE eine neue, spezialisierte Hardware. Damit konnte mit wenig zusätzlicher Hardware eine komfortable Sprachkonferenz realisiert werden.

Damit diese Sprach-Konferenzeinrichtung auch innerhalb von MultimETH gebraucht werden kann, ist eine an das "look and feel" von MultimETH gut angepasste Benutzerschnittstelle erforderlich. Diese hat zwei verschiedene Ausprägungen, zum einen muss sie dem Konferenzleiter alle Konfigurations- und Steuerungsmöglichkeiten bieten, zum anderen müssen die einzelnen Teilnehmer ihre Einstellungen für ihren jeweiligen Sprach-Ausgang beeinflussen können. Dazu ist ähnlich wie in den anderen Teilen von MultimETH verfahren worden. Abbildung 11 zeigt die erforderliche Erweiterung der Prozess-Struktur:

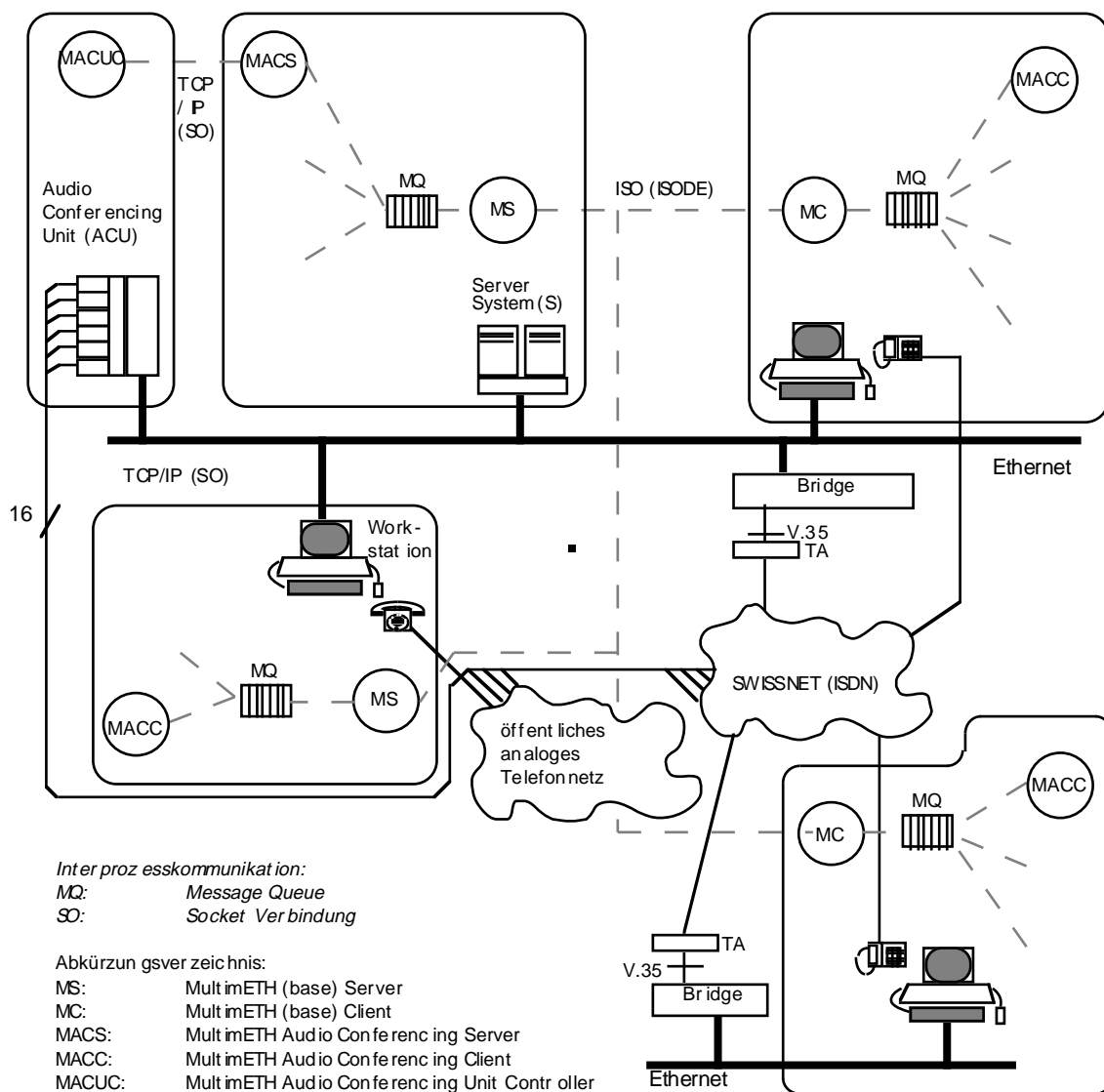


Abbildung 11: Die Prozess-Struktur der Sprachkonferenzeinheit für MultimETH

Die Hauptfunktionalität realisiert der *MultimETH Audio Conferencing Server (MACS)*. Er stellt dem Benutzer die gesamte Konferenz-, Konfigurations- und Überwachungsfunktionalität zur Verfügung. Dazu kommuniziert er mit den verschiedenen Klienten über die ISODE-Verbindung, welche das MultimETH-Konferenzsystem zur Verfügung stellt. Die Meldungen, welche zu den Klienten gesendet werden, werden über die selbe Message Queue dem MultimETH-Server übergeben, welche auch vom MultimETH Network Management und dem Editor verwendet wird. Der *MACS* ermöglicht die Audio-Konferenzschaltung und weiss über den Zustand der *Audio Conferencing Unit (ACU)* jederzeit Bescheid. Er erhält von den Benutzerschnittstellen die Aufträge, welche er bearbeitet. Im Server wird für die Audio-Konferenz eine Verbindungsmatrix aufgebaut und aktuell gehalten. Darin wird die aktuelle Konfiguration der Sprachkonferenz wiedergegeben. Die Verbindungsmatrix arbeitet mit logischen Kanälen, demzufolge wird hier auch eine Abbildung von logischen Kanälen auf physikalische Interfaces der Sprachkonferenzeinheit vorgenommen. Zusätzlich kann jeder Benutzer "seine" Lautstärke selber verändern. Diese Information wird ebenfalls im Server gespeichert, zusammen mit der Verbindungsmatrix verarbeitet und der Sprachkonferenzeinheit gesendet. Diese mischt die Sprachdaten dementsprechend. Dieser *MACS* wird vom MultimETH-Server aufgestartet.

Aus Anwendersicht gibt es zwei verschiedene Benutzerschnittstellen. Zum einen existiert die Benutzerschnittstelle, welche nur dem Konferenzleiter zur Verfügung gestellt wird. Nur diese Schnittstelle erlaubt, Funktionen des Sprachkonferenz-Managements aufzurufen, die Sprachverbindungen zu steuern sowie die SE zu konfigurieren. Funktionen des Sprachkonferenz-Managements sind zum Beispiel das Öffnen einer Konferenz, die Wahl der Lautstärke, der Tonqualität und der Stummschaltung für jeden Eingang, der Empfang von (Fehler)-Meldungen sowie die Bildung von Teilkonferenzen und deren Wiedervereinigung.

Die Benutzerschnittstelle für alle anderen Benutzer (es gibt jeweils nur ein Konferenzleiter pro Konferenz) erlaubt es, die Lautstärke des eigenen Ausgangs zu variieren, wichtige Meldungen und Konferenzinformationen zu empfangen sowie lokale Optionen einzustellen.

Der Server sowie die Benutzerschnittstellen sind aus Konsistenzüberlegungen so implementiert, dass sie Anfragen sequentiell bearbeiten. Die Implementation des Protokolls zwischen Klienten und Server konnte ohne grosse Probleme implementiert werden. Beim Server hingegen bedarf es einiger zusätzlicher Überlegungen. Zum einen muss die Situation behandelt werden, bei welcher Klient und Server "gleichzeitig" eine Anforderung absenden. Diese Situation wird so gelöst, dass die Anfrage des Servers eine höhere Priorität hat und somit die Anfrage des Klienten verworfen wird. Nach erfolgreicher Bearbeitung der Anfrage des Servers muss der Klient seine Anfrage nochmals senden. Da der Server zusätzlich die Hardware der Sprachkonferenzeinheit steuern und überwachen muss, entsteht eine weitere kritische Situation, wenn zwei Anfragen, nämlich eine von einem Klienten und – bevor die Anfrage des Klienten beantwortet ist – eine vom Steuerprozess der Sprachkonferenzeinheit, dem *MACUC (MultimETH Audio Con-*

*ferencing Unit Controller*) eintrifft. Solche, sich touchierenden Anfragen, werden nach dem FIFO-Prinzip behandelt.

Für die Kommunikation zwischen dem *MACS* und dem *MACUC* wurde ein Protokoll implementiert, welches mit Bestätigungen und Laufnummern arbeitet. Die Laufnummer ist Bestandteil der Meldung, die Bestätigung ist ein normales Paket, welches ebenfalls Nutzdaten enthalten kann. Am Anfang jedes Paketes wird die Identifikation der auszuführenden Prozedur angegeben. Das Protokoll hat auf beiden Seiten zwei Zustände, wobei bei gleichzeitiger Anforderung von beiden Seiten die Anforderung der SE Vorrang hat. Dies wurde so gewählt, weil von der SE die zeitkritischeren Anfragen gesendet werden, so zum Beispiel ein Meldungs-Anfragepaket, welches meldet, dass ein Interface gerade angerufen wird, und fragt, ob der Anruf entgegengenommen werden soll.

Die so bereitgestellte Sprachkonferenzeinheit ist sehr einfach zu verwenden und ergänzt somit den Vorgang des verteilten Editierens hervorragend. Die Nutzen einer Ergänzung einer Konferenzapplikation mit dem Medium Audio ist bereits in der einschlägigen Literatur ausreichend beschrieben worden. Insbesondere wurde gezeigt, dass die Produktivität eines elektronischen Treffens, welches Audio-Unterstützung bereitstellt, gegenüber rein datenbasierten Interaktionen beträchtlich gesteigert werden kann.

### 3. Allgemeine Anmerkungen und Gesamtbeurteilung

Während der Implementierung des MultimETH-Prototyps wurden einige allgemeine Erfahrungen bezüglich der Entwicklung komplexer verteilter Software-Systeme gesammelt, die im folgenden kurz zusammengefasst werden:

- Der ursprüngliche Prototyp wurde auf Sun-3-Systemen unter SunOS-4.0.3 entwickelt. Die Portierung auf die seit Anfang 1992 im Projekt vorhandenen Sun-SPARC-Systeme (Sun-4c und Sun-4m mit SunOS-4.1.3) erwies sich als überraschend schwierig, insbesondere da mehrere komplexe Software-Pakete (ISODE, Andrew) involviert waren, die teilweise eine eigene, vom Betriebssystem abweichende, Realisierung von “dynamic linking” von Objekten realisierten. Zusammen mit den ab SunOS-4.1.3 verwendeten “shared libraries” von SunOS ergab dies Probleme, die nur durch den Neu-Design von Teilen von MultimETH ausgeräumt werden konnten.
- Die Verwendung eines “besseren” C-Compilers (GNU “gcc” statt des mit SunOS ausgelieferten C-Compilers “cc”) versprach zunächst eine Verbesserung der Programmierumgebung. Aufgrund von Problemen bei der Integration der vorhandenen Software-Pakete in MultimETH mit dem “gcc”-Compiler wurde das Projekt jedoch schliesslich vollständig mit dem mit SunOS ausgelieferten C-Compiler “cc” implementiert. Eine “Mischung” von Objekten, die durch verschiedene C-Compiler übersetzt wurden, ist aber in jedem Fall problematisch und zu vermeiden.
- Es existieren keine funktional hinreichenden Werkzeuge zum Debugging komplexer verteilter Applikationen, insbesondere wenn das Gesamtsystem zur Laufzeit aus mehreren kommunizierenden Prozessen (teilweise auf verschiedenen Systemen) besteht.

Insgesamt konnte unserer Meinung nach aber im Rahmen des Projektzeitraums und der Projektressourcen eine gute Lösung erarbeitet werden. Der resultierende Prototyp ist sicher noch verbesserbar, kann jedoch durchaus für Demonstrationen und Pilotnutzung eingesetzt werden. Als “Nebeneffekt” konnte bei Mitarbeitern und Studierenden ein beachtlichen Know-how in den Bereichen verteilte Anwendungen und Kommunikationsdienste aufgebaut sowie die Rolle der Schweiz im Bereich computergestützter Gruppenarbeit wesentlich ausgebaut werden. Um dieses Know-how nicht verlorengelassen zu lassen, sollten die Erfahrungen aus dem MultimETH-Projekt sollten daher unserer Meinung nach unbedingt in Folgeprojekte einfließen. Die an MultimETH beteiligten Mitarbeiter sind im Rahmen ihrer Möglichkeiten durchaus bereit und interessiert, ihre Erfahrungen auch über diesen Bericht hinaus entsprechend weiterzugeben.

## 4. Literatur

### 4.1. Studentische Arbeiten an der ETH Zürich

- [Aebi 91] Rolf Aebi, Eine Benutzeroberfläche für MultimETH mit ET++: Realisierung der MultimETH-C-Basis-Schnittstelle, Semesterarbeit, Sommersemester 1991
- [Aebi 92] Rolf Aebi, Untersuchung zur Aufteilung des Kontrollflusses zwischen User Interface Management Systemen und Event-basierten Applikationen, Diplomarbeit, Wintersemester 1991/92
- [Almesberger 92] Werner Almesberger, Entwurf und Teilimplementierung eines verteilten, multimediafähigen Editors, Diplomarbeit, Sommersemester 92
- [Balten/Lenden 93] Christian Baltensperger, Roger Lendenmann, Digitales Multiplexerinterface für einen Digitalen Signalprozessor (DSP), Semesterarbeit, Wintersemester 1992/93
- [Bauer 93] Daniel Bauer, Erweiterung eines verteilten, multimedialen Editors, Diplomarbeit, Wintersemester 1992/93
- [Burg/Marb 92] Christian Burgherr, Peter Marbach, Mischung von digitalisierten Sprachdaten, Diplomarbeit, Wintersemester 1991/92
- [Caronni 93] Germano Caronni, Konzeption und Entwicklung einer Komponente zur Dokumentenverwaltung, Diplomarbeit, Wintersemester 1992/93
- [Cazemier 93] Stefan Cazemier, Untersuchung der Leistungsfähigkeit des IBM-AT-Bus für Echtzeit-Sprachanwendungen über ISDN, Semesterarbeit, Wintersemester 1992/93
- [Cotting 92] Stefan Cotting, Funktionalitätserweiterung für ein OSI-orientiertes Konferenzsystem, Semesterarbeit, Sommersemester 1992
- [Cotting 93] Stefan Cotting, Entwicklung eines OSI-orientierten Whiteboard auf UNIX/X-Windows, Diplomarbeit, Wintersemester 1992/93
- [Gerloff 88] Harald Gerloff, Vergleich und Bewertung von Dokumentenarchitektur-Modellen für Multimedia-Dokumente, Diplomarbeit, Wintersemester 1987/88
- [Hächler 93] Stefan Hächler, Management-Tool für eine LAN-Erweiterung über ISDN, Diplomarbeit, Wintersemester 1992/93
- [Helbing 89] Rainer Helbing, Funktionsmuster für MultiMETH User Interface, Semesterarbeit, Wintersemester 1988/89
- [Imfeld 89] Karl Imfeld, Entwurf eines ROS-basierten Protokolls für ein Mehrbenutzer-Echtzeit-Konferenzsystem, Diplomarbeit, Sommersemester 1989
- [Künzler 93] Urs Künzler, Entwicklung einer graphischen Benutzerschnittstelle zur Steuerung einer Sprachkonferenzeinrichtung, Diplomarbeit, Sommersemester 1993

- [Lichtin 90] Hans-Peter Lichtin, Validierung des MultimETH-Konferenzprotokolls, Diplomarbeit, Sommersemester 1990
- [Paul/Uhlm 91] Marco Pauletti, Jürg Uhlmann: Management Tool für MultimETH, Semesterarbeit, Sommersemester 1991
- [Pflug 90] Jan Pflug, Validierung und Erweiterung des COMIT-Testtreiber-Systems, Semesterarbeit, Sommersemester 1990
- [Ruess 92] Niklaus Ruess, LAN-Kopplung über ISDN, Semesterarbeit, Sommersemester 1992
- [Ruess 93] Niklaus Ruess, Realisierung eines RTF-Andrew-Konverters Diplomarbeit, Wintersemester 1992/93
- [Wild 92] Markus Wild, Entwurf und Teilimplementierung eines verteilten, multimediafähigen Editors, Diplomarbeit, Sommersemester 1992
- [Zweiacker 91] Marc Zweiacker, Entwicklung eines OSI-basierten Konferenzsystems mit verteiltem Server, Diplomarbeit, Sommersemester 1991

#### **4.2. Andere relevante Arbeiten**

- [Lubich 89] H. Lubich: MultimETH: Ein Beitrag zur Konzeption eines Echtzeit-Multimedia-Konferenzsystems, ETH Zürich, Abteilung Informatik, Dissertationsarbeit Nr. 20, 1989
- [Stevens 90] W. Richard Stevens: UNIX Network Programming, Prentice Hall, 1990
- [West 87] C. West: Protocol Validation by Random State Exploration, in: Protocol Specification, Testing and Verification IV, B. Sarikaya, G. V. Bochman (eds.), p. 233-242, Elsevier, 1987