Aufbau und Konfiguration lokaler Multimedia Konferenz-Umgebungen

Frank Oertel
Mat.Nr. 94301
TU-Berlin

Berlin den

19. November 1995

Studienarbeit am

Fachgebiet für Offene Kommunikationssysteme (OKS)
des Fachbereichs Informatik (FB13)
der Technischen Universität Berlin

Betreut durch

Prof. Adam Wolisz (TU-Berlin)
Dr. Henning Schulzrinne (GMD-Fokus)

Inhaltsverzeichnis

1 Einleitung **1**

2 Problemstellung **3**

2.1 Einleitung 3

2.2 Konferenzentstehung 3

2.3 Problembeschreibung 4

2.4 Geforderte Dienstmerkmale 4

3 Die Systemarchitektur **7**

3.1 Einleitung 7

3.2 Überblick 7

3.3 Die Adressierung 8

3.4 Anruf 8

3.4.1 Der Conference Daemon 9

3.4.2 Der Location Server 9

3.4.3 Rufsignalisierung 9

3.5 Die Media Agents 10

3.6 Redirect 10

3.7 Forwarding 10

3.8 Adreß-Caching 11

3.9 Zyklenvermeidung bei Rufumleitung 11

3.10 Proxy-Mode 12

3.11 Lokale Konfiguration 12

4 Das MUCS-Protokoll **13**

4.1 Überblick 13

4.2 Das Anrufpaket 13

4.3 Das Antwortpaket 14

4.4 Auflösung von Gruppennamen 16

4.4.1 Returncodes 16

4.5 Erweiterbarkeit 17

5 Der Conference Controller **19**
5.1 Allgemeines 19
5.2 Das Hauptfenster 20
5.3 Starten einer neuen Konferenz 21
5.4 Die Rufsignalisierung 22
5.4.1 Signalisierungsdauer 23
5.4.2 Rufablehnung 23
5.4.3 Rufumleitung 24
5.4.4 Rufannahme 24
5.5 Verwaltung der Media-Agents 25
5.5.1 Definition der Media-Agent-Capabilities 25
5.5.2 Definition der Media-Agent Start-Parameter 27
5.6 Einstellungen 27
5.7 pmm 30
5.7.1 Kommunikation mit NEVOT 31

6 Ausblick/Bewertung **33**
6.1 Was fehlt? 33
6.2 Weitere Entwicklung 34

A Programmstruktur **A – 1**
A.1 Die `tcl`-Sourcefiles in `./lib` . . . . A – 1
A.2 Die Dokumenation in `./doc` . . . . A – 1

B Programm-Code **B – 3**
B.1 Das Hauptprogramm: `Coco.tcl` . . . . B – 3
B.2 Starten einer neuen Konferenz: `ConfSetup.tcl` . . . . B – 9
B.3 Einladen der Benutzer: `Dialer.tcl` . . . . B – 11
B.4 Starten des Help Processors: `Help.tcl` . . . . B – 16
B.5 Verwaltung der Medien-Agenten: `MediaMgr.tcl` . . . . B – 17
B.6 Anrufsignalisierung: `Notify.tcl` . . . . B – 24
B.7 Das Phone Directory: `Phonebook.tcl` . . . . B – 28
B.8 Die Programmkonfiguration: `Setup.tcl` . . . . B – 29
B.9 Speichern der Einstellungen: `Terminate.tcl` . . . . B – 32
B.10 Hilfsfunktionen: `Utils.tcl` . . . . B – 32

C mucsp Grammatik **C – 35**

Kapitel 1

Einleitung

Im Multimediabereich existieren gegenwärtig eine Reihe von Entwicklungen, deren wesentliche Aufgabe die effiziente und gesicherte Nutzdatenübertragung ist. Zum Beispiel gibt es Programme, mit denen Audio- oder Videodaten übertragen werden können. Wollen jedoch mittels dieser Programme zwei Personen miteinander kommunizieren, so muß mangels geeigneter Signalisierungsmöglichkeiten jeder darüber vorab Kenntnis besitzen. Die Absprache über eine aufzubauende Verbindung sowie das Aushandeln der zu verwendenden Programme (Media Agents), Protokolle und Parameter erfolgt über separate Medien, wie dem klassischen Telefon.

Diese Vorgehensweise ist offensichtlich umständlich und ineffizient, sollen beispielsweise zehn Personen in eine Konferenz eingeladen werden, so würde dies einen erheblichen Aufwand erfordern alle Teilnehmer vorab telefonisch darüber in Kenntnis zu setzen. Die Vorteile, die der Einsatz vernetzter Rechner gegenüber klassischen Kommunikationseinrichtungen mit sich bringt, bleiben ungenutzt. Was in dem genannten Beispiel fehlt, ist eine Unterstützung beim *Aufbau* einer Konferenz, z.B. Mechanismen zum Lokalisieren der Teilnehmer. Ebenso fehlen Mechanismen zur Steuerung des weiteren Konferenzablaufs, z.B. Einladen oder Ausschluß von Teilnehmern. Desweiteren wäre beim gleichzeitigen Einsatz mehrerer Medien (z.B. Audio <u>und</u> Video) auf der Teilnehmerseite eine einheitliche und einfach zu handhabende Oberfläche wünschenswert. Es sollte möglich sein, sämtliche Dienstmerkmale klassischer Telekommunikations-Einrichtungen abzubilden, z.B. Rufumleitung oder Anrufbeantworter, sofern dies jeweils sinnvoll ist.

Gebraucht wird also eine Umgebung, in der das o.g. Szenario (zwei oder mehrere Personen wollen miteinander „telefonieren“) mit der Einfachheit eines klassischen Telefongesprächs durchzuführen ist, d.h.

- das Starten und Konfigurieren der benötigten Programme sollte einfach und schnell möglich sein.
- Die Adressierung eines Teilnehmers sollte über eine leicht zu merkende und gleichbleibende Kennung erfolgen. Im Gegensatz zur klassischen Telefonie soll nicht der einzelne Apparat adressiert werden, sondern der Teilnehmer selbst, unabhängig von seinem gegenwärtigen Aufenthaltsort.
- Mit der Adressierung beginnt eine Aufbauphase, in deren Verlauf alle Kommunikationsparameter ausgetauscht werden, ohne daß andere Kommunikationskanäle (Telefon) nebenher benutzt werden müssen.
- Am Ende der erfolgreich durchgeführten Aufbauphase, ist beim gerufenen Teilnehmer die benötigte Programmumgebung bestehend aus Media Agents und Conference Controller gestartet.

In der vorliegenden Arbeit wird ein System zum Aufbau von Multimedia Konferenzen entwickelt und implementiert. Aufgrund des Themenumfanges beschäftigt sich meine Arbeit nur mit einem Teil, während der zweite von [12] bearbeitet wird. Eine genaue Abgrenzung der Arbeiten ist am Ende von Kapitel 3 zu finden.

Zum Inhalt dieser Ausarbeitung

Nach dem Herausarbeiten der Problematik im Kapitel 2 wird in Kapitel 3 die Architektur eines Multimedia Conferencing Systems (MUCS) entwickelt. Kapitel 4 stellt das dem System zugrundeliegende Protokoll, das sog. MUCSP vor. Kapitel 5 beschreibt die Implementierung eines Conference Controllers, der quasi den Kern meiner Arbeit bildet. Das letzte Kapitel versucht Schwächen des entwickelten Systems sowie zukünftige Entwicklungsrichtungen aufzuzeigen.

Kapitel 2

Problemstellung

2.1 Einleitung

In einer Multimedia Konferenz kommunizieren üblicherweise mehrere Teilnehmer über verschiedene Medien (Audio, Video, Whiteboard, etc.) miteinander. Aufnahme, Verarbeitung (z.B. Komprimierung) und Wiedergabe der dabei anfallenden Datenströme übernehmen sog. Media Agents. So existieren Media Agents, die Audio- (z.B. NEVOT, vat) oder Videodaten (IVS, nv, vic) übertragen können, aber auch „elektronische“ Whiteboards, bei denen mehrere voneinander entfernte Benutzer quasi auf einer gemeinsamen Tafel zeichnen können. Ähnliches ist für Editoren und CAD-Programme denkbar. Der parallele Einsatz mehrerer Medien in einer Konferenz führt zu dem Begriff „Multimedia Konferenz“.

2.2 Konferenzentstehung

Für die Codierung und Übertragung der Mediendatenströme existiert eine Vielfalt an fortgeschrittenen Formaten und Protokollen, die auch bei den relativ niedrigen Übertragungsraten in heute verbreiteten LANs[1] einen sinnvollen Einsatz der Media Agents ermöglichen. Für die Teilnahme an einer Konferenz muß jeder Benutzer die gewünschten Agenten starten und mit den benötigten Parametern versorgen. Letztere müssen ihm hierfür natürlich bekannt sein. Für Konferenzen mit mehr als zwei Teilnehmern kommt der für die Übertragung des Mediendatenstromes benutzten IP-Multicast-Adresse eine besondere Bedeutung zu. Sie stellt quasi, ähnlich wie bei Fernsehsendern, die Kanalnummer des jeweiligen Programms dar. Jeder, der diese Nummer kennt, kann den Fernsehsender empfangen bzw. an der Konferenz teilnehmen. Im Gegensatz zu den Nummern der Fernsehkanäle ist die Multicast-Adresse dynamisch, d.h. sie ist nur für die Dauer der Konferenz gültig. Ihre Vergabe stellt eine Problematik dar, auf die in dieser Arbeit nicht weiter eingegangen wird. Für die Teilnahme an der Konferenz muß die Multicast-Adresse allen Teilnehmern bekannt sein. Für ihre Verteilung bzw. für die netzweite Ankündi-

[1] Gemeint sind vor allem solche auf Ethernetbasis mit Übertragungsraten unterhalb 10Mbit/s

gung der Konferenz mit all ihren Parametern existiert das von Van Jacobsen [3] entwickelte Session Directory `sd`. Während sein Einsatz für feststehende Konferenztermine (wie regelmäßige Abteilungssitzungen etc.) durchaus adäquat erscheint, ist sie für die Initiierung spontaner Konferenzen ungeeignet. Will ein Benutzer eine Konferenz ohne vorherige Absprache mit den Teilnehmern initiieren, so hat er hierzu keine Möglichkeit. Dem Initiator einer Konferenz bleibt in diesem Fall nur die Benutzung des Telefons, was nicht nur umständlich und teuer, sondern bei vielen Teilnehmern auch aufwendig ist. Das Verschicken von e-mails wäre eine Möglichkeit, die jedoch an der oftmals mehrere Stunden betragenden Zustellzeit scheitert.

2.3 Problembeschreibung

Das, was hier also gebraucht wird, ist eine „Signalisierungseinrichtung“, mit der man ähnlich wie beim Telefon einen Benutzer anrufen kann. Um dabei nicht auf der Technologiestufe eines ISDN-Bildtelefons stehenzubleiben, sollten nicht nur die wesentlichen Dienstmerkmale klassischer Telekommunikationseinrichtung (Rufumleitung, Anklopfen, Liste der Anrufer während Abwesenheit) realisiert werden, sondern auch neue, die sich z.B. durch das Einbeziehen von Personenlokalisierungssystemen (z.B. Active Badges, siehe [11, 10, 2]) ergeben.

2.4 Geforderte Dienstmerkmale

Durch eine geeignete Kommunikationsinfrastruktur sollten folgende Punkte realisiert werden:

Personengebundenheit:

Anders als beim Telefon, wo die Rufnummer einem Anschluß zugeordnet ist, sollen die *Teilnehmer* adressiert werden. Die Benutzung der bereits etablierten e-mail-Adressen bietet sich hier an.

Anruf:

Das Einladen bzw. Hinzubeten der Konferenzteilnehmer soll ähnlich wie beim Telefonieren geschehen. Durch Eingabe der Teilnehmerkennung(en) sollen die Teilnehmer angewählt werden. Bei dieser Signalisierung sollen gleichzeitig die Konferenzparameter wie verwendete Medien, Formate, Protokolle, etc. ausgetauscht werden.

Rufannahme, Ablehnung und Umleitung:

Über entsprechende Einrichtungen soll der Angerufene einen Anruf akzeptieren, ablehnen oder umleiten können. Die Ablehnung sollte unter Angabe eines Grundes möglich sein.

Neben diesen grundlegenden Forderungen sollten weitere Möglichkeiten realisiert werden:

Verhalten bei Abwesenheit:

Mit einfachen Mitteln soll ein Benutzer festlegen können, wie Anrufe in seiner Abwesenheit behandelt werden sollen (Ablehnen, Umleiten, Anrufbeantworterfunktion).

Bedingte Anrufdurchstellung:

Mit verschiedenen Parametern soll ein Benutzer festlegen können, daß

- Anrufe von bestimmten Personen nicht mehr signalisiert werden sollen,
- nur eine bestimmte maximale Zahl von gleichzeitigen Konferenzen geführt und dementsprechend Anrufe nicht mehr durchgestellt werden sollen.
- Nur Anrufe einer bestimmten einstellbaren Dringlichkeit sollten durchgestellt werden.

Verleugnungsfunktion:

Auch aus Datenschutzgründen ist zu überlegen, ob eine Verleugnungsfunktion implementiert werden soll. Mit dieser Funktion täuscht der Teilnehmer dem Anrufer vor, daß er nicht da ist. Bei einer „normalen“ Rufablehnung erführe der Anrufer durch die Art der Rückmeldung, von der Anwesenheit des Gerufenen.

Sammelruf:

Beim „Anwählen“ einer Gruppe von Teilnehmern soll der Anrufer entscheiden können, ob er nur mit einem (dem ersten, der annimmt) oder mit allen Teilnehmern „sprechen“ möchte (reach first, reach all).

Konferenzbeendigung:

Es sollte eine Möglichkeit geben, eine Konferenz als beendet zu erklären. Zu diesem definierten Ende könnte dann z.B. die Multicast-Adresse „freigegeben“ werden.

Die Realisierung dieser Forderungen sollte zu einer einfach handhabbaren und ausfallsicheren Lösung führen, deren Bedienung und Administration letztendlich nicht viel komplizierter als die eines Telefonapparats sein sollte. Zweckmäßig wäre *eine* Instanz, mit der alle Komponenten des Systems gesteuert werden können.

Kapitel 3

Die Systemarchitektur

3.1 Einleitung

Ausgehend von den im vorangegangenen Kapitel vorgestellten Anforderungen, ist in Zusammenarbeit mit [12] eine Client-Server-Architektur entstanden, die wir Multimedia Conferencing System kurz MUCS nennen, und die in diesem Kapitel vorgestellt wird.

3.2 Überblick

Aufgrund der Anforderungen in Hinblick auf Verfügbarkeit und Stabilität ist ein *dezentrales* System entstanden, das im wesentlichen aus zwei Komponenten besteht: Einem Conference Controller (cc) und einem Conference Daemon (cd). Die Implementierung des ersten ist Gegenstand dieser Arbeit und im Kapitel 5 beschrieben. Die Beschreibung der Implementierung des Conference Daemons ist in [12] zu finden. Die in früheren Überlegungen entwickelte zentrale Lösung, die innerhalb einer Domain ähnlich einer Telefonzentrale zentraler Punkt ist, über den alle Kommunikationsverbindungen laufen, haben wir fallengelassen.

Abbildung 3.1 zeigt die Komponenten von MUCS. Das Protokoll, mit dem diese Komponenten miteinander kommunizieren nennen wir MUCSP. Es lehnt sich an andere Internet-Protokolle wie SMTP und HTTP an. Der in Abbildung 3.1 gezeigte Conference Controller ist die Instanz, mit der der Anwender Konferenzen eröffnet, indem er seine Media Agents konfiguriert und die Teilnehmer einlädt. Der Conference Daemon ist die Instanz, die die Lokalisierung des gerufenen Teilnehmers sowie die Rufsignalisierung übernimmt. In der Abbildung ist bereits erkennbar, daß der Conference Daemon auf jedem Rechner läuft, der in MUCS eingebunden sein soll. Um eine permanente Erreichbarkeit zu gewährleisten, muß der cd als Hintergrundprozeß laufen bzw. wird bei Bedarf von `inetd` gestartet. Ohne den Conference Daemon kann der Benutzer mittels Conference Controller zwar Verbindungen zu anderen Teilnehmern aufbauen, bleibt selber jedoch unerreichbar.

3.3 Die Adressierung

Als Adressierungschema werden die bereits etablierten e-mail-Adressen in der Form *user@host* bzw. *user@domain* verwendet. Der Grund hierfür ist einfach: Praktisch jeder Internet-Benutzer besitzt bereits eine solche Adresse. Im Vergleich zu Telefonnummern haben e-mail-Adressen den Vorteil, daß sie den Teilnehmer und nicht einen Anschluß oder einen Rechner adressieren[1]. Desweiteren sind e-mail-Adressen leichter als die Ziffernfolge einer Telefonnummer zu merken. Die Einführung eines neuen Adressierungsschemas wäre zudem schwer durchzusetzen.

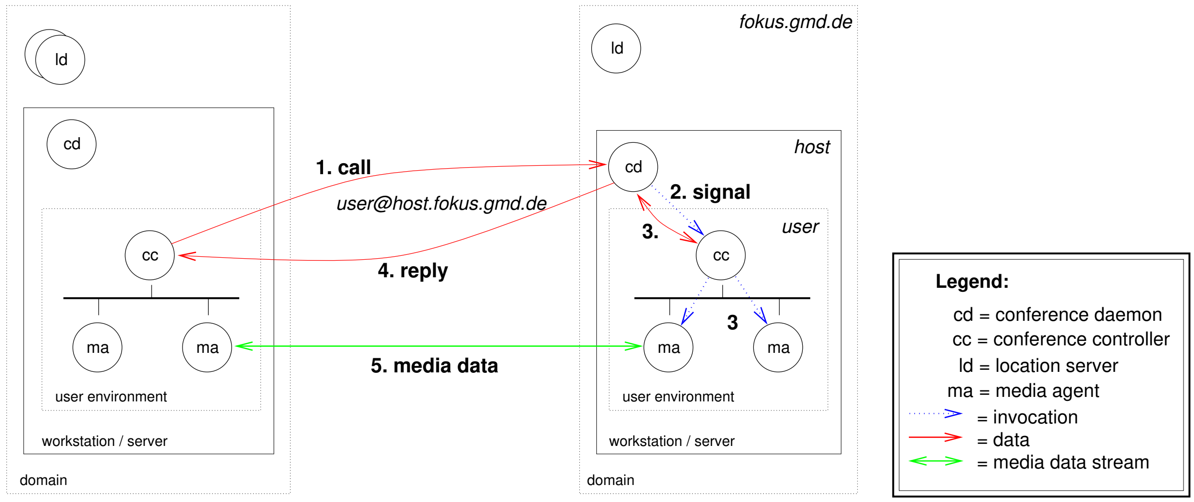

Abb. 3.1: Ablauf eines Anrufes im MUCS-System

3.4 Anruf

Abbildung 3.1 zeigt den Ablauf eines Anrufes. Die Ziffern an den Pfeilen geben die zeitliche Reihenfolge der einzelnen Schritte wieder. Will ein Benutzer einen Anruf zu einem oder mehreren Teilnehmern initiieren, so bedient er sich hierfür des Conference Controllers. Die Adressen häufig angewählter Teilnehmer werden zweckmäßigerweise in einem Verzeichnis vom Conference Controller geführt, so daß sich die Anwahl auf das Anklicken der entsprechenden Einträge beschränkt. Nach dem gleichen Schema, wie beim Verschicken einer e-mail, versucht der Conference Controller die gewählten e-mail-Adressen mittels DNS in IP-Adressen aufzulösen. Mit den so erhaltenen Adressen baut der Conference Controller eine TCP-Verbindung zu den entfernten Rechnern (1.) auf, auf denen der Conference Daemon auf einer definierten Portnummer lauscht. Eine TCP-Verbindung bietet hier die erforderliche Sicherheit, die für die sofortige Signalisierung eines Anrufes benötigt wird. Da die Verbindung nur für die Dauer der Signalisierungsphase existiert, bestehen hinsichtlich der Systemressourcen keine Bedenken. Für die Portnummer ist in `/etc/services` der Name `mucs` definiert.

[1] Vorausgesetzt, der Rechnername wird nicht angegeben!

3.4.1 Der Conference Daemon

Der Conference Daemon hat zunächst die Aufgabe festzustellen, ob der gerufene Benutzer in seiner Domain überhaupt existiert. Wenn nicht, so erhält der Anrufer eine Fehlermeldung. Existiert der gerufene Benutzer, so muß der cd nun feststellen, ob und wo der Benutzer anwesend ist. Dies ist denn auch seine Hauptaufgabe, wofür er sich unter anderem einer weiteren Instanz, dem Location Server bedient.

3.4.2 Der Location Server

Der Location Server hat die ausschließliche Aufgabe, alle Rechner in der eigenen Domain herauszufinden, an denen der Benutzer eingeloggt, sprich: vermutlich anwesend, ist. Für diesen Zweck bedient er sich verschiedener Informationsquellen, derer es in UNIX-Systemen zahlreiche gibt (z.B. die Datei `/etc/utmp`).

3.4.3 Rufsignalisierung

Hat der Conference Daemon den Benutzer lokalisiert, wird geprüft, ob eine Rufumleitung oder andere Aktionen, die die Durchstellung verhindern, definiert wurden. Hierfür wird ein Programm oder Script ausgeführt (Call Handling File), dessen Inhalt vom Benutzer beliebig gestaltet werden kann. Terminiert das Programm mit „positivem“ Returncode, so erfolgt die Rufsignalisierung. Hierzu wird ein weiteres Programm oder Script gestartet, in welchem letztendlich der Conference Controller zwecks Rufsignalisierung gestartet wird. Über ein geeignetes Fenster hat der Angerufene nun die Möglichkeit den Anruf entgegenzunehmen, abzulehnen oder umzuleiten. Hiermit endet die Anrufphase.

Für beide Aufgaben bietet sich die Verwendung von Shell-Scripts an, da diese ohne großen Aufwand erstellt und verändert werden können. Dennoch stellt das manuelle Erstellen von Scripts bzw. eines Programms keine besonders komfortable Lösung dar. Der Benutzer sollte vielmehr über ein Fenster des Conference Controllers die Möglichkeit haben, den Inhalt der Scripts durch grafische Bedienungselemente zu bestimmen. Im folgenden ist ein Ausschnitt eines möglichen Call Handling Files gezeigt:

```
Priority: urgent       {ring firebells.au}
From: my_boss          {ring hail_to_chief.au}
From: @ibm.com         {reject busy}
Subject: MUCS          {forward czahl@oz.de}
if {$hour > 17} {
  forward fo@home -reason "Gone home for the day"
}
```

Bei der Beschreibung des Anrufablaufes wird schnell klar, daß eine sinnvolle Umsetzung nur auf Rechnersystemen mit grafischen Oberflächen, also Window-Systemen, möglich ist. Durch die unterschiedlichen Anmeldearten (`telnet`, `rlogin`, `xdm`, etc.) auf UNIX-Systemen ergeben sich bei der Rufsignalisierung besondere Probleme, wenn der Benutzer z.B. an einem ASCII-Terminal oder über ein X-Terminal an mehreren Rechnern gleichzeitig angemeldet ist. Die Lösung dieser Probleme sowie die Realisierung des Location Servers ist Gegenstand der Arbeit von [12].

3.5 Die Media Agents

Nach der Rufannahme werden beim Angerufenen automatisch die benötigten Media Agents gestartet. Welche dies sind, hängt zum einen von den vom Anrufer geforderten Parametern (Protokolle, Formate, etc.) und zum anderen von den beim Angerufenen vorgenommenen Konfigurationen ab. Benutzt der Anrufer z.B. Video mit MPEG II Kodierung und Audio im LPC-Format, so kann eine Konferenz auch dann stattfinden, wenn der Angerufene keine Video-Hardware besitzt oder keinen Agenten, der das MPEG II Format versteht. In diesem Fall würde mit den verbleibenden Medien (hier Audio) die Konferenz dennoch geführt werden können. Sinnvoll, in der gegenwärtigen MUCS-Version aber nicht realisiert, wäre eine Liste von Alternativ-Parametern, die der Anrufer angibt. Auch wäre die Möglichkeit, Mindestanforderungen zu definieren, sinnvoll. Sollen beispielsweise Audio und Video in einer Konferenz benutzt werden, so ist es im Allgemeinen nicht sinnvoll, die Konferenz ohne Ton zu führen, wenn der Angerufene nur über Video-Möglichkeiten verfügt.

3.6 Redirect

Bei einem Anruf kann es vorkommen, daß der Location Server feststellt, daß sich der Gerufene an *einem* anderen oder an *mehreren* Rechnern (einschließlich oder ausschließlich des eigenen) befindet. In diesem Fall muß der Anrufer von dem oder den anderen Aufenthaltsorten des Gerufenen in Kenntnis gesetzt werden, damit an den entsprechenden Rechnern eine Rufsignalisierung erfolgen kann. In einem solchen Fall, in dem sich der Gerufene zusätzlich oder ausschließlich an anderen Rechnern innerhalb der selben Domain befindet, erfolgt ein *Redirect* durch den Conference Daemon. Bei diesem Redirect werden dem Anrufer alle alternativen Rechneradressen mitgeteilt. In Abbildung 3.2 ist ein Szenario dargestellt, bei dem der Teilnehmer *user@fokus.gmd.de* gerufen wird. Da im Domain-Teil der e-mail-Adresse kein Rechnername angegeben ist, wird beim Auflösen der Rufadresse vom DNS die Adresse des in der Domain als Mailexchanger definierten Rechners zurückgeliefert (hier `hostA`). Da der gewünschte Benutzer jedoch nicht an diesem sondern an `hostB` sitzt, erfolgt ein Redirect (3.). Mit der so erhaltenen Adresse von `hostB` nimmt der Anrufer nun Kontakt auf (4.).

3.7 Forwarding

Neben dieser durch den Conference Daemon eingeleiteten Rufumleitung, dem Redirect, existiert eine vom Benutzer initiierte Umleitung, deren Funktionsweise zwar mit der des Redirect identisch ist, beim

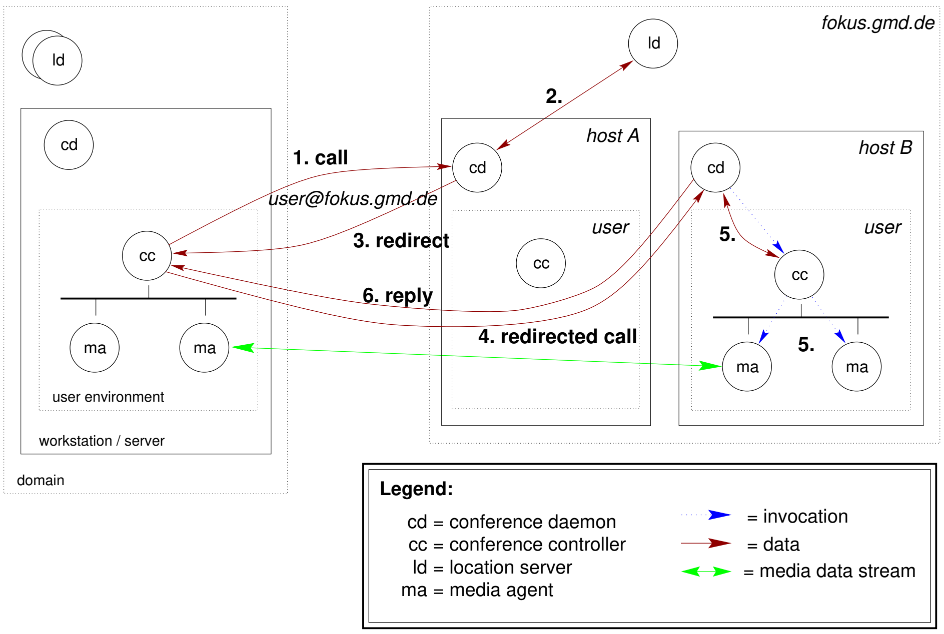

Abb. 3.2: Rufumleitung im MUCS-System

Anrufer u.U. aber unterschieden werden soll.

3.8 Adreß-Caching

Unter der Annahme, daß Benutzer generell am gleichen Arbeitsplatz bzw. Rechner arbeiten, wäre es sinnvoll, wenn sich der Anrufer die durch ein Redirect bzw. permanente Umleitung erhaltene neue Adresse merkt, damit diese später wiederverwendet werden kann. Die Implementierung eines solchen Caches im Conference Controller würde vor allem für die Mailexchanger eine erhebliche Entlastung bedeuten und die Anrufphase verkürzen.

3.9 Zyklenvermeidung bei Rufumleitung

Beim Redirect und beim Forwarding muß der Conference Controller durch geeignete Mechanismen sicherstellen, daß keine Zyklen entstehen. Auch kann es vorkommen, daß ein Benutzer an dem gerufenen Rechner *und* an anderen anwesend ist. In diesem Fall erhielte der Anrufer ein Redirect mit den Adressen der anderen Rechner einschließlich der des bereits gerufenen. Da aus Gründen der Vereinfachung des Ablaufes in der Signalisierungsphase immer zwei Pakete ausgetauscht werden, endet die Kommunikation zwischen cc und cd nach Erhalt des Redirect. Durch ein Flag im Request-Paket muß beim zweiten Anruf verhindert werden, daß die gerufenen Rechner erneut ein Redirect liefern. Hierfür wurde im Anrufpaket das Tag `This-Host-Only:` eingeführt, dessen Wert auf `TRUE` gesetzt wird. Die Abwesenheit dieser Zeile impliziert den Wert `FALSE`.

3.10 Proxy-Mode

Alternativ zur vorgestellten Form des Redirect wäre auch eine Weiterleitung durch den Conference Daemon möglich, da dieser die neue Adresse als erster kennt. Der zuerst angewählte Conference Daemon übernähme dann also eine Proxy-Funktion, die sich allerdings über beliebig viele Conference Daemons fortsetzen könnte. Hierdurch würde nicht nur der Signalisierungsweg unkontrollierbar groß, auch die Erkennung von Zyklen würde erschwert. Aus diesen Gründen wird in der gegenwärtigen MUCS-Version von der Implementierung des Proxy-Modes abgesehen.

3.11 Lokale Konfiguration

In der aufgezeigten Architektur wird der Conference Controller beim Anrufer zwecks Rufsignalisierung gestartet. Als Ergebnis einer positiven Quittung durch den Anrufer soll dessen lokale Konferenzumgebung, also die benötigen Media Agents gestartet und mit den Konferenzparametern versorgt werden. Für diesen Zweck wird ein lokaler Kommunikationsmechanismus benötigt, über den die Agenten mit dem Controller kommunizieren können.

In [6] wird hierfür eine Lösung, in Form eines Nachrichtenverteilerprozesses vorgeschlagen. Alle lokalen Komponenten (Media Agents, Conference Controller), die in das System eingebunden werden sollen, melden sich bei diesem Prozeß an und bleiben mit ihm verbunden (via TCP-Verbindung oder lokalem Multicast). Ausgetauscht werden Informationen in Form von ASCII-Zeilen. Prozesse, die Informationen für andere haben, senden diese an den Verteiler. Prozesse, die Interesse an Informationen haben, tun dies in Form eines regulären Ausdrucks kund. Aufgrund der Verwendung regulärer Ausdrücke sowie des Multicasts wird dieser Prozeß mit „Pattern Matching Multicast“ kurz `pmm` bezeichnet. Abbildung 3.3 zeigt die durch den Einsatz von `pmm` entstehende lokale Kommunikationsinfrastruktur.

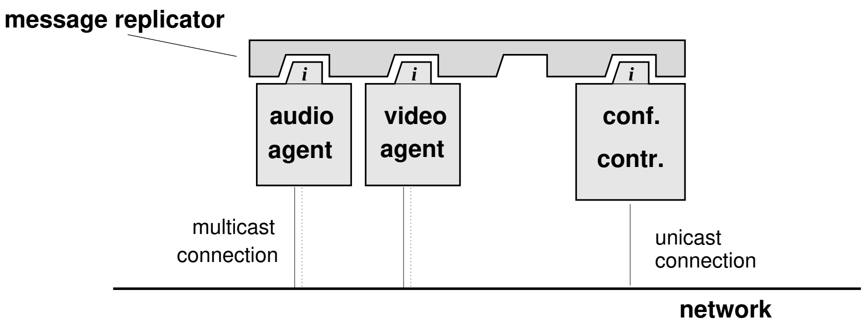

Abb. 3.3: Lokale Kommunikationsinfrastruktur

In Anlehnung an die Darstellung sowie der angestrebten Austauschbarkeit der Komponenten könnte man `pmm` auch als lokalen Software-Bus betrachten. Media Agents, die mit `pmm` kommunizieren wollen, müssen sich an das in [7] beschriebene Format halten. Derzeit sind nur der in [8] beschriebene Audio-Agent NEVOT sowie der in [4] beschriebene Video-Agent `vic` zu `pmm` kompatibel. Jedoch ist der Änderungsaufwand für andere bestehende Media Agents relativ gering.

Kapitel 4

Das MUCS-Protokoll

4.1 Überblick

Der Aufbau des für die Kommunikation zwischen Conference Controller (cc) und Conference Daemon (cd) verwendeten MUCS-Protokolls (MUCSP) orientiert sich am HTTP. Dementsprechend werden zwischen cc und cd Pakete in Form von ASCII-Zeilen ausgetauscht. Da die ausgetauschten Pakete immer aus mehreren Zeilen bestehen, deren Reihenfolge und Anzahl jedoch variabel ist, wird das Ende eines Paketes durch eine Leerzeile gekennzeichnet. Die Gesamtgröße eines Paketes ist lediglich durch die Implementierung des Conference Daemons limitiert. Entsprechend den Funktionen eines Paketes werden Anrufpakete (request packets) und Antwortpakete (response packets) unterschieden.

4.2 Das Anrufpaket

Ein Anrufpaket wird vom Conference Controller des anrufenden Teilnehmers generiert und beginnt immer mit der Zeile Call <user> MUCS/1.0. Der gerufene Benutzer ist in <user> in der Form <username>@<hostname> angegeben. Mit MUCS/1.0 wird dem Angerufenen die beim Anrufer implemenierte MUCS-Version mitgeteilt, derzeit 1.0. Die folgenden Zeilen, deren Reihenfolge irrelevant ist, beschreiben weitere Anrufparameter. Jede Zeile ist in der Form <tag>: <value> aufgebaut, wobei <tag> ein reserviertes Wort ist und <value> eine beliebige ASCII-Zeichenfolge durch Newline terminiert. Für das Anrufpaket sind die in Tabelle 4.1 aufgezählten Tags definiert.

Die formale Spezifikation des Protokolls findet sich in Anhang C wieder. Obwohl die Namensgebung der Tags teilweise mit der des in [1] beschriebenen Session Directory Protokolls (SDP) identisch ist, wurde bei der Wahl des Formates bewußt auf eine Kompatibilität zu SDP verzichtet. Die Einsatzbereiche von MUCSP und SDP erschienen zu unterschiedlich. Mit SDP werden Multimedia Sessions angekündigt, während mit MUCSP gezielt Benutzer „angesprochen“ und zu einer Multimedia-Konferenz eingeladen werden. Handelt es sich im einen Fall um eine unidirektionale Kommunikation, so führt die Rückantwort

Tag	Bedeutung
Info	Konferenzbezeichnung
Origin	Absenderkennung in Form einer e-mail Adresse
Media	Beschreibung der verwendeten Medien
Priority	Priorität des Anrufes
Face	xface bitmap string
This-Host-Only	Verhindert Zyklen bei Redirects

Tabelle 4.1: Tags des Anrufpaketes

bei MUCSP zu einer bidirektionalen Kommunikation, die im SDP nicht vorgesehen ist.

Beipiel

Will der Benutzer `herbie` in `domainA` den Benutzer `alf` in `domainB` erreichen, sendet herbies Conference Controller folgendes Paket an den Conference Daemon, der auf alfs Rechner läuft:

```
Call alf@domainB MUCS/1.0
Origin: herbie@domainA
Info: Monday morning meeting
Media: audio VAT PCMU 8000 224.1.1.2 4567
Priority: normal
```

Der für `domainB` zuständige Conference Daemon wird über DNS ermittelt. Dieser gibt das Paket unverändert an den von ihm gestarteten Conference Controller (siehe 5.4) weiter. Der cd interpretiert lediglich die erste Zeile um den zu rufenden Benutzer festzustellen, die restlichen Zeilen werden unabhängig von ihrem Aufbau durchgereicht.

Die Angabe der auf die `Call`-Zeile folgenden Zeilen ist zwar optional[1], jedoch ist offensichtlich, daß ein Anruf nur dann sinnvoll ist, wenn der Angerufene erfährt, mit welchem Medien und auf welchen Adressen er an der Konferenz teilnehmen soll. Diese Informationen werden in der Media-Zeile übertragen, welche aus diesem Grund obligatorisch ist. Die Zeilen Origin, Info, Priority sind optional, weil eine Verbindung auch ohne sie möglich ist.

4.3 Das Antwortpaket

Ein Antwortpaket beginnt immer mit einer Zeile der Form `MUCS/1.0 <nnn> <message>`, wobei `<nnn>` analog zum HTTP ein dreistelliger Returncode und `<message>` eine beliebige Zeichenkette ist. Tabelle 4.2 zeigt die Bedeutung der ersten Ziffer des Returncodes.

[1] dies gilt nicht für die das Paket terminierende Leerzeile

Ziffer	Bedeutung
1	Informationszeile
2	Erfolgreich ausführter Request.
3	Rufumleitung
4	Client-Fehler
5	Server-Fehler

Tabelle 4.2: Bedeutung der ersten Ziffer des Returncodes

Existiert der Benutzer nicht in der Domain, innerhalb der cd läuft, so schickt dieser folgende Antwort zurück:

```
MUCS/1.0 405 Unknown
```

Diese Fehlermeldung wird durch eine Leerzeile terminiert, da keine weiteren Informationen folgen. Wenn der Benutzer zwar existiert aber nicht angemeldet ist, so schickt der cd

```
MUCS/1.0 404 Not Found
```

Der anrufende Client (Conference Controller) kann nun entweder den wenig aussagekräftigen Text „Not Found" als Fehlermeldung anzeigen oder aber anhand des Returncodes eine eigene Meldung generieren.

Konnte der gerufene Benutzer gefunden und ihm der Anruf signalisiert werden, so erzeugt der cd keinen Returncode. Dies geschieht stattdessen durch den Conference Controller, der die Rufsignalisierung beim Angerufenen übernimmt. In einem Fenster kann sich der Angerufene für eine Annahme, Ablehnung oder Umleitung des Rufes entscheiden. Entsprechend diesen Möglichkeiten erzeugt der Conference Controller den Returncode 202, 430 oder 302. Je nach gewählter Möglichkeit enthält das Antwortpaket zusätzliche Zeilen, die Tabelle 4.3 zeigt.

Ziffer	Bedeutung
Location:	Angabe der neuen Adresse bei einer Rufumleitung
Reason:	optionale Angabe eines Grundes etc. bei Rufablehnung

Tabelle 4.3: Tags des Antwortpaketes

Das vom entfernten cc generierte Antwortpaket wird vom cd transparent an den Anrufer zurückgereicht.

Beipiel

```
MUCS/1.0 302 Forward to alf@domainC
Location: alf@domainC
```

Nach dem Eintreffen des Antwortpaketes beim Anrufer wird die Verbindung zum Conference Daemon geschlossen. Für einen Anruf werden also immer zwei Pakete zwischen cc und cd ausgetauscht.

4.4 Auflösung von Gruppennamen

Will ein Benutzer Kontakt mit einer Einzelperson oder einer Gruppe von Personen aufnehmen, so geschieht dies durch Angabe einer e-mail-Adresse. Mittels DNS wird diese beim Anrufer aufgelöst. Verbirgt sich hinter der Adresse auf der *Anruferseite* eine Gruppe von Adressen, etwa durch `/etc/aliases`, so wird zu jeder dieser Adressen eine Verbindung aufgebaut.

Verbirgt sich hinter einer Adresse auf der Seite des *Angerufenen* eine Gruppe, so wird diese vom cd aufgelöst. Jede der dabei erhaltenen Adressen wird in einer Location-Zeile des Antwortpaketes zurückgegeben. Ähnliches geschieht, wenn zwar nur eine Person gerufen wurde, der cd mittels seines Location Servers oder anderer Mechanismen festgestellt hat, daß der Benutzer an mehreren Rechnern angemeldet ist. Auch in diesem Fall erhält der Anrufer ein Antwortpaket mit mehreren Location-Zeilen. Für das obige Beispiel könnte dies so aussehen:

```
MUCS/1.0 303 Redirect
Location: alf@hostA.domainB
Location: alf@hostB.domainB
```

Der Conference Daemon teilt hier dem Anrufer mit, daß der gerufene Benutzer auf zwei Rechnern angemeldet ist, auf `hostA` und auf `hostB`. Zur Unterscheidung der vom Benutzer initiierten Rufumleitung, dem Forwarding, bezeichnen wir diese Variante der Umleitung als Redirect. Ob und wann eine Rufumleitung beim Anrufer bestätigt werden muß hängt von dessen lokalen Einstellungen bzw. der Implementation des Conference Controllers ab.

4.4.1 Returncodes

Die folgende Aufstellung zeigt die derzeit definierten Returncodes des Antwortpaketes.

Code	**Bedeutung**
1??	Informationszeilen
202	Rufannahme
301	permanente Rufumleitung (Forward)
302	temporäre Rufumleitung (Forward)
303	Redirect durch Conference Daemon
404	Der gerufene Benutzer ist nicht anwesend (eingeloggt)
405	Der gerufene Benutzer existiert nicht in der Domain
410	Timeout bei der Rufsignalisierung
430	Rufablehnung
5??	siehe [12]

Tabelle 4.4: Returncodes des Antwortpaketes

4.5 Erweiterbarkeit

Da der Conference Daemon die im Anrufpaket enthaltenen Zeilen ohne sie zu interpretieren transparent durchreicht, ist im Prinzip egal, welchen Inhalt und welches Format sie haben. Wichtig ist nur, daß der Request mit einer Leerzeile terminiert wird. Aus diesem Grund können später problemlos neue Tags definiert werden, ohne daß der Conference Daemon hierfür verändert werden muß.

Kapitel 5

Der Conference Controller

Der Conference Controller (`coco`) ist das zentrale Front-End, mit dem der Benutzer Konferenzen aufbaut, Benutzer einlädt sowie die Media Agents konfiguriert. Seine grafische Oberfläche wurde ausschließlich mit dem `tcl/tk`-Paket von [5] sowie `tcl-dp` von Brian Smith [9] entwickelt. Die Syntax der Kommandozeile lautet:

```
coco [-p <port>] [-h <hostname>]
```

Bei einer Rufsignalisierung geben die Optionen den Rechnernamen sowie die Portnummer an, über die der Conference Controller Kontakt mit dem Conference Daemon aufnehmen muß, der ihn gestartet hat.

Jede Instanz des Conference Controllers verwaltet genau eine Konferenz. Nach dem Erstaufruf des Controllers können weitere Instanzen über den `New`-Button des Hauptfensters erzeugt werden. Das Eintreffen eines Anrufes erzeugt <u>immer</u> eine neue Instanz (siehe 5.4).

5.1 Allgemeines

Beim Entwurf der Benutzungsoberfläche stand eine einfach zu bedienende und selbstdokumentierende Oberfläche im Vordergrund. Hierfür wurde nicht nur auf eine einheitliche Gestaltung aller Fenster geachtet sondern auch darauf, daß die gewünschten Funktionen mit möglichst wenigen Interaktionen durchzuführen sind. Schließlich soll die Bedienung des Systems nicht wesentlich komplizierter sein, als die eines Telefons. Mit folgenden Maßnahmen wurde u.a. versucht, diesen Zielen nahe zu kommen:

- Außer dem Hauptfenster haben alle Fenster eine Kopfzeile, mit einer Überschrift und einem Icon. Ikonifiziert wird das Fenster mit dieser Überschrift und dem Icon dargestellt.
- Alle Buttons, mit denen ein Fenster geschlossen wird, ohne eine Aktion auszulösen, sind mit dem rot geschriebenen Wort `Dismiss` beschriftet. Im Hauptprogramm wird hierfür die globale Variable `Resource(ExitButton)` definiert.
- Jedes Fenster besitzt einen `Help`-Button, über den ein entsprechendes Programm, z.B. ein WWW-Browser, aufgerufen werden kann. Im Hauptprogramm wird für diesen Button die globa-

le Variable `Resource(HelpButton)` definiert.

- Alle Eingabefelder haben einen hellblauen Hintergrund und die Schriftart Courier, welche im Hauptprogramm über die globale Variable `Resource(Entry)` gesetzt werden.
- Mit wenigen Ausnahmen haben alle Fenster am unteren Rand eine Statuszeile, deren Aussehen durch die globale Variable `Resource(Statusline)` bestimmt wird. In dieser Zeile werden verschiedene Meldungen zum Programmstatus bzw. Hinweise zu einzelnen Bedienungselementen angezeigt.

5.2 Das Hauptfenster

Abbildung 5.1 zeigt das Hauptfenster des Conference Controllers, das nach dem Starten erscheint. Das Fenster läßt sich grob in drei Teile einteilen: Im linken Drittel des Fensters befinden sich unterhalb des Logos einige allgemeine Buttons. Rechts daneben werden in einer Liste die aktuellen Konferenzteilnehmer angezeigt. In der Abbildung ist ein Teilnehmer erkennbar, der mit einem Media Agent an der Konferenz teilnimmt. Für jeden Agenten wird vor dem Teilnehmernamen ein Symbol dargestellt, daß Aufschluß den Zustand des Agenten gibt. Ein Gleichheitszeichen bedeutet, daß der Zustand des Agenten undefiniert ist, weil der Teilnehmer z.B. neu hinzugekommen ist. Die Note zeigt bei Audioagenten an, daß der Teilnehmer spricht. Durch Anklicken des Symbols wird der betreffende Agent stummgeschaltet. Diese Funktionalität ist derzeit nur für Audio-Agenten realisiert. Welche Bedeutung das Anklicken des Symbols für andere Agenttypen hat, bleibt offen. Die Statuszeile am unteren Fensterrand bildet den dritten Bereich. Wird z.B. der Maus-Cursor über einen Button bewegt, so erscheint in der Statuszeile ein kurzer Hinweis über die Bedeutung dieses Buttons.

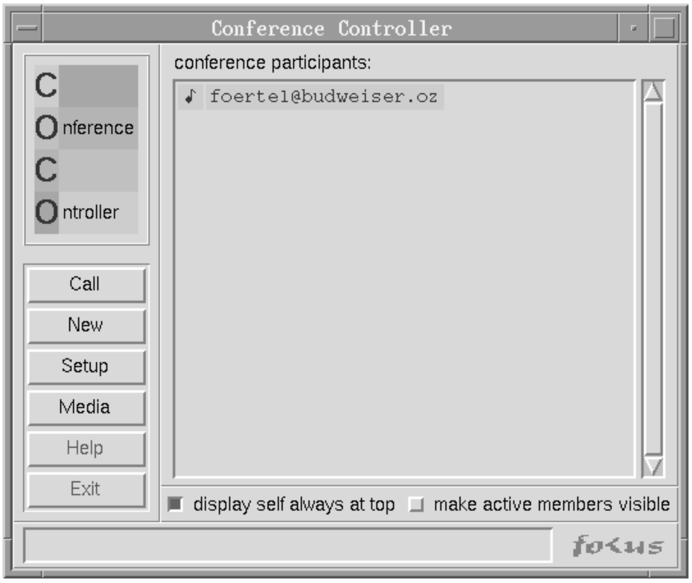

Abb. 5.1: Der Conference Controller

Der `Help`-Button startet ein unter *Setup* definiertes Programm, wie z.B. `Mosaic` oder `netscape` und übergibt diesem einen URL. Jeder `Help`-Button hängt diesem URL ein Label an, so daß alle Programmfunktionen in einem HTML-File beschrieben werden können. Der `Help`-Button des Hauptfensters erzeugt z.B. den URL `<url>#Main`, wobei `<url>` ein unter *Setup* definierbarer URL ist. Der `Media`-Button führt zum Aufruf des sog. Media-Managers (siehe 5.5). Der `Setup`-Button führt zum Setup-Window, in dem sämtliche Programmparameter eingestellt werden können (siehe 5.6).

5.3 Starten einer neuen Konferenz

Das Starten einer neuen Konferenz geschieht vom Hauptfenster aus über den `Call`-Button. Es erscheint das in 5.2 gezeigte Fenster, in dem die Konferenzparameter eingetragen werden. Das Feld *Subject* ist derzeit mit der Prozeß-Id des Conference Controlers vorbelegt. Die Priorität ist ein Wert, mit dem der Benutzer die Dringlichkeit seines Rufes beim Anrufer signalisieren kann. Derzeit werden `emergency`, `urgent` und `normal` unterschieden, welche mittels Radiobutton ausgewählt werden. Die Auswahl der für die Konferenz zu benutzenden Media-Agents erfolgt über die Auswahlliste, deren Inhalt mit Hilfe des Media-Managers definiert wird. Durch Betätigen der Checkbuttons können die gewünschten Agents ausgewählt werden. Die im Media-Manager als Default markierten Agents sind bereits aktiviert (hier NeVoT). In den Eingabefeldern hinter den Agentenbezeichnungen können die für die Medienübertragung erforderlichen Multicast-Adressen und Portnummern eingegeben werden. Derzeit ist unklar, wie dies automatisiert werden kann.

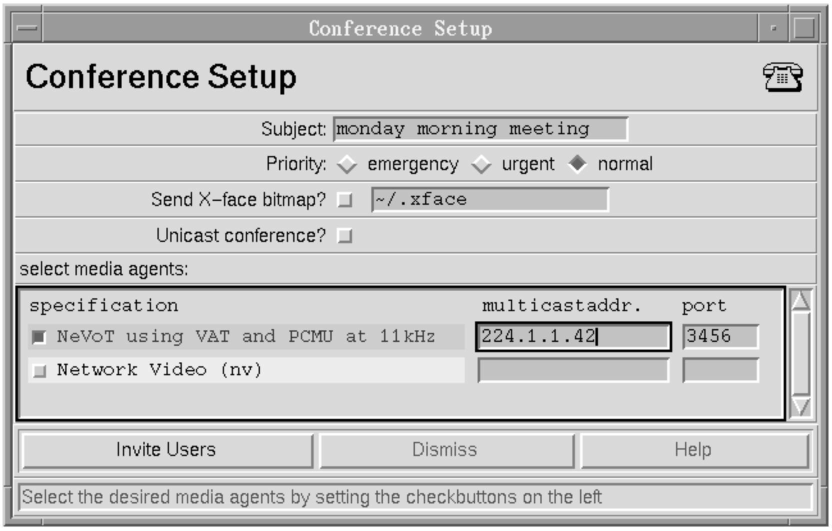

Abb. 5.2: Conference Setup Window

Nach Auswahl der Agenten können die Benutzer eingeladen werden. Hierzu dient der `Invite Users`-Button. Abbildung 5.3 zeigt die Bedienungsoberfläche des sog. Dialers, über den sich bis zu 20 Benutzer gleichzeitig einladen lassen. Die Zahl 20 ist willkürlich festgelegt und kann im Programm problemlos geändert werden. Hinter jedem der Eingabefelder befindet sich eine Statuszeile, die Aufschluß über den Erfolg des Anrufes gibt. Dies ermöglicht eine kompaktere Darstellung, als

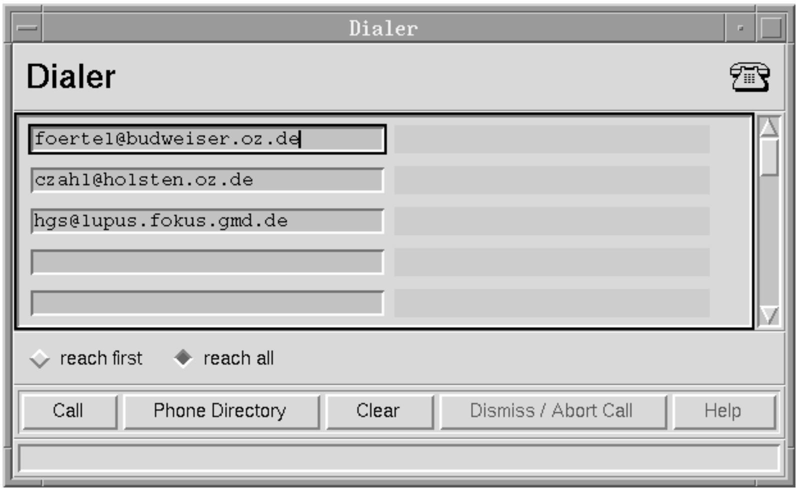

Abb. 5.3: Der Dailer

z.B. beim Einsatz von Dialogboxen. Würden nur 10 Teilnehmer eingeladen, so würde die Anzeige der Ergebnisse in Dialogboxen schnell unüberschaubar werden. Der `Clear`-Button löscht alle Adreß-Eingabefenster sowie die Statuszeilen dahinter. Der `Phone Directory`-Button ruft das sog. Telefonbuch auf, welches z.Z. nur aus einer Liste der in `~/.mailrc` eingetragenen aliase besteht (siehe Abbildung 5.4). Durch Anklicken eines der aliase werden die entsprechenden e-mail-Adressen in die Eingabefelder eingetragen.

Die Eingabe der Adressen kann auch manuell erfolgen. Über den `Call`-Button wird der Verbindungsaufbau initiiert. Jede der eingegebenen e-mail-Adressen wird hierfür zunächst per DNS in eine IP-Adresse umgewandelt. Danach wird versucht, für jede Adresse eine TCP-Verbindung zu dem unter der jeweiligen Adresse laufenden Conference Daemon aufzubauen. Hierfür wird die unter dem Namen `mucs` in `/etc/services` eingetragene Portnummer verwendet. Das Ergebnis des Verbindungsaufbauversuches wird jeweils in dem Statusfeld hinter dem Adreßfeld angezeigt. Auch die Statusmeldungen des Conference Daemons werden hier angezeigt. Haben alle eingeladenen Teilnehmer den Anruf akzeptiert, verschwindet das Dialer-Fenster, da in diesem Fall keine weitere Aktion in dem Fenster erforderlich ist. Die eingeladenen Benutzer erscheinen in der Liste des Hauptfensters. Hat der Angerufene manuell eine Rufumleitung vorgenommen oder permanent definiert, so erscheint beim Anrufer eine Dialogbox, in der die Umleitung bestätigt werden muß. Diese Bestätigung kann per *Setup* unterdrückt werden. Bei einem Redirect durch den Conference Daemon wird automatisch die neue Adresse in das jeweilige Eingabefeld im Dialer eingetragen und angewählt.

Abb. 5.4: Phone Directory

5.4 Die Rufsignalisierung

Die Signalisierung eines Rufes übernimmt der Conference Controller, der hierfür vom Conference Daemon beim Angerufenen über ein Script gestartet wird. Existiert im Homedirectory des Benutzers ein Script namens `~/.mucs-startCc`, so wird dieses benutzt. Anderenfalls erfolgt die Signalisierung durch das Standardscript `startCc` in `/usr/local/mucs/etc`. In diesen Scripts muß der Conference Controller mit der Option `-p` aufgerufen werden, durch welche er die Portnummer mitgeteilt bekommt, auf der er Kontakt mit dem Conference Daemon aufnehmen muß. Wie bereits erwähnt, wird beim Eintreffen eines Anrufes immer ein neue Instanz des Conference Controllers erzeugt. Zur Anrufsignalisierung wird das in 5.5 gezeigte Fenster benutzt, welches per *Setup* auch ikonifiziert gestartet werden kann und somit weniger Platz in Anspruch nimmt. Eine zusätzliche optionale akustische Signalisierung kann durch Abspielen eines Audio-Files erfolgen. Im oberen Bereich des Fensters wird die e-mail-Adresse von Anrufer und Angerufenen zusammen mit dem Konferenznamen, der Priorität und den

Medien-Parametern angezeigt. Für jedem beim Anrufer gestarteten Media-Agent wird eine Zeile mit der Medien-Beschreibung angezeigt, welche z.Z aus dem Medientyp (Audio, Video, Whiteboard, Text), dem Protokoll (RTP oder VAT), dem Übertragungsformat, der Übertragungsrate sowie der Multicast-Adresse und Portnummer besteht. Konnten für ein Medium lokal keine passenden Agenten gefunden werden, so wird die entsprechende Zeile rot dargestellt. Wird für mindestens ein Medium ein passender Agent gefunden, erfolgt die Rufsignalisierung. Konnte für keinen der vom Anrufer verwendeten Medientypen ein passender Agent gefunden werden, so wird die Signalisierung unterdrückt. Der durch das Start-Script erzeugte Conference Controller schickt dem Angerufenen stattdessen eine Mail. Über die mit `Accept`, `Reject` und `Forward` beschrifteten Buttons kann der Benutzer den eingegangenen Ruf annehmen, ablehnen oder umleiten.

5.4.1 Signalisierungsdauer

Die Zeit, die für die Rufsignalisierung insgesamt benötigt wird, ist zum einen von der Entfernung der beteiligten Workstations im Netz abhängig. Zum anderen aber auch von der Gestaltung des Call Handling Files. Im Falle unserer recht langsamen Testumgebung, mit mehreren Workstations im gleichen Netzsegment, vergingen bis zu 7 sec, bis ein Anruf signalisiert wurde. Dies lag unter anderem daran, daß wir mit Shell-Scripten gearbeitet haben, aber auch an dem zu geringen Arbeitsspeicher der beteiligten Rechner.

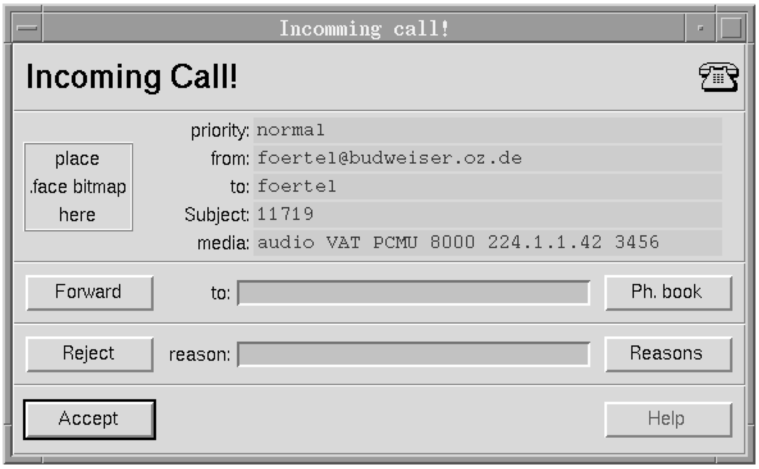

Abb. 5.5: Front-End zur Rufsignalisierung

5.4.2 Rufablehnung

Bei einer Ablehnung kann der Benutzer in dem Eingabefeld rechts neben dem `Reject`-Button eine Begründung eingeben, bzw. mit dem `Reasons`-Button einen vordefinierten Text aus einer Liste durch Doppelklicken auswählen. Diese Gründe sind frei wählbare Texte, die in dem in Abbildung 5.6 gezeigten Fenster verwaltet werden. Dieses Fenster ist auch über den `Setup`-Button des Hauptfensters erreichbar. Mit dem `Add`-Button kann ein Grund in die Liste eingetragen werden. Mit dem `Delete`-Button wird der aktuell selektierte gelöscht. Mit dem `Default`-Button wird der aktuell selektierte Text als Default markiert. Mit diesem Text wird das `reason`-Feld bei der Rufsignalisierung vorbelegt. Nach dem Betätigen des `Reject`-Buttons terminiert `coco` beim Angerufenen; das ikonifizierte Hauptfenster sowie das für die Rufsignalisierung verschwinden. Beim Anrufer erscheint in der Statuszeile des

Dialers die Meldung „`Rejected`“ gefolgt von der Begründung, sofern angegeben.

5.4.3 Rufumleitung

Bei einer Umleitung des Rufes (forward), muß in dem Eingabefeld rechts neben dem `Forward`-Button die neue Adresse eingegeben werden. Über den mit `Ph. Book` beschrifteten Button kann das Phonebook (siehe Abbildung 5.4) aufgerufen werden. Nach Betätigen des `Forward`-Buttons wird dem Anrufer die neue Adresse mitgeteilt, beim Angerufenen terminiert `coco`. Diese Form der Rufumleitung wird als temporäre betrachtet, welche vom Anrufer bestätigt werden muß, sofern dies beim Anrufer nicht abgeschaltet ist (siehe 5.6). Mit der neuen Adresse wird die alte im Dialer-Fenster überschrieben. Nach der Bestätigung erfolgt sofort der Verbindungsaufbau zu der neuen Adresse.

Abb. 5.6: Begründungen

Neben dieser manuellen Rufumleitung ist eine automatische Rufumleitung mittels des Call Handling Files vorgesehen. Mit diesem Script oder Programm kann der Benutzer für Zeiten seiner Abwesenheit oder aus anderen Gründen permanente Rufumleitungen einstellen. In diesem Fall würde also bei einem Anruf der Conference Controller gar nicht erst gestartet werden.

5.4.4 Rufannahme

Für alle vom Anrufer geforderten Medien, zu denen passende Agenten-Definitionen gefunden wurden, werden nach dem Betätigen des `Accept`-Buttons anhand der in der Definition abgelegten Kommandozeile die Media-Agents gestartet. Das Signalisierungsfenster verschwindet und das Hauptfenster des Conference-Controllers, welches bis dahin nur ikonifiziert existierte, erscheint. Mit der beim Anrufer eintreffenden Annahmebestätigung werden auch bei ihm die gewählten Media Agents gestartet. Möglicherweise kann dieses verzögerte Starten der Agenten beim Anrufer zu einem Problem führen: Mit der Rufannahme durch den Angerufenen werden bei diesem die Agenten *zuerst* gestartet. In der Teilnehmerliste des Hauptfensters des Conference Controllers ist somit der Anrufer noch nicht sichtbar, was u.U. zur Irritation führt.

5.5 Verwaltung der Media-Agents

Aufgabe des Media-Managers ist das Definieren von Eigenschaften lokal vorhandener Media-Agents (z.B. NeVoT von [8]). Zu diesen Eigenschaften zählen der Typ (Audio, Video, Whiteboard, Text), verarbeitbare Protokolle und Formate etc., sowie letztendlich die Kommandozeile, mit der eine Instanz dieses Agenten erzeugt wird. Über den `Media`-Button des Hauptfensters erreicht man den sog. Media-Manager, dessen Aussehen Abbildung 5.7 zeigt. In der oberen Liste werden die lokal verfügbaren Agenten mit ihren Eigenschaften bzw. Fähigkeiten (capabilities) definiert. Diese Definitionen werden beim Eintreffen von Anrufen benutzt, um für die vom Anrufer geforderten Medien und Parameter einen passenden Agenten zu finden. In der unteren Liste werden lokale Agenten definiert, die für „abgehende Gespräche“ benutzt werden sollen. Da sich beide Definitionen stark ähneln, liegt der Gedanke nahe, sie in einer Liste zu vereinen. Aus folgenden Gründen habe ich davon abgesehen: Zum einen definiert die obere Liste für lokal vorhandene Media-Agents die Eigenschaften. Dabei wird i.a. ein Media-Agent mehrere Protokolle Formate und Sample-Raten beherrschen, welche hier aufgezählt werden. Diese Liste wird bei einem Anruf benutzt, um automatisch den passenden Agenten zu finden und zu starten. Zum anderen dient die untere Liste der Voreinstellung bestimmter Agenten, die der Benutzer lokal benutzen möchte, wenn er eine Konferenz aufbaut. Dabei kann durchaus mehrfach derselbe Agent mit unterschiedlichen Parametern eingetragen werden, z.B. einmal NeVoT mit „RTP PCM8 8kHz“ und einmal NeVoT mit „RTP PCM16 22kHz“.

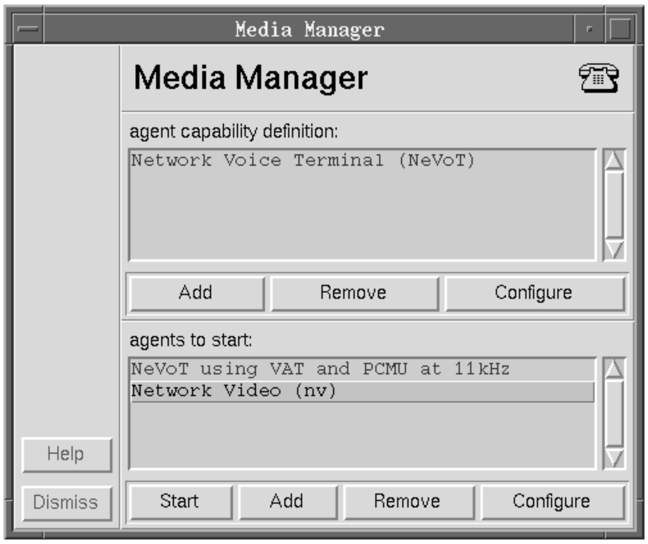

Abb. 5.7: Front-End des Media-Managers

5.5.1 Definition der Media-Agent-Capabilities

Wie oben erwähnt, dienen diese Definitionen dem Auffinden eines geeigneten Agenten beim Eintreffen eines Anrufes. Da der Anrufer derzeit nicht die Möglichkeit hat, alternative Medien-Parameter anzugeben, ist die richtige Definition dieser Parameter dafür ausschlaggebend, ob eingehende Anrufe entgegengenommen werden können oder nicht. Dementsprechend sollten hier alle Parameter, die der Agent beherrscht, gesetzt werden. In der gegenwärtigen Programmversion sind dies neben dem Medientyp, das Protokoll, das Übertragungsformat sowie die Samplingrate.

Eingestellt werden können alle Kombinationen dieser vier Parameter, auch wenn sie nicht sinnvoll sind, z.B. „`whiteboard VAT PCMU 44100`“. Dies könnte möglicherweise eine Einschränkung darstellen, wenn der Agent bestimmte Permutationen der eingestellten Parameter nicht zuläßt. Andererseits wäre es mühseelig, alle Permutationen aufzuzählen, die ein Agent „versteht", wenn man von min. vier Übertragungsraten und min sechs Formaten ausgeht, die ein Agent beherrscht.

Mit dem `Add`-Button wird eine neue Agentendefinition erzeugt, welche mittels des `Remove`-Buttons wieder gelöscht werden kann. Durch Doppelklicken auf einen in der Liste gezeigten Agenten oder durch Betätigen des `Configure`-Buttons kann die Agenten-Definition geändert werden. Es erscheint das in 5.8 gezeigte Fenster. Dort ist die Definition für den Audio-Agenten NEVOT zu sehen, der die Protokolle `vat` und `RTP` beherrscht. Die möglichen Übertragungsformate und Raten sind in den Eingabefeldern unterhalb der Scroll-Listen aufgezählt. Durch Doppelklicken auf ein Listenelement wird dieses ans Ende der Aufzählung im Eingabefeld angehangen. Die in dem Feld `description` eingegebene Bezeichnung wird lediglich innerhalb des Conference Controllers zur Benennung dieser Definition verwendet. Der `unique agent identifier` ist eine Kennung, die der Audio-Agent nach dem Starten über `pmm` aussendet. Hieran erkennt der Conference Controller den erfolgreichen Start des Agenten, ohne aktiv warten zu müssen.

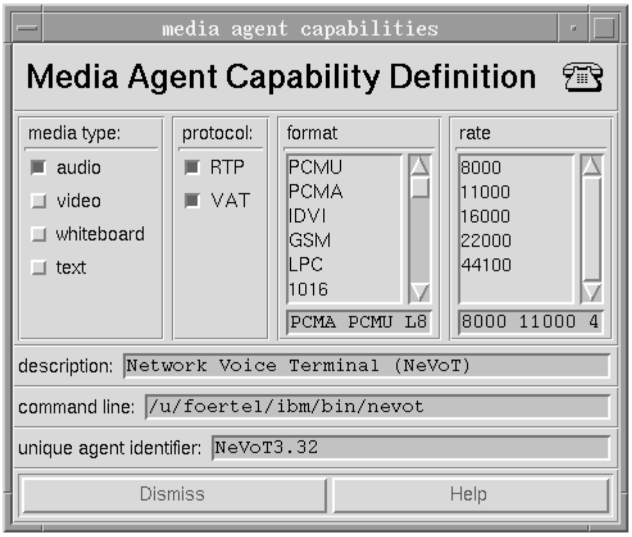

Abb. 5.8: Agent Capability Definition

5.5.2 Definition der Media-Agent Start-Parameter

In der zweiten der in Abb. 5.7 gezeigten Liste werden die bei abgehenden Anrufen zu verwendenden Media-Agents definiert. I.d.R. werden dies zwar die selben sein, die in der obigen Definition bereits aufgezählt wurden, jedoch wird hier für jede Definition genau eine Parameter-Kombination festgelegt, mit der der Agent benutzt werden soll. Unterhalb der zweiten in Abb. 5.7 gezeigten Liste befinden sich vier Buttons. Über den `Start`-Button kann der in der Liste aktivierte Agent gestartet werden, wenn seine Kommandozeile bereits eingetragen ist (siehe Abb. 5.9). Dies dient allerdings mehr Testzwecken, denn hierbei wird der gestartete Media-Agent in keine Konferenz eingebunden. Mit dem `Remove`-Button kann die Definition des in der Liste markierten Agenten gelöscht werden. Durch Doppelklicken eines in der Liste gezeigten Agenten oder über den `Configure`-Button kann die aktuelle Definition geändert werden. Der `Add`-Button fügt eine neue Definition hinzu. In beiden Fällen erscheint das in Abbildung 5.9 gezeigte Fenster. Hier können der Medientyp, das verwendete Protokoll, Format sowie die Sampling Rate eingestellt werden. Im unteren Bereich wird neben einer Kurzbezeichnung die Kommandozeile eingetragen, mit der eine Instanz des Agenten gestartet wird. Über den Checkbutton wird dieser Agent markiert und beim Aufbau einer Konferenz automatisch zum Starten vorgeschlagen (siehe 5.3).

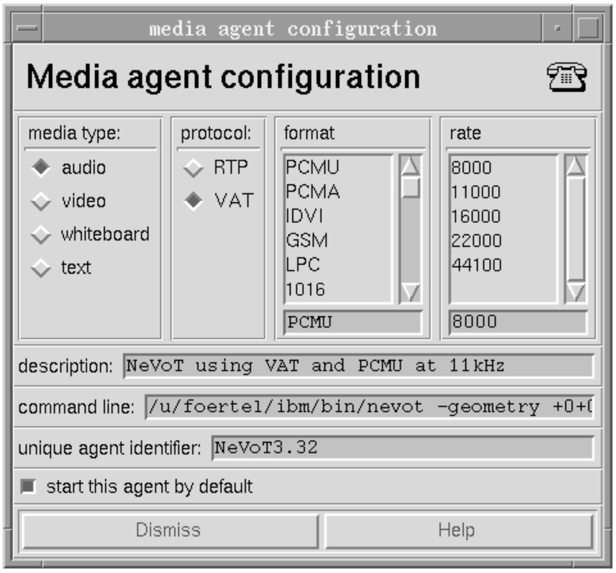

Abb. 5.9: Media-Agent Start-Parameter

5.6 Einstellungen

Z.Zt. sind alle Einstellungen für den Conference Controller in einem Fenster (siehe Abbildung 5.10) zusammengefaßt, wodurch dies recht groß und unübersichtlich geworden ist (zumindest auf einem 17”-Bildschirm). Auch folgt die Anordnung der Bedienungselemente eher ästhetischen denn logischen Gesichtspunkten. So wird beispielsweise die aktustische Rufsignalisierung oben links per Checkbutton eingeschaltet, das dazu abzuspielende Audio-File aber unten eingetragen. Aus Zeitgründen war eine Änderung leider nicht mehr möglich.

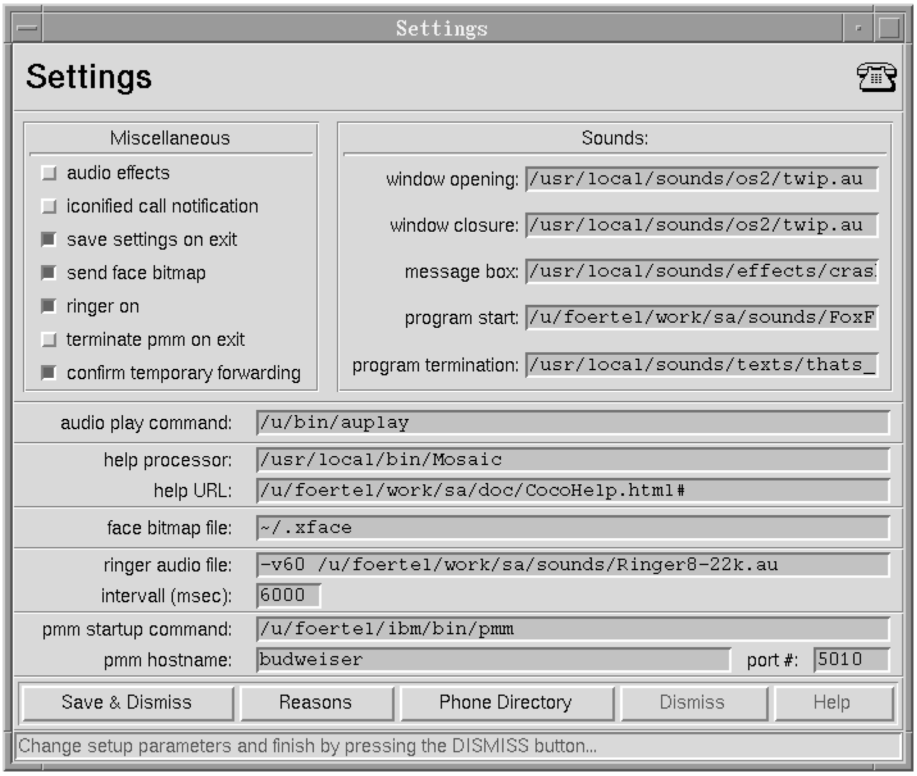

Abb. 5.10: Front-End für Programm-Parameter

Oben links sind diverse Bool'sche Parameter untergebracht, die mittels Checkbutton zu setzen sind. Die einzelnen Checkbuttons *gesetzt* bedeuten:

audio effects

Diverse Programmfunktionen können mit akustischen Effekten unterlegt werden, z.B. Öffnen oder Schließen eines Fensters. Wen das stört, der kann das hier abschalten.

iconified call notification

Bei einem Anruf wird das Fenster, in dem der Ruf angezeigt wird, nur als Icon gestartet. Damit dies nicht übersehen wird, sollte die akustische Signalisierung eingeschaltet sein.

save settings on exit

Bei Programmende sollen alle Einstellungen gespeichert werden. Im Setup-Fenster können mit dem `Save & Dismiss`-Button die Einstellungen sofort gespeichert werden.

send face bitmap

Beim Abschicken von Einladungen soll ein `.face` mitgeschickt werden. Der Dateiname ist unten angebbar.

ringer on

Anrufe sollen auch akustisch signalisiert werden. Das abzuspielende Sound-File kann unten angegeben werden.

terminate pmm on exit

pmm soll beim Verlassen des Conference Controllers beendet werden. Dies funktioniert z.Z. nur, wenn er auch von coco gestartet wurde, anderenfalls ist die Prozeß-Id, die für das Kill-Signal gebraucht wird, unbekannt.

confirm temporary forwarding

Temporäre Rufumleitungen (keine Redirects) sollen durch den Anrufer bestätigt werden. Bei einer Umleitung erscheint eine Dialogbox mit der neuen Adresse, die bestätigt werden muß.

Oben rechts werden unter der Überschrift *sounds* für verschiedene Programmaktionen die Sound-Files angegeben, die abgespielt werden, wenn links audio effects „angekreuzt" ist. In der unteren Hälfte werden diverse alphanumerische Parameter angegeben, deren Bedeutung folgende ist:

audio play command:

Das Kommando, welches zum Abspielen der diversen Audio-Files benutzt werden soll.

help processor:

Das Programm, welches aufgerufen werden soll, wenn der Benutzer in einem der coco-Fenster auf Help drückt.

help URL:

Dieser URL wird beim Aufruf des Hilfe-Programms zusammen mit einem Label übergeben. Der URL sollte daher mit einem # abgeschlossen werden. Wird z.B. der Help-Button im Hauptfenster gedrückt, so wird folgendes Kommando ausgeführt: <cmd line> <URL#>Main.

face bitmap file:

Das .face -File, das beim Einladen von Benutzern mitverschickt wird.

ringer audio file:

Sound-File, das bei der Rufsignalisierung, also beim „Klingeln" abgespielt werden soll.

intervall:

Das zuvor genannte Sound-File wird in Intervallen, der hier angegebenen Länge (in msec.) abgespielt.

pmm startup command:

Kommandozeile, mit der pmm gestartet wird, falls er beim Aufruf des Conference Controllers noch nicht läuft.

pmm hostname:

Name des Rechners, auf dem pmm laufen soll. Z.Z. aus Sicherheitsgründen sollte dies nur localhost sein können. Durch eine Änderung im pmm kann jedoch jeder Host angegeben werden. Wird ein anderer als localhost angegeben, so muß auch das Startkommando entsprechend verändert werden.

port #:

Port auf hostname, über den pmm erreichbar ist.

Alle Parameter werden bei Programmbeendigung in der Datei ~/.mucs-setup als tcl-Script gespeichert, wenn der Parameter save settings on exit gesetzt ist. Existiert diese Datei beim Aufruf von Conference Controllers nicht, so wird sie mit Standardeinstellungen erzeugt. Hier ein kurzer Ausschnitt der aus ~/.mucs-setup:

```
set setup(audioPlayer)            "/u/bin/auplay"
set setup(audioEffects)           0
set setup(audibleNotification)    1
set setup(iconifiedNotification) 1
set setup(ringerIntervall)        4000
set setup(sendFaceBitmap)         1
set setup(faceBitmap)             " /.xface"
set setup(reasons)                {busy}
set setup(helpProcessor)          {/usr/local/bin/Mosaic}
set setup(pmmPort)                5010
set setup(pmmCmd)                 {/u/foertel/ibm/bin/pmm}
set setup(pmmHost)                {localhost}
set setup(pmmKill)                0
set setup(saveSettings)           1
set setup(mailDeliveryAgent)      "/usr/lib/sendmail"
```

5.7 pmm

Wie bereits bei der Vorstellung der MUCS-Architektur erwähnt, sind alle Komponenten, mit denen der Benutzer lokal interagiert, über einen Software-Bus pmm miteinander verbunden. Dieser ist in [6] beschrieben. Zu den Komponenten, die in MUCS mittels pmm kommunizieren gehören z.Z. der Conference Controller coco,der Audio-Agent NEVOTsowie der Video-Agent vic. Der Software-Bus ist ein Prozeß, der nichts anderes macht, als Nachrichten über die mit ihm verbundenen Prozesse zu verteilen Das Nachrichtenformat ist in [7] beschrieben. Jeder Prozeß, der mit pmm verbunden ist, kann festlegen, welche Nachrichten er erhalten will, wofür einfache reguläre Ausdrücke verwendet werden.

Nach dem Starten des Conference Controllers versucht dieser zunächst eine Verbindung zu pmm aufzubauen. Gelingt ihm das, so wird das fokus-Logo in der rechten unteren Ecke des Hauptfensters türkisfarben dargestellt. Kam die Verbindung nicht zustande, so versucht coco den pmm mit der bei den Einstellungen abgelegten Kommandozeile zu starten. Dies kann auch auf einem anderen Rechner sein, was sich aus Sicherheitsgründen nicht empfiehlt. In meiner Testumgebung war jedoch die Maschine, in der die Audio-Hardware steckte (auf der also auch der Audio-Agent laufen mußte) so langsam, daß ich den Conference Controller auf einem anderen Rechner laufen lies. Konnte pmm nicht gestartet werden oder kam die Verbindung aus anderen Gründen nicht zustande, so wird das Fokus-Logo im Hauptfenster rot dargestellt, eine entsprechende Fehlermeldung erscheint in der Statuszeile. Erst wenn pmm läuft, besteht die Möglichkeit, Media-Agents zu starten.

In der bisherigen Version ließ pmm keine Verbindungen von anderen Rechnern als localhost zu. Die entsprechende Programmstelle habe ich so geändert, daß dies möglich ist.

5.7.1 Kommunikation mit NEVOT

An der Art der Kommunikation zwischen pmm und NEVOT hat sich nichts geändert. Lediglich beim Starten sendet NEVOT eine Agentid zu pmm an der coco wiederum erkennt, daß der Agent gestartet ist. Bisher war dies nicht möglich, da NEVOT nach dem Starten stumm blieb, solange keine Media-Session lief. Die Zeile, die NEVOT an pmm schickt lautet:

```
agent NeVoT3.32 attached
```

Beim Terminieren meldet sich NEVOT mit folgender Zeile wieder ab:

```
agent NeVoT3.32 detached
```

Die Id (NeVoT3.32), die bei diesen Zeilen angegeben ist, wird in der Media-Agent-Definition eingetragen (siehe 5.5). Nach dem Ausführen der ebenfalls dort abgelegten Kommandozeile, braucht coco nicht zu warten, bis der Agent wirklich gestartet ist (Auf meiner Entwicklungsmaschine dauerte dies bis zu 20 sec). Nach dem Erhalt der attached-Meldung beginnt coco mit dem Aufbau der Media-Session. Dies geschieht wie in [7] und [6] beschrieben, durch Schicken der folgenden Zeilen an NEVOT:

```
session <sessionname>
<sessionname> network UDP
<sessionname> cname <username>@<hostname>
<sessionname> transport <protocol>
<sessionname> media <mediadescr>
<sessionname> port <port>"
<sessionname> open <addr>"
<sessionname> send 1"
```

Hierbei setzt sich <sessionname> gemäß [7] aus der Prozeß-Id, dem Agententyp sowie einer Kanalnummer getrennt durch Schrägstriche zusammen, z.B. 11433/audio/1. Die restlichen Parameter entsprechen denen, die der Benutzer beim Konferenzaufbau angibt. Mit der letzten Zeile (send 1) wird das Muting für diesen Agenten abgeschaltet.

Kapitel 6

Ausblick/Bewertung

6.1 Was fehlt?

Aufgrund von Zeitnöten ist die Implementation des Conference Controllers nicht ganz so weit fortgeschritten, wie in Kapitel 2 gefordert. Die in Kapitel 5 beschriebene Implementation ist jedoch lauffähig und hat sich als durchaus brauchbar erwiesen. Dennoch gibt es folgende Einschränkungen:

- Die derzeit einzigen Media Agents, die in die lokale Architektur eingebunden werden können, sind NEVOT und `vic`. Die Einbeziehung anderer Audio Agenten sollte jedoch keinen großen Aufwand bedeuten. Hier ist die in [7] beschriebene `pmm`-Schnittstelle zu verwenden.
- Die Einbeziehung anderer Media Agents als Audio Agents, ist zwar theoretisch berücksichtigt worden, jedoch konnte sie nicht getestet werden. Einige Änderungen im Conference Controller für die Statusanzeige sind erforderlich.
- Auch ist unklar, ob die Darstellung der Konferenzteilnehmer im Hauptfenster in Form einer Listbox sinnvoll ist. Bei dieser Darstellung können je nach verwendeten Fonts und Anzahl der Teilnehmer bis zu 30 Teilnehmer sichtbar angezeigt werden. Maximiert man das Controller Fenster, so lassen sich ca. drei bis vier Spalten zu je 30 Teilnehmer darstellen.
- Bei einem Anruf sollte der Anrufer alternative Medien-Parameter angeben können, damit die Wahrscheinlichkeit des Zustandekommens einer Konferenz größer wird.

Andere offene Punkte fallen eher in die Kategorie kosmetische Änderungen, eine Änderung des MUCS-Protokolls ist hierfür nicht erforderlich:

- Die zwei Checkbuttons im Hauptfenster besitzen noch keine Funktionalität.
- Für die Rufablehnung besteht derzeit nur die Möglichkeit, *einen* Grund als Default zu markieren. Es wäre möglicherweise besser, drei oder mehr Gründe direkt im Signalisierungsfenster als Button zu haben. Das vereinfachte die Handhabung.
- Die Übertragung und Anzeige der `.face`-Dateien ist zwar vorbereitet und in der Bedienungsoberfläche angedeutet, jedoch noch nicht realisiert.

- Im Dialer stellt die auf 20 begrenzte Anzahl der Adreßfelder eine unnötige Begrenzung dar. Besser wäre eine „endlose“ Liste oder ein `Add`-Button, mit dem beliebig neue Felder erzeugt werden können.
- Das Setup-Window sollte in Pulldown-Menüs aufgeteilt werden, wobei logisch zusammengehörende Einstellungen in einem Fenster untergebracht werden sollten.
- Bei jeder Betätigung eines `Help`-Buttons wird eine Instanz des angegebenen Help-Prozessors gestartet. Die WWW-Browser `Mosaic` und `netscape` sehen jedoch eine Möglichkeit vor, bei der nur eine Instanz läuft. Sollen nach dem Erstaufruf weitere Dokumente angezeigt werden, so wird dies mittels einer Pipe o.ä. dem Programm mitgeteilt. Eine Unterstützung dieser Mechanismen sollte realisiert werden.

6.2 Weitere Entwicklung

Das entwickelte Multimedia Conferencing System (MUCS) unterstützt den Anwender lediglich beim *Aufbau* von Multimedia Konferenzen. Bei den Versuchen mit diesem System hat sich seine Brauchbarkeit durchaus erwiesen, wenn auch die Signalierungszeiten in unserer Testumgebung teilweise 7 sec betrugen. Das MUCS-Protokoll ist für künftige Erweiterungen völlig offen, wenn beispielsweise für die Signalisierung eines Anrufes neue Parameter benötigt werden. Dennoch: Für die Steuerung des weiteren Konferenzablaufs (z.B. Floor-Control) sowie für die Beendigung von Konferenzen ist keine Unterstützung vorgesehen. Neben funktionalen Erweiterungen des Conference Controllers betrachte ich diese beiden Bereiche als besonders interessant für künftige Erweiterungen bzw. Entwicklungen. Tabelle 6.1 stellt abschließend realisierte und nicht realisierte Merkmale gegenüber.

Merkmal	**realisiert?**
Anrufsignalisierung	ja
automatische und manuelle Rufumleitung	ja
Gruppenaddressierung	ja
Rufablehnung mit oder ohne Begründungen	ja
Integration anderer Media Agents als Audio Agents	vorbereitet
Zusammenfassung von Konferenzparametern zu Profilen	nein
Aushandlungsmechanismus für alternative Medienparameter	nein
Liste der Anrufer während Abwesenheit	nein
Grafisches Front-End für Rufumleitungsregeln etc.	nein
Proxy-Mode für Conference Daemon	nein
Unterstützung für Konferenzbeendigung	nein

Tabelle 6.1: MUCS-Merkmale

Anhang A

Programmstruktur

Ausgehend von einem gemeinsamen Verzeichnispfad befinden sich Programmcode und Dokumentation in eigenen Unterverzeichnissen:

`lib`	Enthält alle `tcl`-Sourcefiles
`doc`	Enthält diese Dokumentation
`bitmaps`	Enthält diverse X11-Bitmaps für die Anzeige in `coco`

A.1 Die `tcl`-Sourcefiles in `./lib`

Außer dem Hauptprogramm `./Coco.tcl` von `coco` das im Hauptverzeichnis steht, befinden sich alle Sourcefiles in dem Unterverzeichnis `./lib`. Von dort aus werden sie mitr dem dem Autoload-Feature von `tcl` aufgerufen.

`ConfSetup.tcl`	Conference setup window
`Dialer.tcl`	Dialer window
`Help.tcl`	Startup of help processor
`MediaMgr.tcl`	Media manager window
`Notify.tcl`	Call notification window
`Phonebook.tcl`	Phone directory window
`ProfMgr.tcl`	Profile manager (future extension)
`Setup.tcl`	Setup screen
`Terminate.tcl`	Programm termination functions
`Utils.tcl`	miscellaneous utility functions

A.2 Die Dokumenation in `./doc`

Neben dieser `LaTeX`-Dokumentation in `Coco.tex` gibt es ein noch nicht ganz fertiggestelltes HTML-File `CocoHelp.html`, daß eine Online-Dokumentation in englischer Sprache enthält. Dieses File wird als URL beim Aufruf des durch eine `Help`-Taste gestarteten WWW-Browsers übergeben.

Anhang B

Programm-Code

B.1 Das Hauptprogramm: `Coco.tcl`

```
#
# Coco.tcl
#

#
# First we "declare" some global stuff...
#

set auto_path [linsert $auto_path 0 /u/foertel/work/sa/lib]

set MUCS_SETUP_FILE  /.mucs-setup
set MUCS_VERSION "MUCS/1.0"

if [file readable $MUCS_SETUP_FILE] {
  source $MUCS_SETUP_FILE
} else {
  set setup(audioPlayer)              "/u/bin/auplay"
  set setup(audioEffects)             0
  set setup(audibleNotification)      1
  set setup(iconifiedNotification)    1
  set setup(ringerIntervall)          4000
  set setup(sendFaceBitmap)           1
  set setup(faceBitmap)               " /.xface"
  set setup(reasons)                  {busy}
  set setup(helpProcessor)            {/usr/local/bin/Mosaic}
  set setup(helpURL)                  {/u/foertel/work/sa/doc/CocoHelp.html#}
  set setup(pmmPort)                  5010
  set setup(pmmCmd)                   {/u/foertel/ibm/bin/pmm}
  set setup(pmmHost)                  {localhost}
  set setup(pmmKill)                  0
  set setup(saveSettings)             1
  set setup(mailDeliveryAgent)        "/usr/lib/sendmail"
  set setup(callFence)                5
  set setup(confirmFwds)              1

  set sounds(openWin)                 "/usr/local/sounds/os2/twip.au"
  set sounds(closeWin)                "/usr/local/sounds/os2/twip.au"
  set sounds(warning)                 "/usr/local/sounds/effects/crash.au"
  set sounds(welcome)                 "/u/foertel/work/sa/sounds/FoxFanfare-22k-s.au"
  set sounds(progExit)                "/usr/local/sounds/texts/thats_all_folks.au"
  set sounds(callNotify)              "/u/foertel/work/sa/sounds/Ringer8-22k.au"
  set sounds(notAvail)                "/u/foertel/work/sa/sounds/NotAvailable-11k.au"

  set setup(agent0id)                 0
  set setup(agent1id)                 0
  set setup(agent0ids)                "0"
  set setup(agent1ids)                "0"

  set agents0(0,name)                 "Network Voice Terminal (NeVoT)"
  set agents0(0,cname)                "NeVoT3.32"
  set agents0(0,types)                "audio"
  set agents0(0,protocols)            "RTP VAT"
  set agents0(0,fmts)                 "GSM PCMU PCMA L8 L16 LPC"
  set agents0(0,rates)                "8000 16000 44100"
  set agents0(0,cmd)                  "/u/foertel/ibm/bin/nevot"

  set agents1(0,name)                 "NeVoT using RTP and PCM8 at 11kHz"
  set agents1(0,cname)                "NeVoT3.32"
  set agents1(0,default)              1
  set agents1(0,type)                 audio
```

```
  set agents1(0,protocol)            RTP
  set agents1(0,fmt)                 L8
  set agents1(0,rate)                8000
  set agents1(0,cmd)                 "/u/foertel/ibm/bin/nevot"
}

set Resource(Display)                {-font -*-courier-medium-r-normal--14-*-*-*-*-*-*-* \
                                      -bg grey80 -fg navyblue}
set Resource(Display2)               {-font -*-courier-medium-r-normal--14-*-*-*-*-*-*-*}
set Resource(Entry)                  {-font -*-courier-medium-r-normal--14-*-*-*-*-*-*-* \
                                      -relief sunken -bg LightSteelBlue}
set Resource(Headline)               {-font -*-helvetica-bold-r-normal--24-*-*-*-*-*-*-*}
set Resource(Statusline)             {-font -*-helvetica-medium-r-normal--14-*-*-*-*-*-*-* \
                                      -fg firebrick -relief sunken -anchor w}
set Resource(ExitButton)             {-text "Dismiss" -fg firebrick -activeforeground firebrick}
set Resource(HelpButton)             {-text "Help" -fg forestgreen -activeforeground forestgreen}
#
set iconBitmaps(Main)                bitmaps/phone35x25.xbm
set iconBitmaps(Dialer)              bitmaps/phone35x25.xbm
set iconBitmaps(ConfSetup)           bitmaps/phone35x25.xbm
set iconBitmaps(MediaMgr)            bitmaps/phone35x25.xbm
set iconBitmaps(Notify)              bitmaps/phone35x25.xbm
set iconBitmaps(Phonebook)           bitmaps/phone35x25.xbm
set iconBitmaps(Setup)               bitmaps/phone35x25.xbm
#
set mediaFormats                     {PCMU PCMA IDVI GSM LPC 1016 G721 G723 L8 L16 MPA VDVI \
                                      TSP0 VSC JPEG MPV MP2T Ce1B H261 nv CPV HDCC CUSM PicW \
                                      RGB WB NT MMBL}
set mediaRates                       {8000 11000 16000 22000 44100}

#
# And now some "functions", which will be used exclusivly whithin this main file
#
###############################################################################
proc InfoHandler {mode fd} {
#
# Here we handle the pmm messages
#
        global taskList

        if ![eof $fd] {
          set line [dp_packetReceive $fd]
          if {$line == ""} {                         # Sometimes we get empty
            return;                                  # lines... Why?!?!
          }
          set line [string trim $line]
#puts "from pmm: $line"
#puts "taskList: $taskList"
#
# first we check "taskList", to see if there is a task defined for that message
#
          set taskId 0
          foreach task $taskList {
            if {[lindex $task 0] == $line} {
              set taskList [lreplace $taskList $taskId $taskId]
              eval [lindex $task 1]
            }
            incr taskId
          }
          if [string match "*/*/* hello *" $line] {
            set cname [lindex [split $line] 2]
            set sessName [lindex [split $line] 0]
            UpdateUserList $sessName $cname NEW ""
            return
          }
          if [string match "*/*/*/* bye" $line] {
            set sessName [lindex [split $line] 0]
            set line [lindex [split $line] 0]
            set cname [lindex [split $line "/"] 3]
            UpdateUserList $sessName $cname LEAVE ""
            return
          }
          if [string match "*/*/*/* active" $line] {
            set sessName [lindex [split $line] 0]
            set line [lindex [split $line] 0]
            set mediaType [lindex [split $line "/"] 1]
            set cname [lindex [split $line "/"] 3]
            UpdateUserList $sessName $cname ACTIVE $mediaType
            return
          }
          if [string match "*/*/*/* !active" $line] {
            set sessName [lindex [split $line] 0]
            set line [lindex [split $line] 0]
            set mediaType [lindex [split $line "/"] 1]
            set cname [lindex [split $line "/"] 3]
            UpdateUserList $sessName $cname INACTIVE $mediaType
            return
```

```
          }
          if [string match "*/*/* error *" $line] {
            set errMsg [lindex $line 2]
            set sessName [lindex $line 0]
            set mediaType [lindex [split $sessName "/"] 1]
            mkDialog .mediaErr error "-text \"Error from $mediaType media agent: \n $errMsg\" \
              -aspect 800" {Continue ""}
          }
        } else {
          .bottom.logo configure -fg tomato
          mkDialog .pmmabort error {-text "WARNING: pmm connection aborted!" -ascpect 800} \
            {Continue ""}
        }
}
################################################################################
proc Logo {w} {

        set back1 "-background grey80"
        set back2 "-background grey75"
        set back3 "-background grey70"
        set back4 "-background grey65"
        frame $w.f
        set fg1 "-foreground navyblue"
        eval frame $w.l1 $back4
          eval label $w.l1.ll -text "C" -font -*-helvetica-bold-r-normal--24-*-*-*-*-*-* \
             -width 1 $back1 $fg1
          pack $w.l1.ll -side left
        eval frame $w.l2 $back3
          eval label $w.l2.ll -text "O" -font -*-helvetica-bold-r-normal--24-*-*-*-*-*-* \
            -width 1 $back2 $fg1
          eval label $w.l2.lr -text "nference" $back3
          pack $w.l2.ll $w.l2.lr -side left
        eval frame $w.l3 $back2
          eval label $w.l3.ll -text "C" -font -*-helvetica-bold-r-normal--24-*-*-*-*-*-* \
            -width 1 $back3 $fg1
          pack $w.l3.ll -side left
        eval frame $w.l4 $back1
          eval label $w.l4.ll -text "O" -font -*-helvetica-bold-r-normal--24-*-*-*-*-*-* \
            -width 1 $back4 $fg1
          eval label $w.l4.lr -text "ntroller" $back1
          pack $w.l4.ll $w.l4.lr -side left
        pack $w.l1 $w.l2 $w.l3 $w.l4 -in $w.f -expand true -fill x
        pack $w.f -padx 2m -pady 2m
}
################################################################################
proc ToggleAudioMuting {cname sessName muteOn} {

        global pmmFd userList

        set frameWidget ""
        set id 0
        foreach userEntry $userList {
          if {[lindex $userEntry 0] == $cname} {
            set frameWidget [lindex $userEntry 1]
            set talks [lindex $userEntry 2]
            break
          }
          incr id
        }

        if {$frameWidget == ""} {
          return
        }

        if $muteOn {
          if $talks {
            $frameWidget.audio configure -bitmap @bitmaps/Talk0.xbm
          } else {
            $frameWidget.audio configure -bitmap @bitmaps/Silent1.xbm
          }
          bind $frameWidget.audio <Button-1> "ToggleAudioMuting $cname $sessName 0"
          dp_packetSend $pmmFd "$sessName send 0"
          set muted 1
        } else {
          if $talks {
            $frameWidget.audio configure -bitmap @bitmaps/Talk1.xbm
          } else {
            $frameWidget.audio configure -bitmap @bitmaps/Silent0.xbm
          }
          bind $frameWidget.audio <Button-1> "ToggleAudioMuting $cname $sessName 1"
          dp_packetSend $pmmFd "$sessName send 1"
          set muted 0
        }
        set userList [lreplace $userList $id $id "$cname $frameWidget $talks $muted"]
}
################################################################################
proc UpdateUserList {sessName cname what mediaType} {
```

```
#
# Here we update the main windows list of conference participants.
#
        global Resource userId userList userListWidget

        #
        # Check whether cname is being already displayed
        #
        set frameWidget ""
        set id 0
        foreach userEntry $userList {
          if {[lindex $userEntry 0] == $cname} {
            set frameWidget [lindex $userEntry 1]
            set talks [lindex $userEntry 2]
            set muted [lindex $userEntry 3]
            break
          }
          incr id
        }

        if {$frameWidget == ""} {
          #
          # User not in list
          #
          incr userId
          set f $userListWidget.f$userId
          lappend userList "$cname $f 0 0"
          frame $f
            pack \
              [eval label $f.audio -bitmap @bitmaps/Unavailable.xbm $Resource(Display) \
                -cursor \{hand2 red\}] \
              [eval label $f.l1 -wi 20 -text \"$cname\" $Resource(Display) -anchor w] \
              -side left -padx 1 -pady 1 -anchor w
          BindWidgetExpl $f.audio .bottom.status "Click here to toggle audio muting"
#         BindWidgetExpl $f.video .bottom.status "Click here to toggle video muting"
#         BindWidgetExpl $f.whiteboard .bottom.status "Click here to disable whiteboard"
          bind $f.audio <Button-1> "ToggleAudioMuting $cname $sessName 1"
          $userListWidget window create end -window $f
          return
        }

        #
        # Already in list
        #
        if {$what == "LEAVE"} {
          destroy $frameWidget
          set userList [lreplace $userList $id $id]
          return
        }
        set bitmapFile ""
        if {$what == "ACTIVE"} {
          set talks 1
          if {$muted} {
            set bitmapFile @bitmaps/Talk0.xbm
          } else {
            set bitmapFile @bitmaps/Talk1.xbm
          }
        }
        if {$what == "INACTIVE"} {
          set talks 0
          if {$muted} {
            set bitmapFile @bitmaps/Silent1.xbm
          } else {
            set bitmapFile @bitmaps/Silent0.xbm
          }
        }
        set userList [lreplace $userList $id $id "$cname $frameWidget $talks $muted"]
        if {$bitmapFile != ""} {
          if {$mediaType == "audio" || $mediaType == "video" || $mediaType == "whiteboard"} {
            $frameWidget.$mediaType configure -bitmap $bitmapFile
          }
        }
}
################################################################################
proc Usage {} {
        puts {usage: coco [-p <port>] [-h <host>]}
}
################################################################################

#
# *** main ***
#

#
# Parse command line arguments
#
for {set i 0} {$i < $argc} {incr i} {
```

```
  switch -- [lindex $argv $i] {
    -h {incr i
        set callHost [lindex $argv $i]
puts $callHost
       }
    -p {incr i
        set callPort [lindex $argv $i]
puts $callPort
       }
    default {Usage; exit}
  }
}

puts "CoCo: stand by, please..."
AudioEffect welcome


#
# initialize global variables
#
set reachFirst 0
set userId 0
set userList {}
set taskList {}

#
# create front-end
#
wm title . "Conference Controller"
wm iconbitmap . @$iconBitmaps(Main)

if [info exists callPort] {
  wm iconify .
}

frame .top
  frame .l -bd 1 -relief raised
    frame .l.f1 -relief ridge -bd 2

Logo .l.f1

    pack .l.f1 -padx 2m -pady 2m

  frame .l.f2 -bd 1 -relief sunken
    pack \
      [eval button .l.f2.exit  $Resource(ExitButton) -text \"Exit\" -command \"Terminate\"] \
      [eval button .l.f2.help  $Resource(HelpButton) -command \"Help Main .bottom.status\"] \
      [button .l.f2.media -text "Media" -command "MediaManager .media"] \
      [button .l.f2.setup -text "Setup" -command "SetUp .setup"] \
      [button .l.f2.new -text "New" ]\
      [button .l.f2.start -text "Call" -command {ConferenceSetup .confmgr} ]\
      -side bottom -exp true -fill both -padx 0 -pady 0
    pack .l.f2 -exp true -fill both -padx 2m -pady 2m
  pack .l -in .top -side left -fill both


  frame .r
    frame .r.f2 -bd 1 -relief raised
      frame .r.f2.f1
        pack [label .r.f2.f1.l1 -text "conference participants:" -anchor w] \
          -side top -anchor w -fill x
        pack [text .r.f2.f1.t -state disabled -yscroll ".r.f2.f1.s set" -height 10 -width 40] \
          -side left -exp true -fill both
        pack [scrollbar .r.f2.f1.s -command {.r.f2.f1.t yview}] -side left -fill y
      pack .r.f2.f1 -padx 2m -pady 1m -side left -expand true -fill both
    pack .r.f2 -side top -exp true -fill both
    frame .r.f3 -bd 1 -relief raised
      pack \
        [checkbutton .r.f3.c1 -text "display self always at top"] \
        [checkbutton .r.f3.c2 -text "make active members visible"] \
        -side left -fill x
    pack .r.f3 -side top -fill x
  pack .r -in .top -side left -expand true -fill both
set userListWidget .r.f2.f1.t

  frame .bottom -bd 1 -relief raised
    eval label .bottom.status -text \"this is the main windows message line...\" \
      $Resource(Statusline)
    pack .bottom.status -padx 2m -pady 1m -fill x -expand true -side left -ipady 1m
    pack [label .bottom.logo -bitmap @bitmaps/fokus_logo.xbm -foreground #2189bb22a51e] \
      -padx 2m -side right

pack .top -expand true -fill both
pack .bottom -side top -fill x

#
# pointer bindings
```

```
#
BindWidgetExpl .l.f2.setup .bottom.status \
  "Opens the setup screen"
BindWidgetExpl .l.f2.exit  .bottom.status \
  "Terminates this program without any further questions"
BindWidgetExpl .l.f2.media .bottom.status \
  "Opens the media manager window"
BindWidgetExpl .l.f2.help  .bottom.status \
  "Starts the help processor"
BindWidgetExpl .l.f2.start .bottom.status \
  "Opens a window where you can set-up a new conference"
BindWidgetExpl .l.f2.new   .bottom.status \
  "Starts a new instance of CoCo"
BindWidgetExpl .r.f3.c1    .bottom.status \
  "Shows yourself always at the top of the user list"
BindWidgetExpl .r.f3.c2    .bottom.status \
  "Active paticipants are made visible by moved them up"

#
# global bindings
#
if {$tk_version < "4.0"} {
  bind Entry @          {%W insert insert @}
  bind Entry <Left>     {%W icursor [expr [%W index insert] - 1]}
  bind Entry <Right>    {%W icursor [expr [%W index insert] + 1]}
  bind Entry <Home>     {%W icursor 0; %W view 0}
  bind Entry <End>      {%W icursor end; %W view [expr [%W index end] - 10]}
}

#
# try to connect to pmm
#
set pmmId 0
if {$setup(pmmPort) != "" && $setup(pmmHost) != ""} {
  set cc [catch {set ret [dp_connect $setup(pmmHost) $setup(pmmPort)]} msg]
  if {$cc} {
    #
    # couldn't connect to pmm, try to start him/it
    #
    if {$setup(pmmCmd) != ""} {
      puts "CoCo: pmm is not running, trying to start it..."
      if ![catch {set pmmId [eval exec $setup(pmmCmd) &]} msg] {
        after 2000
        set cc [catch {set ret [dp_connect $setup(pmmHost) $setup(pmmPort)]} msg]
      }
    }
  }
  if $cc {
    .bottom.logo configure -fg tomato
    .bottom.status configure -text "$msg"
  } else {
    #
    # we could connect, so we register now
    #
    set pmmFd [lindex $ret 0]
    dp_filehandler $pmmFd r InfoHandler
    dp_packetSend $pmmFd "+ *"
#    dp_packetSend $pmmFd "+ */*/* hello *"
#    dp_packetSend $pmmFd "+ */*/*/* bye"
#    dp_packetSend $pmmFd "+ */*/*/* *active"
    .bottom.status configure -text "Successfully connected to pmm..."
  }
} else {
  #
  # due to missing host names and/or port number we couldn't connect to pmm
  #
  .bottom.logo configure -fg tomato
  .bottom.status configure -text "No connection to pmm!"
  update
  mkDialog .nopmm warning \
    {-text "Cannot connect to pmm, setup parameters are missing or incomplete" \
    -aspect 800} {Continue ""}
}

#
# Check for incoming call
#
if [info exists callPort] {
  if [info exists callHost] {
    set ret [dp_connect $callHost $callPort]
  } else {
    set ret [dp_connect localhost $callPort]
  }
  set fd [lindex $ret 0]
  dp_filehandler $fd r CallHandler
  set callList($fd) ""
}
```

```
#### END OF FILE ####
```

B.2 Starten einer neuen Konferenz: ConfSetup.tcl

```
#
# lib/ConfSetup.tcl
#
set MAX_AGENTS 20
#
################################################################################
proc ConferenceSetup {w} {

        global activatedAgents agents1 confName confPrio iconBitmaps \
               MAX_AGENTS mediaAddr mediaPorts Resource setup unicastConference

        set unicastConference 1

        AudioEffect openWin
        toplevel $w
        wm title $w "Conference Setup"
        wm iconbitmap $w @$iconBitmaps(ConfSetup)

        set confPrio normal
        set confName [pid]

        frame $w.f1 -bd 1 -relief raised
          pack [eval label $w.f1.l1 -text \"Conference Setup\" $Resource(Headline) \
                -anchor w -padx 2m -pady 3m] -side left -anchor nw
          pack [eval label $w.f1.l2 -bitmap @bitmaps/phone35x25.xbm \
                -anchor e -padx 2m -pady 3m] -side right -padx 2m -pady 2m
        frame $w.f2 -bd 1
          frame $w.f2.f2 -relief raised -bd 1
            pack \
              [label $w.f2.f2.l1 -text "Subject:" -wi 30 -anchor e] \
              [eval entry $w.f2.f2.e1 -wi 25 -textvar confName $Resource(Entry)] \
              -side left
          frame $w.f2.f3 -relief raised -bd 1
            pack \
              [label $w.f2.f3.l1 -text "Priority:" -wi 30 -anchor e] \
              [radiobutton $w.f2.f3.r1 -text "emergency" -var confPrio -value "emergency"] \
              [radiobutton $w.f2.f3.r2 -text "urgent" -var confPrio -value "urgent"] \
              [radiobutton $w.f2.f3.r3 -text "normal" -var confPrio -value "normal"] \
              -side left
          frame $w.f2.f6 -relief raised -bd 1
            pack \
              [label $w.f2.f6.l1 -text "Send X-face bitmap?" -wi 30 -anchor e] \
              [checkbutton $w.f2.f6.c1] \
              [eval entry $w.f2.f6.e1 $Resource(Entry) -textvar setup(faceBitmap)] \
              -side left
          frame $w.f2.f7 -relief raised -bd 1
            pack \
              [label $w.f2.f7.l1 -text "Unicast conference?" -wi 30 -anchor e] \
              [checkbutton $w.f2.f7.c1 -var unicastConference] \
              -side left
          frame $w.f2.f4 -relief raised -bd 1
            pack [label $w.f2.f4.l -text "select media agents:" -anchor w] \
              -side left

          frame $w.f2.f5 -relief raised -bd 1
            pack [text $w.f2.f5.t -state disabled \
                  -yscroll "$w.f2.f5.s set" -height 6 -wi 74] \
              -side left -exp true -fill both
            pack [scrollbar $w.f2.f5.s -command "$w.f2.f5.t yview"] \
              -side left -fill y

        frame $w.f
          pack \
            [eval label $w.f.l2 -anchor w -text \"specification\" -wi 38 $Resource(Display2)] \
            [eval label $w.f.l3 -anchor w -text \"multicastaddr.\" -wi 16 $Resource(Display2)] \
            [eval label $w.f.l4 -anchor w -text \"port\" -wi 6 $Resource(Display2)] \
            -side left -exp true -fill x -anchor w -padx 1m
        $w.f2.f5.t window create end -window $w.f -stretch true -pady 1

#
# fill scrolled text widget with media agent descriptions
#
        set i 0
        set port 3455
        foreach id $setup(agent1ids) {
          incr port
          if $agents1($id,default) {
            set mediaPorts($i) $port
            set activatedAgents($i) 1
```

```
          } else {
            set activatedAgents($i) 0
          }
          frame $w.f2.f5.t.ff$i
            pack \
              [eval checkbutton $w.f2.f5.t.ff$i.l2 -anchor w \
               -variable activatedAgents($i) -text \"$agents1($id,name)\" \
               -wi 35 $Resource(Display)] \
              [eval entry $w.f2.f5.t.ff$i.e1 -wi 16 -textvar mediaAddr($i) $Resource(Entry)] \
              [eval entry $w.f2.f5.t.ff$i.e2 -wi 6 -textvar mediaPorts($i) $Resource(Entry)] \
            -side left -anchor w -exp true -fill x -padx 1m
          bind $w.f2.f5.t.ff$i.e1 <Return> "InvokeDialer $w"
          bind $w.f2.f5.t.ff$i.e2 <Return> "InvokeDialer $w"
          $w.f2.f5.t window create end -window $w.f2.f5.t.ff$i -stretch true -pady 1
          incr i
        }

          pack $w.f2.f2 $w.f2.f3 $w.f2.f6 $w.f2.f7 $w.f2.f4 $w.f2.f5 \
            -side top -exp true -fill both

        frame $w.f3 -bd 1 -relief raised
          frame $w.f3.f1 -bd 1 -relief sunken
            pack \
              [button $w.f3.f1.b2 -text "Invite Users" -command "InvokeDialer $w"] \
              [eval button $w.f3.f1.b1 $Resource(ExitButton) \
                -com \"LeaveConferenceSetup $w\"] \
              [eval button $w.f3.f1.b3 $Resource(HelpButton) \
                -com \"Help ConfSetup $w.f4.status\"] \
              -side left -exp true -fill x
            pack $w.f3.f1 -exp true -fill x -padx 1m -pady 1m

        frame $w.f4 -bd 1 -relief raised
          pack [eval label $w.f4.status $Resource(Statusline) \
            -text \"Enter the conference parameters in the entries above and press START...\"] \
            -side left -exp true -fill x -padx 1m -pady 1m

        pack $w.f1 $w.f2 $w.f3 $w.f4 -side top -exp true -fill both
#
# perform some useful bindings
#
        BindWidgetExpl $w.f2.f3.r1 $w.f4.status \
          "Select the desired call priority by setting the appropriate radiobutton"
        BindWidgetExpl $w.f2.f3.r2 $w.f4.status \
          "Select the desired call priority by setting the appropriate radiobutton"
        BindWidgetExpl $w.f2.f3.r3 $w.f4.status \
          "Select the desired call priority by setting the appropriate radiobutton"
        BindWidgetExpl $w.f2.f2.e1 $w.f4.status \
          "Enter a descriptive conference name or subject"
        BindWidgetExpl $w.f2.f5.t  $w.f4.status \
          "Select the desired media agents by setting the checkbuttons on the left"
        BindWidgetExpl $w.f3.f1.b1 $w.f4.status \
          "Closes this window without starting a conference"
        BindWidgetExpl $w.f3.f1.b2 $w.f4.status \
          "Opens the dialer window from where you invite the conference participants"
        BindWidgetExpl $w.f3.f1.b3 $w.f4.status \
          "Starts the help processor which is currently under construction"
#
        focus $w.f2.f2.e1
}
###############################################################################
proc CreateMediaSession {id addr port} {

        global agents1 pmmFd taskList

        set sessName "[pid]/$agents1($id,type)/1"
#puts "CREATING MEDIA SESSION $sessName"
        dp_packetSend $pmmFd "session $sessName"

        lappend taskList "\"$sessName created\" \
          \"SetupMediaSession $sessName $id $addr $port\""
}
###############################################################################
proc InvokeDialer {w} {

        global activatedAgents mediaAddr mediaPorts

#
# now we check, if any media agent was selected. At leat one is required
#
        set agentCount 0
        foreach i [array names activatedAgents] {
          if {$activatedAgents($i) == 1} {
            if {$mediaPorts($i) == ""} {
              mkDialog .err warning {-text "Media address/port are required for each checked agent!" \
                -aspect 700} {Continue ""}
              return
            }
```

```
            incr agentCount
          }
        }
        if {$agentCount == 0} {
          mkDialog .err warning {-text "Specify at least one media agent, please!" \
            -aspect 800} {Continue ""}
          return
        }
#       StartMediaAgents $w

        Dialer $w.dialer
}
################################################################################
proc LeaveConferenceSetup {w} {

        AudioEffect closeWin
        destroy $w
}
################################################################################
proc SetupMediaSession {sessName id addr port} {

        global agents1 pmmFd

puts "SETUP MEDIA SESSION $sessName $addr $port"
        dp_packetSend $pmmFd "$sessName network UDP"
#       dp_packetSend $pmmFd "$sessName priority duplex"
#       dp_packetSend $pmmFd "$sessName mute_new false"
#       dp_packetSend $pmmFd "$sessName delay_smoth 3"
#       dp_packetSend $pmmFd "$sessName min_delay 0.0"
#       dp_packetSend $pmmFd "$sessName max_delay 5.0"
#       dp_packetSend $pmmFd "$sessName initial_delay 0.1"
#       dp_packetSend $pmmFd "$sessName delay_multiplier 4"
#       dp_packetSend $pmmFd "$sessName late_ts 1"
#       dp_packetSend $pmmFd "$sessName immediate 0"
        dp_packetSend $pmmFd "$sessName cname [idstring "%u@%d"]"
        dp_packetSend $pmmFd "$sessName transport $agents1($id,protocol)"
        dp_packetSend $pmmFd "$sessName media $agents1($id,fmt) 1 $agents1($id,rate) 0.02"
        dp_packetSend $pmmFd "$sessName port $port"
        dp_packetSend $pmmFd "$sessName open $addr"
        dp_packetSend $pmmFd "$sessName send 1"
}
################################################################################
proc StartMediaAgents {unicastAddr} {

        global agents1 activatedAgents MAX_AGENTS pmmFd mediaAddr mediaPorts \
          setup taskList unicastConference

        foreach i [array names activatedAgents] {
          if {$activatedAgents($i) == 1} {
            set id [lindex $setup(agent1ids) $i]
            if {$agents1($id,cmd) == ""} {
              mkDialog .mediaErr error "-text \"Cannot start $agents1($id,name), no \
                command line specified!\" -aspect 800" {Continue ""}
              continue
            }
#           $statusline configure -text "Starting $agents1($id,name)..."
            update
            eval exec $agents1($id,cmd) &
            if $unicastConference {
              set addr $unicastAddr
            } else {
              set addr $mediaAddr($i)
            }
            lappend taskList "\"agent $agents1($id,cname) attached\" \
                \"CreateMediaSession $id $addr $mediaPorts($i)\""
          }
        }
#       after 4000 $statusline configure -text ""
        update
}
#### END OF FILE ####
```

B.3 Einladen der Benutzer: Dialer.tcl

```
#
# lib/Dialer.tcl
#
set MAX_ENTRIES 20
#
#*******************************************************************************
proc AbortCall {w} {
#
# This procedure is invoked when the user presses the DISMISS/ABORT CALL
# button in the dialer window
```

```
#
        global replyList

        AudioEffect closeWin

        if [winfo exists .confmgr] {
          destroy .confmgr
        } else {
          destroy $w
        }

        #
        # shutdown all sockets and remove filehandlers - ignore errors
        #
        foreach fd [array names replyList] {
          catch {dp_shutdown $fd both}
          catch {dp_filehandler $fd}
          catch {close $fd}
        }
}
###############################################################################
proc CallSubscribers {w} {
#
# Here we loop through all entries in the dialer window and try to call
# to each "number". For unicast conferences we allow to contact only one
# participant.
#
        global conFailure hostList MAX_ENTRIES unicastConference

        set hostList {}
        set success 0
        set conFailure 0
        for {set i 1} {$i <= $MAX_ENTRIES} {incr i} {
          set addr [$w.line1.t.f$i.e$i  get]
          if {$addr == ""} {
            continue
          }
          if {[CallSingleSubscriber $w $i] == -1} {
            set conFailure 1
          } else {
            set success 1
            if {$unicastConference} {
              break
            }
          }
        }
        if !$success {
          mkDialog .error warning {-text "Specify at least one valid destination, please!" \
            -aspect 800} {Continue ""}
        }
}
###############################################################################
proc CallSingleSubscriber {w i} {
#
# Here we try to contact the address specified in entry #i of the dialer window.
# If we succeed, we return the address of the contacted host else -1.
#
        global agents1 activatedAgents conFailure confName confPrio hostList \
                mediaAddr mediaPorts MUCS_VERSION replyList setup unicastConference

        #
        # analyze entries contents
        #
        set addr [$w.line1.t.f$i.e$i get]
        set noOfComponents [llength [split $addr @]]
        set userName [lindex [split $addr @] 0]
        if {$noOfComponents > 1} {
          set domainPart [lindex [split $addr @] 1]
        } else {
          $w.line1.t.f$i.e$i insert end @[idstring "%d"]
          set domainPart [idstring "%d"]
        }
        #
        # resolve host name
        #
        set addrList [DomainExpand $domainPart]
        if {$addrList ==  ""} {
          $w.line1.t.f$i.l2$i configure -text "Cannot resolve host/domainname!"
          set conFailure 1
          return -1
        }
        #
        # With the list of host addresses returned from the domain name
        # resolution, we try to contact the first avialable host.
        #
        foreach host $addrList {
          set hostAddr [lindex $host 1]
```

```
          #
          # check whether address exists only once in list
          #
          if {[lsearch $hostList "$userName@$hostAddr"] != -1} {
            $w.line1.t.f$i.l2$i configure -text "Multiple address specification!"
            return -1
          } else {
            lappend hostList "$userName@$hostAddr"
          }
          $w.line1.t.f$i.l2$i configure -text "Trying to connect to $hostAddr..."
          update
          if ![catch {set fd [lindex [dp_connect $hostAddr mucs] 0]} retCode] {
            $w.line1.t.f$i.l2$i configure -text "connected, calling for $userName"
            update
            #
            # Setup the MUCS protocol call package:
            #
            dp_send $fd "Call $userName $MUCS_VERSION"
            if {$confName != ""} {
              dp_send $fd "Info: $confName"
            }
            dp_send $fd "Origin: [idstring "%u@%d"]"
            foreach j [array names activatedAgents] {
              if {$activatedAgents($j) == 1} {
                set id [lindex $setup(agent1ids) $j]
                set type $agents1($id,type)
                set prot $agents1($id,protocol)
                set fmt $agents1($id,fmt)
                set rate $agents1($id,rate)
                if {$unicastConference} {
                  set addr [idstring "%i"]
                } else {
                  set addr $mediaAddr($j)
                }
                set port $mediaPorts($j)
                dp_send $fd "Media: $type $prot $fmt $rate $addr $port"
              }
            }
            dp_send $fd "Priority: $confPrio"
            dp_send $fd ""
            dp_filehandler $fd r ReplyHandler
            set labelName $w.line1.t.f$i.l2$i
            #
            # Store some usefull infos as the first three elements
            # of "replyList". We need them later...
            #
            set replyList($fd) [list $labelName $w $i $hostAddr]
            $w.line1.t.f$i.l2$i configure -text "found, waiting for response..."
            update
            return $hostAddr
          }
        }
        $w.line1.t.f$i.l2$i configure -text "Cannot connect to host!"
        return -1
}
############################################################################
proc ClearEntries {baseName} {
#
# This proc. is invoked when the user presses the CLEAR button in the dialer window
#
        global MAX_ENTRIES reachFirst

        for {set i 1} {$i <= $MAX_ENTRIES} {incr i} {
          $baseName.f$i.e$i delete 0 end
          $baseName.f$i.l2$i configure -text ""
        }
}
############################################################################
proc Dialer {w} {

        global iconBitmaps MAX_ENTRIES reachFist Resource tk_version

        if [winfo exists $w] {
          wm deiconify $w
          raise $w
          return
        }

        AudioEffect openWin
        toplevel $w

        wm title $w "Dialer"
        wm iconbitmap $w @$iconBitmaps(Dialer)

        frame $w.headline -bd 1 -relief raised
          pack [eval label $w.headline.l -text "Dialer" $Resource(Headline) -anchor w \
            -padx 2m -pady 3m] \
```

```
            -side left -anchor nw
          pack [eval label $w.headline.b -bitmap @bitmaps/phone35x25.xbm -anchor e \
            -padx 2m -pady 3m] \
            -side right -padx 2m -pady 2m
          pack $w.headline -fill x

        #
        # make scrolled text with MAX_ENTRIES entries in it.
        #
        frame $w.line1 -bd 1 -relief raised
          text $w.line1.t -state disabled -yscroll "$w.line1.s set" \
            -setgrid true -height 10 -width 71
          scrollbar $w.line1.s -command "$w.line1.t yview"
          pack $w.line1.t -side left -exp true -fill both
          pack $w.line1.s -side right -fill y
        for {set i 1} {$i <= $MAX_ENTRIES} {incr i} {
          frame $w.line1.t.f$i
            eval entry $w.line1.t.f$i.e$i -width 30 $Resource(Entry)
            label $w.line1.t.f$i.l2$i -anchor w -width 30 -fg firebrick -bg grey80
            pack $w.line1.t.f$i.l2$i $w.line1.t.f$i.e$i \
              -side right -anchor e -padx 1m -pady 1m -fill x -exp true
          $w.line1.t window create end -window  $w.line1.t.f$i -padx 3 -pady 1
        }
        pack $w.line1 -exp true -fill both

        frame $w.line3 -bd 1 -relief raised
          pack \
            [radiobutton $w.line3.c1 -var reachFirst -val 1 -text "reach first"] \
            [radiobutton $w.line3.c2 -var reachFirst -val 0 -text "reach all"] \
            -side left -padx 2m -pady 2m
          pack $w.line3 -fill x

        frame $w.line2 -bd 1 -relief raised
          frame $w.line2.f1 -bd 1 -relief sunken
            pack \
              [button $w.line2.f1.b2 -text "Call" \
                -command "CallSubscribers $w"] \
              [button $w.line2.f1.b3 -text "Phone Directory" \
                -command "Phonebook $w.pb $w.line2.f1.b3 $w.line1.t MULTIPLE"] \
              [button $w.line2.f1.b4 -text "Clear" \
                -command "ClearEntries $w.line1.t"] \
              [eval button $w.line2.f1.b1 $Resource(ExitButton) \
                -text \"Dismiss / Abort Call\" \
                -command \"AbortCall $w\"] \
              [eval button $w.line2.f1.b5 $Resource(HelpButton) \
                -command \"Help Dialer\"] \
            -side left -exp true -fill x
            pack $w.line2.f1 -fill x -padx 1m -pady 1m
          pack $w.line2 -fill x

        frame $w.status -bd 1 -relief raised
          eval label $w.status.l -text \"Enter the address of the user(s) you which to call to\" \
            -anchor w $Resource(Statusline)
          pack $w.status.l -padx 1m -pady 1m -fill x
          pack $w.status -fill x

        #
        # pointer bindings
        #
        BindWidgetExpl $w.line2.f1.b2 $w.status.l "Calls user(s) with the address specified above"
        BindWidgetExpl $w.line2.f1.b1 $w.status.l "Closes this window and aborts all pending calls"
        BindWidgetExpl $w.line2.f1.b3 $w.status.l "Opens the phone directory window"
        BindWidgetExpl $w.line2.f1.b4 $w.status.l "Clears all of the address entries above"
        #
        # place focus initially into first e-mail entry
        #
        focus $w.line1.t.f1.e1
}
################################################################################
proc ForwardCall {w i location} {

        $w.line1.t.f$i.e$i delete 0 end
        $w.line1.t.f$i.e$i insert end $location
        $w.line1.t.f$i.e$i xview end

        set hostAddr [CallSingleSubscriber $w $i]

}
################################################################################
proc ProcessReply {fd} {
#
# This function is invoked when we receive the empty line which
# delimits the MIME header.
#
        global conFailure replyList setup

#puts "REPLYLIST=$replyList($fd)"; flush stdout
```

```
        #
        # Get stored infos from "replyList"
        #
        set statusLabel [lindex $replyList($fd) 0]
        set winName [lindex $replyList($fd) 1]
        set index [lindex $replyList($fd) 2]
        set hostAddr [lindex $replyList($fd) 3]

        #
        # Analyse reply data and update status line.
        #
        set status ""
        set rejectReason ""
        set location ""
        set callAccepted 0

        foreach line [lrange $replyList($fd) 4 end] {
#         puts "LINE=$line"
          set tag [string tolower [lindex $line 0]]
          set value [lrange $line 1 end]
          switch $tag {
            code        {set status [lindex  $value 0]; set msg [lrange $value 1 end]}
            reason:     {set rejectReason "$value"}
            location:   {set location "$value"}
          }
        }

#       puts "STATUS=$status"
#       puts "MSG=$msg"

        if {$status == ""} {
          #
          # Protocol error, because no status code > 199 was received
          #
          set msg "Protocol error! No status code reveived!"
        }
        switch $status {
          202 {                                     # ACCEPTANCE
            set callAccepted 1
            }
          302 {                                     # TEMP. FORWARED
            set conFailure 1
            }
          303 {                                     # LOCAL REDIRECTION
            set conFailure 1
            }
          404 {
            set conFailure 1
            set msg "User is not present!"
            }
          405 {
            set conFailure 1
            set msg "User/group does not exist!"
            }
          430 {                                     # REJECTION
            set conFailure 1
            if {$rejectReason != ""} {
              set msg "Rejected, $rejectReason"
            } else {
              set msg "Rejected without reason"
            }
            }
          default {
            set conFailure 1
          }
        }
#puts "REPLYLIST=$replyList($fd)"; flush stdout

        #
        # If the dialer window has not already been disappeared we display the result
        #
        if [winfo exists $statusLabel] {
          $statusLabel configure -text "$msg"
        }

        #
        # If "replyList" is empty and no errors where occured, then
        # the dialer window may disappear. If at least one participant
        # has accepted, the conference is established.
        #
        unset replyList($fd)
        if {![llength [array names replyList]] && !$conFailure} {
          AudioEffect closeWin
          destroy $winName
          if [winfo exists .confmgr] {
            destroy .confmgr
          }
```

```
        }
        dp_shutdown $fd both
        close $fd

        if {$status == "302"} {
          if $setup(confirmFwds) {
            mkDialog .fwd warning "-text \"Please confirm forwarding address \n\n$location\" \
              -aspect 800" \
              "Confirm \"ForwardCall $winName $index $location\"" {No ""}
          } else {
            ForwardCall $winName $index $location
          }
        }
        if {$status == "303"} {
          ForwardCall $winName $index $location
        }
        if $callAccepted {
          StartMediaAgents $hostAddr
        }

}
#***************************************************************************
proc ReplyHandler {mode fd} {

        global replyList

        #
        # Broken pipe?
        #
        if {[gets $fd reply] == -1} {
          set statusLabel [lindex $replyList($fd) 0]
          if [winfo exists $statusLabel] {
            $statusLabel configure -text "Connection aborted!"
          }
          close $fd
          unset replyList($fd)
          return
        }
#puts "reply from CD: $reply"

        #
        # Read protocol line and store it in "replyList" if it isn't
        # an information line. Those lines are displayed immediately.
        # Protocol version numbers and order of lines are ignored!
        #
        if {$reply == "" && [llength $replyList($fd)]} {
          ProcessReply $fd
        } else {
          if [string match "MUCS/?.? 1?? *" $reply] {
            set statusLabel [lindex $replyList($fd) 0]
            if [winfo exists $statusLabel] {
              set msg [lrange $reply 2 end]
              $statusLabel configure -text "$msg"
            }
            return
          }
          if [string match "MUCS/?.? ??? *" $reply] {
            set replyList($fd) [lappend replyList($fd) "code [lrange $reply 1 end]"]
          } else {
            set replyList($fd) [lappend replyList($fd) "$reply"]
          }
        }
}
#***************************************************************************
proc SendFace {fd} {
#
# !!! obselete !!!
#
        global setup

        if {![catch {set f [open $setup(faceBitmap) r]} Msg]} {
          while {[gets $f line] >= 0} {
            if {![eof $fd] && $line != ""} {
              puts $fd "Face: $line"
            }
          }
          close $f
        }
}
#***************************************************************************
```

B.4 Starten des Help Processors: `Help.tcl`

```
#
```

```
# lib/Help.tcl
#
##############################################################################
proc Help {topic status} {

        global setup

        if {$setup(helpProcessor) == ""} {
          mkDialog .error warning "-text \"No help available!\" -aspect 500" {Continue ""}
        } else {
          if {$status != ""} {
            $status configure -text "Spawning help processor..."
            update
          }
          exec $setup(helpProcessor) $setup(helpURL)$topic &
        }
}
#### END OF FILE ####
```

B.5 Verwaltung der Medien-Agenten: MediaMgr.tcl

```
#
# lib/MediaMgr.tcl
#
set MAX_AGENTS 30
#
##############################################################################
proc AgentAdd1 {w listBox} {

        global agents0 setup

        $listBox insert end "*** new agent ***"
        $listBox selection clear 0 end
        $listBox selection set end

        incr setup(agent0id)
        set i $setup(agent0id)
        lappend setup(agent0ids) $i

        set agents0($i,name)      "Network Voice Terminal (NeVoT)"
        set agents0($i,cname)     "NeVoT3.32"
        set agents0($i,types)     "audio"
        set agents0($i,protocols) "RTP VAT"
        set agents0($i,fmts)      "PCMA PCMU L8 L16"
        set agents0($i,rates)     "8000 11000 44100"
        set agents0($i,cmd)       /u/foertel/ibm/bin/nevot

        AgentConfigure1 $w $listBox
}
##############################################################################
proc AgentAdd2 {w listBox} {

        global agents1 setup

        $listBox insert end "*** new agent ***"
        $listBox selection clear 0 end
        $listBox selection set end

        incr setup(agent1id)
        set i $setup(agent1id)
        lappend setup(agent1ids) $i

        set agents1($i,name)     ""
        set agents1($i,cname)     ""
        set agents1($i,default)  0
        set agents1($i,type)     audio
        set agents1($i,protocol) RTP
        set agents1($i,fmt)      L8
        set agents1($i,rate)     800
        set agents1($i,cmd)      ""

        AgentConfigure2 $w $listBox
}
##############################################################################
proc AgentConfigure1 {w listBox} {

        global agents0 iconBitmaps mediaFormats mediaRates prots Resource setup types

        if [winfo exists $w] {
          raise $w
          return
        }
```

```
set liIndex [$listBox curselection]
if {$liIndex == ""} {
  mkDialog .starterr warning {-text "Nothing selected to configure!" -aspect 800} \
    {Continue ""}
  return
}
set index [lindex $setup(agent0ids) $liIndex]

toplevel $w
wm title $w "media agent capabilities"
wm iconbitmap $w @$iconBitmaps(MediaMgr)

frame $w.first -bd 1 -relief raised
  pack [eval label $w.first.l1 -text \"Media Agent Capability Definition\" \
        $Resource(Headline) -anchor w -padx 2m -pady 3m] -side left -anchor nw
  pack [eval label $w.first.l2 -bitmap @bitmaps/phone35x25.xbm \
        -anchor e -padx 2m -pady 3m] -side right -padx 2m -pady 2m

frame $w.f2 -bd 1 -relief raised
  frame $w.f2.f1 -bd 2 -relief groove
    pack\
      [label $w.f2.f1.l -text "media type:" -anchor w] \
      [frame $w.f2.f1.f -bd 1 -height 1m -relief ridge] \
      [checkbutton $w.f2.f1.r1 -var types(1) -offval "" -onvalue "audio" \
        -text "audio" -anchor w] \
      [checkbutton $w.f2.f1.r2 -var types(2) -offval "" -onvalue "video" \
        -text "video" -anchor w] \
      [checkbutton $w.f2.f1.r3 -var types(3) -offval "" -onvalue "whiteboard" \
        -text "whiteboard" -anchor w] \
      [checkbutton $w.f2.f1.r4 -var types(4) -offval "" -onvalue "text" \
        -text "text" -anchor w] \
      -fill x -padx 1m
  frame $w.f2.f2 -bd 2 -relief groove
    pack \
      [label $w.f2.f2.l -text "protocol:" -anchor w] \
      [frame $w.f2.f2.f -bd 1 -height 1m -relief ridge] \
      [checkbutton $w.f2.f2.r1 -offval "" -onvalue "RTP" -var prots(1) \
        -text "RTP" -anchor w] \
      [checkbutton $w.f2.f2.r2 -offval "" -onvalue "VAT" -var prots(2) \
        -text "VAT" -anchor w] \
      -fill x -padx 1m
  frame $w.f2.f3 -bd 2 -relief groove
    pack \
      [label $w.f2.f3.l1 -text "format" -anchor w] \
      [frame $w.f2.f3.f -bd 1 -height 1m -relief ridge] \
      -fill x -padx 1m
    frame $w.f2.f3.f1
      pack [listbox $w.f2.f3.f1.li -yscroll "$w.f2.f3.f1.s set" -width 10 -height 6 ] \
        -exp true -side left -fill both
      pack [scrollbar $w.f2.f3.f1.s -command "$w.f2.f3.f1.li yview"] \
        -side left -fill y
      pack $w.f2.f3.f1 -exp true -fill both -padx 1m
    pack \
      [eval entry $w.f2.f3.e -wi 10 -textvar agents0($index,fmts) $Resource(Entry)] \
      -fill x -padx 1m

$w.f2.f3.e icursor end
  frame $w.f2.f4 -bd 2 -relief groove
    pack \
      [label $w.f2.f4.l -text "rate" -anchor w] \
      [frame $w.f2.f4.f -bd 1 -height 1m -relief ridge] \
      -fill x -padx 1m
    frame $w.f2.f4.f1
      pack [listbox $w.f2.f4.f1.li -yscroll "$w.f2.f4.f1.s set" -width 10 -height 6 ] \
        -exp true -side left -fill both
      pack [scrollbar $w.f2.f4.f1.s -command "$w.f2.f4.f1.li yview"] \
        -side left -fill y
      pack $w.f2.f4.f1 -exp true -fill both -padx 1m
    pack \
      [eval entry $w.f2.f4.e -wi 7 -textvar agents0($index,rates) $Resource(Entry)] \
      -fill x -padx 1m
  pack $w.f2.f1 $w.f2.f2 $w.f2.f3 $w.f2.f4 -side left \
    -pady 1m -padx 1m -exp true -fill both

frame $w.f3 -bd 1 -relief raised
  pack [label $w.f3.l1 -text "description:" -anchor w] \
    -side left -anchor w
  pack [eval entry $w.f3.e1 -textvar agents0($index,name) $Resource(Entry)] \
    -side left -anchor w -exp true -fill x -padx 1m -pady 1m

frame $w.f4 -bd 1 -relief raised
  pack [label $w.f4.l1 -text "command line:" -anchor w] \
    -side left -anchor w
  pack [eval entry $w.f4.e -textvar agents0($index,cmd) $Resource(Entry)] \
    -side left -anchor w -exp true -fill x -padx 1m -pady 1m
```

```
        frame $w.f5 -bd 1 -relief raised
          pack [label $w.f5.l1 -text "unique agent identifier:" -anchor w] \
            -side left -anchor w
          pack [eval entry $w.f5.e -textvar agents0($index,cname) $Resource(Entry)] \
            -side left -anchor w -exp true -fill x -padx 1m -pady 1m

        frame $w.last -bd 1 -relief raised
          frame $w.last.f -bd 1 -relief sunken
            pack \
              [eval button $w.last.f.b1 $Resource(ExitButton) \
                -command \"CloseAgtConf1 $w $listBox $liIndex $index\"] \
              [eval button $w.last.f.b2 $Resource(HelpButton)] \
              -side left -exp true -fill x
          pack $w.last.f -exp true -fill x -padx 1m -pady 1m

        pack \
          $w.first \
          $w.f2 \
          $w.f3 \
          $w.f4 \
          $w.f5 \
          $w.last -exp true -fill both
#
# fill form with types, format names and ratings
#
        set i 0
        foreach type $agents0($index,types) {
          incr i
          set types($i) $type
        }
        set i 0
        foreach protocol $agents0($index,protocols) {
          incr i
          set prots($i) $protocol
        }
        foreach format $mediaFormats {
          $w.f2.f3.f1.li insert end $format
        }
        foreach rate $mediaRates {
          $w.f2.f4.f1.li insert end $rate
        }
#
# bindings
#
        bind $w.f2.f3.f1.li <Double-1> "SetFormat1 $w.f2.f3 agents0($index,fmt)"
        bind $w.f2.f4.f1.li <Double-1> "SetFormat1 $w.f2.f4 agents0($index,rate)"
        bind $w.f2.f3.e <Return> "CloseAgtConf1 $w $listBox $liIndex $index"
        bind $w.f2.f4.e <Return> "CloseAgtConf1 $w $listBox $liIndex $index"
        bind $w.f3.e1 <Return> "CloseAgtConf1 $w $listBox $liIndex $index"
        bind $w.f4.e <Return> "CloseAgtConf1 $w $listBox $liIndex $index"
        bind $w.f5.e <Return> "CloseAgtConf1 $w $listBox $liIndex $index"
}
###############################################################################
proc AgentConfigure2 {w listBox} {

        global agents1 iconBitmaps mediaFormats mediaRates Resource setup

        if [winfo exists $w] {
          raise $w
          return
        }
        set liIndex [$listBox curselection]
        if {$liIndex == ""} {
          mkDialog .starterr warning {-text "Nothing selected to configure!" -aspect 800} \
            {Continue ""}
          return
        }
        set index [lindex $setup(agent1ids) $liIndex]

        toplevel $w
        wm title $w "media agent configuration"
        wm iconbitmap $w @$iconBitmaps(MediaMgr)

        frame $w.first -bd 1 -relief raised
          pack [eval label $w.first.l1 -text \"Media agent configuration\" $Resource(Headline) \
                -anchor w -padx 2m -pady 3m] -side left -anchor nw
          pack [eval label $w.first.l2 -bitmap @bitmaps/phone35x25.xbm \
                -anchor e -padx 2m -pady 3m] -side right -padx 2m -pady 2m

        frame $w.f2 -bd 1 -relief raised
          frame $w.f2.f1 -bd 2 -relief groove
            pack\
              [label $w.f2.f1.l -text "media type:" -anchor w] \
              [frame $w.f2.f1.f -bd 1 -height 1m -relief ridge] \
              [radiobutton $w.f2.f1.r1 -var agents1($index,type) -value "audio" \
                 -text "audio" -anchor w] \
              [radiobutton $w.f2.f1.r2 -var agents1($index,type) -value "video" \
```

```
            -text "video" -anchor w] \
         [radiobutton $w.f2.f1.r3 -var agents1($index,type) -value "whiteboard" \
            -text "whiteboard" -anchor w] \
         [radiobutton $w.f2.f1.r4 -var agents1($index,type) -value "text" \
            -text "text" -anchor w] \
         -fill x -padx 1m
     frame $w.f2.f2 -bd 2 -relief groove
       pack \
         [label $w.f2.f2.l -text "protocol:" -anchor w] \
         [frame $w.f2.f2.f -bd 1 -height 1m -relief ridge] \
         [radiobutton $w.f2.f2.r1 -value "RTP" -var agents1($index,protocol) \
            -text "RTP" -anchor w] \
         [radiobutton $w.f2.f2.r2 -value "VAT" -var agents1($index,protocol) \
            -text "VAT" -anchor w] \
         -fill x -padx 1m
     frame $w.f2.f3 -bd 2 -relief groove
       pack \
         [label $w.f2.f3.l1 -text "format" -anchor w] \
         [frame $w.f2.f3.f -bd 1 -height 1m -relief ridge] \
         -fill x -padx 1m
       frame $w.f2.f3.f1
         pack [listbox $w.f2.f3.f1.li -yscroll "$w.f2.f3.f1.s set" -width 10 -height 6 ] \
           -exp true -side left -fill both
         pack [scrollbar $w.f2.f3.f1.s -command "$w.f2.f3.f1.li yview"] \
           -side left -fill y
         pack $w.f2.f3.f1 -exp true -fill both -padx 1m
       pack \
         [eval entry $w.f2.f3.e -wi 10 -textvar agents1($index,fmt) $Resource(Entry)] \
         -fill x -padx 1m
     frame $w.f2.f4 -bd 2 -relief groove
       pack \
         [label $w.f2.f4.l -text "rate" -anchor w] \
         [frame $w.f2.f4.f -bd 1 -height 1m -relief ridge] \
         -fill x -padx 1m
       frame $w.f2.f4.f1
         pack [listbox $w.f2.f4.f1.li -yscroll "$w.f2.f4.f1.s set" -width 10 -height 6 ] \
           -exp true -side left -fill both
         pack [scrollbar $w.f2.f4.f1.s -command "$w.f2.f4.f1.li yview"] \
           -side left -fill y
         pack $w.f2.f4.f1 -exp true -fill both -padx 1m
       pack \
         [eval entry $w.f2.f4.e -wi 7 -textvar agents1($index,rate) $Resource(Entry)] \
         -fill x -padx 1m
     pack $w.f2.f1 $w.f2.f2 $w.f2.f3 $w.f2.f4 -side left -pady 1m \
       -padx 1m -exp true -fill both

   frame $w.f3 -bd 1 -relief raised
     pack [label $w.f3.l1 -text "description:" -anchor w] \
       -side left -anchor w
     pack [eval entry $w.f3.e1 -textvar agents1($index,name) $Resource(Entry)] \
       -side left -anchor w -exp true -fill x -padx 1m -pady 1m

   frame $w.f4 -bd 1 -relief raised
     pack [label $w.f4.l1 -text "command line:" -anchor w] \
       -side left -anchor w
     pack [eval entry $w.f4.e -textvar agents1($index,cmd) $Resource(Entry)] \
       -side left -anchor w -exp true -fill x -padx 1m -pady 1m

   frame $w.f5 -bd 1 -relief raised
     pack [label $w.f5.l1 -text "unique agent identifier:" -anchor w] \
       -side left -anchor w
     pack [eval entry $w.f5.e -textvar agents0($index,cname) $Resource(Entry)] \
       -side left -anchor w -exp true -fill x -padx 1m -pady 1m

   frame $w.f6 -bd 1 -relief raised
     pack [checkbutton $w.f6.c -var agents1($index,default) \
       -text "start this agent by default"] \
       -side left -anchor w

   frame $w.last -bd 1 -relief raised
     frame $w.last.f -bd 1 -relief sunken
       pack \
         [eval button $w.last.f.b1 $Resource(ExitButton) \
            -command \"CloseAgtConf2 $w $listBox $liIndex $index\"] \
         [eval button $w.last.f.b2 $Resource(HelpButton)] \
         -side left -exp true -fill x
     pack $w.last.f -exp true -fill x -padx 1m -pady 1m

   pack \
     $w.first \
     $w.f2 \
     $w.f3 \
     $w.f4 \
     $w.f5 \
     $w.f6 \
     $w.last -exp true -fill both
#
```

```
# fill lists with format names and ratings
#
        foreach format $mediaFormats {
          $w.f2.f3.f1.li insert end $format
        }
        foreach rate $mediaRates {
          $w.f2.f4.f1.li insert end $rate
        }
#
# bindings
#
        bind $w.f2.f3.f1.li <Double-1> "SetFormat2 $w.f2.f3 agents1($index,fmt)"
        bind $w.f2.f4.f1.li <Double-1> "SetFormat2 $w.f2.f4 agents1($index,rate)"
        bind $w.f2.f3.e <Return> "CloseAgtConf2 $w $listBox $liIndex $index"
        bind $w.f2.f4.e <Return> "CloseAgtConf2 $w $listBox $liIndex $index"
        bind $w.f3.e1 <Return> "CloseAgtConf2 $w $listBox $liIndex $index"
        bind $w.f4.e <Return> "CloseAgtConf2 $w $listBox $liIndex $index"
        bind $w.f5.e <Return> "CloseAgtConf2 $w $listBox $liIndex $index"
}
###############################################################################
proc AgentRemove1 {listBox} {

        global agents0 setup

        set liIndex [$listBox curselection]
        if {$liIndex == ""} {
          mkDialog .starterr warning {-text "Nothing selected to remove!" -aspect 800} \
            {Continue ""}
          return
        }
        set index [lindex $setup(agent0ids) $liIndex]
        if {$index != ""} {
          mkDialog .ask question \
            "-text \"Really remove agent definition for '$agents0($index,name)'?\" \
            -aspect 800" "Yes \"RemoveAgent1 $listBox $liIndex\"" {No ""}
        }
}
###############################################################################
proc AgentRemove2 {listBox} {

        global agents1 setup

        set liIndex [$listBox curselection]
        if {$liIndex == ""} {
          mkDialog .starterr warning {-text "Nothing selected to remove!" -aspect 800} \
            {Continue ""}
          return
        }
        set index [lindex $setup(agent1ids) $liIndex]
        if {$index != ""} {
          mkDialog .ask question \
            "-text \"Really remove agent definition for '$agents1($index,name)'?\" \
            -aspect 800" "Yes \"RemoveAgent2 $listBox $liIndex\"" {No ""}
        }
}
###############################################################################
proc AgentStart {listBox} {

        global agents1 setup

        set liIndex [$listBox curselection]
        if {$liIndex == ""} {
          mkDialog .starterr warning {-text "Nothing selected to start!" -aspect 800} ß
            {Continue ""}
          return
        }
        set index [lindex $setup(agent1ids) $liIndex]
        if {$agents1($index,cmd) == ""} {
          mkDialog .starterr warning {-text "Cannot start agent, no command line defined!" \
            -aspect 800} {Continue ""}
          return
        }

        eval exec $agents1($index,cmd) &
}
###############################################################################
proc CloseAgtConf1 {w listBox liIndex index} {

        global agents0 prots types

        if {$agents0($index,name) == ""} {
          mkDialog .err error {-text "A name is required for each media agent" -aspect 800} \
            {Continue ""}
          focus $w.f3.e1
          return
        }
        $listBox delete $liIndex
```

```
        $listBox insert $liIndex $agents0($index,name)
        $listBox selection set $liIndex
        destroy $w

        set agents0($index,types) {}
        foreach i [array names types] {
          if {$types($i) != ""} {
            lappend agents0($index,types) $types($i)
          }
        }
        set agents0($index,protocols) {}
        foreach i [array names prots] {
          if {$prots($i) != ""} {
            lappend agents0($index,protocols) $prots($i)
          }
        }

}
#############################################################################
proc CloseAgtConf2 {w listBox liIndex index} {

        global agents1

        if {$agents1($index,name) == ""} {
          mkDialog .err error {-text "A name is required for each media agent" -aspect 800} \
            {Continue ""}
          focus $w.f3.e1
          return
        }
        $listBox delete $liIndex
        $listBox insert $liIndex $agents1($index,name)
        $listBox selection set $liIndex
        destroy $w
}
#############################################################################
proc MediaManager {w} {

        global agents0 agents1 iconBitmaps MAX_AGENTS Resource setup

        if [winfo exists $w] {
          wm deiconify $w
          raise $w
          return
        }

        AudioEffect openWin
        toplevel $w
        wm title $w "Media Manager"
        wm iconbitmap $w @$iconBitmaps(MediaMgr)

        frame $w.f0 -bd 1 -relief raised
          pack\
            [eval button $w.f0.b1 $Resource(ExitButton) -command \"destroy $w\"] \
            [eval button $w.f0.b2 $Resource(HelpButton)] \
            -side bottom -padx 1m -pady 1m -fill x
        frame $w.f1 -bd 1 -relief raised
          pack [eval label $w.f1.l1 -text \"Media Manager\" $Resource(Headline) \
                -anchor w -padx 2m -pady 3m] -side left -anchor nw
          pack [eval label $w.f1.l2 -bitmap @bitmaps/phone35x25.xbm \
                -anchor e -padx 2m -pady 3m] -side right -padx 2m -pady 2m

        frame $w.f2 -bd 1 -relief raised
          frame $w.f2.f1
            pack [label $w.f2.f1.l1 -text "agent capability definition:" -anchor w] -anchor w
            pack [eval listbox $w.f2.f1.l -yscroll \"$w.f2.f1.s set\" \
                  -height 5 -width 40 $Resource(Display)] \
              -side left -expand true -fill both
            pack [scrollbar $w.f2.f1.s -command "$w.f2.f1.t yview"] \
              -side left -fill y
          frame $w.f2.f2 -bd 1 -relief sunken
            pack\
              [button $w.f2.f2.b2 -text "Add" \
                -command "AgentAdd1 $w.agtcfg $w.f2.f1.l"] \
              [button $w.f2.f2.b3 -text "Remove" \
                -command "AgentRemove1 $w.f2.f1.l"] \
              [button $w.f2.f2.b4 -text "Configure" \
                -command "AgentConfigure1 $w.agtcfg $w.f2.f1.l"] \
              -side left -expand true -fill x
          pack $w.f2.f1 $w.f2.f2 -expand true -fill x -padx 1m -pady 1m

        frame $w.f3 -bd 1 -relief raised
          frame $w.f3.f1
            pack [label $w.f3.f1.l1 -text "agents to start:" -anchor w] -anchor w
            pack [eval listbox $w.f3.f1.l -yscroll \"$w.f3.f1.s set\" \
                  -height 5 -width 40 $Resource(Display)] \
              -side left -expand true -fill both
            pack [scrollbar $w.f3.f1.s -command "$w.f3.f1.s yview"] \
```

```
            -side left -fill y
        $w.f3.f1.l selection set 0

          frame $w.f3.f2 -bd 1 -relief sunken
            pack \
              [button $w.f3.f2.b1 -text "Start" -command "AgentStart $w.f3.f1.l"] \
              [button $w.f3.f2.b2 -text "Add" \
                -command "AgentAdd2 $w.agtcfg $w.f3.f1.l"] \
              [button $w.f3.f2.b3 -text "Remove" \
                -command "AgentRemove2 $w.f3.f1.l"] \
              [button $w.f3.f2.b4 -text "Configure" \
                -command "AgentConfigure2 $w.agtcfg $w.f3.f1.l"] \
              -side left -expand true -fill x
          pack $w.f3.f1 $w.f3.f2 -expand true -fill x -padx 1m -pady 1m

        pack $w.f0 -side left -fill y
        pack $w.f1 $w.f2 $w.f3 -expand true -fill both
#
# fill listbox widgets with media descriptions
#
        foreach id $setup(agent0ids) {
          $w.f2.f1.l insert end $agents0($id,name)
        }
        foreach id $setup(agent1ids) {
          $w.f3.f1.l insert end $agents1($id,name)
        }
#
# bindings
#
        bind $w.f2.f1.l <Double-1> "AgentConfigure1 $w.agtcfg1 $w.f2.f1.l"
        bind $w.f3.f1.l <Double-1> "AgentConfigure2 $w.agtcfg2 $w.f3.f1.l"
}
##############################################################################
proc RemoveAgent1 {listBox liIndex} {

        global agents0 setup

        $listBox delete $liIndex
        $listBox selection set $liIndex
        set i [lindex $setup(agent0ids) $liIndex]
        set setup(agent0ids) [lreplace $setup(agent0ids) $liIndex $liIndex]

        unset agents0($i,name)
        unset agents0($i,types)
        unset agents0($i,protocols)
        unset agents0($i,fmts)
        unset agents0($i,rates)
        unset agents0($i,cmd)
}
##############################################################################
proc RemoveAgent2 {listBox liIndex} {

        global agents1 setup

        $listBox delete $liIndex
        $listBox selection set $liIndex
        set i [lindex $setup(agent1ids) $liIndex]
        set setup(agent1ids) [lreplace $setup(agent1ids) $liIndex $liIndex]

        unset agents1($i,name)
        unset agents1($i,default)
        unset agents1($i,type)
        unset agents1($i,protocol)
        unset agents1($i,fmt)
        unset agents1($i,rate)
        unset agents1($i,cmd)
}
##############################################################################
proc SetFormat1 {baseName var} {

        global $var

        set var [$baseName.f1.li get [$baseName.f1.li curselection]]
        $baseName.e insert end " $var"
}
##############################################################################
proc SetFormat2 {baseName var} {

        global $var

        set var [$baseName.f1.li get [$baseName.f1.li curselection]]
        $baseName.e delete 0 end
        $baseName.e insert 0 "$var"
}
##############################################################################
```

B.6 Anrufsignalisierung: `Notify.tcl`

```
#
# lib/Notify.tcl
#
##############################################################################
proc CallHandler {mode fd} {

	global afterId callList

#puts "CALL HANDLER INVOKED $fd!";  flush stdout
	if {[gets $fd callData] == -1} {
	  close $fd;				# close socket
	  catch {after cancel $afterId};		# stop ringing
	  unset callList($fd);			# remove call request
	  catch {destroy .call$fd};		# close notifiy window
	  mkDialog .abort warning {-text "Call aborted by peer!" -aspect 800} {Continue ""}
	  return
	}
#puts "callData=$callData"; flush stdout

	if {$callData == ""} {
	  #
	  # Call request complete! Parse request lines saved in "callList" and try
	  # to find matching media agents and notify user. If not at least one
	  # media agents was found, we send a mail to the recipient and terminate.
	  #
	  if {[ParseCallData $fd] == -1} {
	    unset callList($fd);			# remove call request
	    catch {after cancel $afterId};		# stop ringing
	    close $fd;				# close pipe
	  } else {
	    if [FindMediaAgents $fd] {
	      CallNotification .call$fd $fd
	    } else {
	      NotifyViaMail $fd
	      exit
	    }
	  }
	} else {
	  lappend callList($fd) $callData
	}
}
##############################################################################
proc CallNotification {w fd} {

	global afterId callData iconBitmaps MUCS_VERSION setup sounds Resource

	if $setup(audibleNotification) {
	  DingDong
	}

	dp_send $fd "$MUCS_VERSION 112 Call is being signaled"

	#
	# make front-end
	#
	toplevel $w
	wm title $w "Incomming call!"
	wm iconbitmap $w @$iconBitmaps(Notify)

	if $setup(iconifiedNotification) {
	  wm iconify $w
	}

	frame $w.headline -bd 1 -relief raised
	eval label $w.headline.l -text \"Incoming Call!\" $Resource(Headline) \
	  -anchor w -padx 2m -pady 3m
	eval label $w.headline.b -bitmap @bitmaps/phone35x25.xbm -anchor e -padx 2m -pady 3m
	pack $w.headline.l -side left -anchor nw
	pack $w.headline.b -side right

	frame $w.f1 -relief raised -bd 1
	  frame $w.f1.left -bd 2 -relief ridge
	    pack [label $w.f1.left.face1 -text "place"]
	    pack [label $w.f1.left.face2 -text ".face bitmap"]
	    pack [label $w.f1.left.face3 -text "here"]
	    pack $w.f1.left -side left -fill x -padx 2m -pady 1m

	  frame $w.f1.right
	  frame $w.f1.right.l0
	    if [info exists callData($fd,prio)] {
	      pack [eval label $w.f1.right.l0.l2 \
			-text \{$callData($fd,prio)\} -anchor w $Resource(Display)] \
		-side right -exp true -fill x -anchor e
	      pack [label $w.f1.right.l0.l1 -wi 8 -text "priority:" -anchor e] \
		-side right -anchor e
```

```
          }
        frame $w.f1.right.l1
          pack [eval label $w.f1.right.l1.l2 -text {$callData($fd,origin)} \
            -anchor w $Resource(Display)] \
            -side right -exp true -fill x -anchor e
          pack [label $w.f1.right.l1.l1 -wi 8 -text "from:" -anchor e] \
            -side right -anchor e
        frame $w.f1.right.l2
          pack [eval label $w.f1.right.l2.l2 -text {$callData($fd,userName)} \
            -anchor w $Resource(Display)] \
            -side right -exp true -fill x -anchor e
          pack [label $w.f1.right.l2.l1 -wi 8 -text "to:" -anchor e] \
            -side right -anchor e
        frame $w.f1.right.l3
          if [info exists callData($fd,info)] {
            pack [eval label $w.f1.right.l3.l2 -text {$callData($fd,info)} \
              -anchor w $Resource(Display)] \
              -side right -exp true -fill x -anchor e
            pack [label $w.f1.right.l3.l1 -wi 8 -text "Subject:" -anchor e] \
               -side right -anchor e
          }
      pack $w.f1.right.l0 $w.f1.right.l1 $w.f1.right.l2 $w.f1.right.l3 \
        -side top -exp true -fill x
      #
      # display one line for each media requested
      #
      set i 4
      set j 0
      foreach media $callData($fd,mediaDescr) {
        frame $w.f1.right.l$i
          pack [eval label $w.f1.right.l$i.l2 -text {$media} -anchor w $Resource(Display)] \
            -side right -exp true -fill x -anchor e
          pack [label $w.f1.right.l$i.l1 -wi 8 -text "media:" -anchor e] \
            -side right -anchor e
          pack $w.f1.right.l$i -side top -exp true -fill x
        if {[lindex $callData($fd,agentIds) $j] == "AGENT_NOT_FOUND"} {
          $w.f1.right.l$i.l2 configure -fg firebrick -text "$media (unavailable)"
        }
        incr i
        incr j
      }
        pack $w.f1.right -side left -fill x -exp true -padx 5m -pady 1m

      frame $w.f2 -relief raised -bd 1
        pack [button $w.f2.b1 -wi 10 -text "Forward" -command "ReplyToCall $w $fd FORWARD"] \
          -padx 2m -pady 2m -anchor w -side left
        pack [button $w.f2.b2 -wi 10 -text "Ph. book" \
          -command "Phonebook .pb2 $w.f2.b2 $w.f2.e SINGLE"] \
          -anchor e -side right -padx 2m
        pack \
          [eval entry $w.f2.e -width 30 $Resource(Entry) -textvariable forwardTo] \
          [label $w.f2.l -text "to:"] \
          -anchor e -side right

      frame $w.f3 -relief raised -bd 1
        pack [button $w.f3.b1 -wi 10 -text "Reject" -command "ReplyToCall $w $fd REJECT"] \
          -padx 2m -pady 2m -anchor w -side left
        pack [button $w.f3.b2 -wi 10 -text "Reasons" -command "ReasonList $w.reason $w.f3.e"] \
          -padx 2m -anchor e -side right
        pack \
          [eval entry $w.f3.e -width 30 $Resource(Entry) -textvariable rejectReason] \
          [label $w.f3.l -text "reason:"] \
          -anchor e -side right

      frame $w.f4 -relief raised -bd 1
        pack [button $w.f4.join1 -wi 10 -text "Accept" -command "ReplyToCall $w $fd ACCEPT"] \
          -padx 2m -pady 3m -anchor w -side left
        pack [eval button $w.f4.help -wi 10 $Resource(HelpButton) \
          -command \"Help IncommingCall {} \"] \
          -padx 2m -pady 3m -anchor e -side right

      pack \
        $w.headline \
        $w.f1 \
        $w.f2 \
        $w.f3 \
        $w.f4 \
        -side top -expand true -fill x

      focus $w.f4.join1

      if {$setup(iconifiedNotification) && $setup(audibleNotification)} {
        bind $w <Visibility> {after cancel $afterId}
      }
}
################################################################################
proc DingDong {} {
```

```
        global setup sounds afterId

        catch "eval exec $setup(audioPlayer) $sounds(callNotify) &"
        set afterId [after $setup(ringerIntervall) DingDong]
}
###############################################################################
proc FindMediaAgents {fd} {

        global agents0 callData setup

        set hits 0
        foreach mediaAgent $callData($fd,mediaDescr) {
#puts "INCOMMING CALL! Scanning for: $mediaAgent"
          set mediaType [lindex $mediaAgent 0]
          set mediaProtocol [lindex $mediaAgent 1]
          set mediaFormat [lindex $mediaAgent 2]
          set mediaRate [lindex $mediaAgent 3]
          set mediaAddr [lindex $mediaAgent 4]
          set mediaPort [lindex $mediaAgent 5]
          if {$mediaAddr == "" || $mediaPort == ""} {
            continue
          }
          set found 0
          foreach id $setup(agent0ids) {
            if {[lsearch $agents0($id,types) $mediaType] >= 0 && \
                [lsearch $agents0($id,protocols) $mediaProtocol] >= 0 && \
                [lsearch $agents0($id,fmts) $mediaFormat] >= 0 && \
                [lsearch $agents0($id,rates) $mediaRate] >= 0} {
              if {$agents0($id,cmd) != ""} {
                set found 1
                incr hits
                lappend callData($fd,agentIds) "$id $mediaAddr $mediaPort"
                break;
              }
            }
          }
          if !$found {
            lappend callData($fd,agentIds) "AGENT_NOT_FOUND"
          }
        }
        if $hits {
          return $hits
        } else {
          return 0
        }
}
###############################################################################
proc NotifyViaMail {fd} {

        global callData env setup

        if {$setup(mailDeliveryAgent) == ""} {
          return
        }
        set fn $env(HOME)/.mucs-mail$fd
        if {![catch {set f [open $fn w]} Msg]} {
          puts $f "Subject: MUCS invitation rejected by conference controller"
          puts $f ""
          puts $f "    FROM: $callData($fd,origin)"
          puts $f "    DATE: "
          foreach mediaAgent $callData($fd,mediaDescr) {
            puts $f "   MEDIA: $mediaAgent"
          }
          puts $f "  REASON: no matching media agent configured!"
          close $f
          exec $setup(mailDeliveryAgent) $env(USER) < $fn
          exec rm $fn
        }
}
###############################################################################
proc ParseCallData {fd} {

        global callData callList setup
#
#       extract/analyse call data
#
        set callData($fd,userName) ""
        set callData($fd,origin) ""
        set callData($fd,mediaDescr) ""
        foreach line $callList($fd) {
          set tag [string tolower [lindex $line 0]]
          set value [lrange $line 1 end]
#puts "TAG=$tag"
#puts "VAL=$value"
          switch $tag {
            call        {set callData($fd,userName) [lindex $value 0]}
```

```
            info:       {set callData($fd,info) "$value"}
            origin:     {set callData($fd,origin) "$value"}
            media:      {lappend callData($fd,mediaDescr) "$value"}
            priority:   {if {$value < $setup(callFence)} {
                           dp_send $fd "Error: priority too low!"
                           return -1
                         } else {
                           set callData($fd,prio) "$value"
                         }
                        }
          }
        }
#       unset callList($fd)
        return 0
}
################################################################################
proc ReasonList {w {entryWidget NULL}} {

        global newReason Resource setup

        toplevel $w
        wm title $w "Reasons"
        wm iconbitmap $w @bitmaps/phone35x25.xbm

        eval label $w.l -text \"Reject Reasons\" $Resource(Headline) -anchor w -padx 2m -pady 3m

        frame $w.f2
          pack [scrollbar $w.f2.scroll -command "$w.f2.list yview"] \
            -side right -fill y
          pack [listbox $w.f2.list -yscroll "$w.f2.scroll set" -setgrid 1 -height 6 -width 30] \
            -side left -expand 1 -fill both -padx 1m -pady 1m

        frame $w.f3
          frame $w.f3.f1 -bd 1 -relief sunken
            pack \
              [button $w.f3.f1.b1 -text "Add" \
                -com "$w.f2.list insert 0 \$newReason; lappend setup(reasons) \$newReason"] \
              [button $w.f3.f1.b2 -text "Delete" -com "RemoveReason $w.f2.list"] \
              [button $w.f3.f1.b3 -text "Default" -com "SetDefaultReason $w.f2.list"] \
              -expand true -fill both -side left
            pack $w.f3.f1 -exp true -fill x -padx 1m -pady 1m

        frame $w.f4
          pack [eval entry $w.f4.e -textvar newReason $Resource(Entry)] \
            -expand true -fill both -padx 1m -pady 1m
        bind $w.f4.e <Return> "$w.f2.list insert 0 \$newReason; lappend setup(reasons) \$newReason"

        frame $w.f5
          frame $w.f5.f1 -bd 1 -relief sunken
            pack \
              [eval button $w.f5.f1.b1 $Resource(ExitButton) -command \"destroy $w\"] \
              [eval button $w.f5.f1.b2 $Resource(HelpButton) -command \"Help PHONEBOOK\"] \
              -expand true -fill both -side left
          pack $w.f5.f1 -exp true -fill x -padx 1m -pady 1m

        pack $w.l $w.f2 $w.f3 $w.f4 $w.f5 -expand true -fill both
#
#       fill list with reasons
#
        foreach reason $setup(reasons) {
          $w.f2.list insert end $reason
        }
        if {$entryWidget != "NULL"} {
          bind $w.f2.list <Double-Button-1> \
            "$entryWidget insert end \[selection get\];after 100 destroy $w"
        }
}
################################################################################
proc RemoveReason {w} {

        global setup

        set i [$w curselection]
        $w delete $i $i
        $w select set $i
}
################################################################################
proc ReplyToCall {w fd action} {

        global  agents0 afterId callData forwardTo MUCS_VERSION rejectReason \
                setup taskList

        catch {after cancel $afterId};                  # stop ringing

        if {$action == "FORWARD"} {
          if {$forwardTo == ""} {
            mkDialog .err error {-text "Specify new destination please!" -aspect 800} \
```

```
            {Continue ""}
          focus $w.f2.e
          return
        }
        dp_send $fd "$MUCS_VERSION 302 Forward to $forwardTo"
        dp_send $fd "Location: $forwardTo"
        after 1000 exit
      }
      if {$action == "ACCEPT"} {
        dp_send $fd "$MUCS_VERSION 202 Call accepted"
        #
        # launch media agents
        #
        foreach agt $callData($fd,agentIds) {
          set id [lindex $agt 0]
          if {$id != "AGENT_NOT_FOUND"} {
            set addr [lindex $agt 1]
            set port [lindex $agt 2]
            eval exec $agents0($id,cmd) &
            lappend taskList "\"agent $agents0($id,cname) attached\" \
              \"CreateMediaSession $id $addr $port\""
          }
        }
      }
      if {$action == "REJECT"} {
        dp_send $fd "$MUCS_VERSION 430 Call rejected"
        if {$rejectReason != ""} {
          dp_send $fd "Reason: $rejectReason"
        }
        after 1000 exit
      }
      dp_send $fd ""
      dp_shutdown $fd both
      destroy $w
}
##############################################################################
proc SetDefaultReason {w} {

      global setup

}
#### END OF FILE ####
```

B.7 Das Phone Directory: `Phonebook.tcl`

```
#
# lib/Phonebook.tcl
#
##############################################################################
proc Phonebook {w {b NULL} {entryWidget NULL} {selType NULL}} {

      global aliasList iconBitmaps Resource

      if [winfo exists $w] {
        wm deiconify $w
        raise $w
        return
      }

      $b configure -state disabled
      AudioEffect openWin

      toplevel $w
      wm title $w "Phone Directory"
      wm iconbitmap $w @$iconBitmaps(Phonebook)

      eval label $w.l -text \"Phone Directory\" $Resource(Headline) -anchor w
      pack $w.l -fill x -padx 1m -pady 2m

      frame $w.f2
        scrollbar $w.f2.scroll -command "$w.f2.list yview"
        eval listbox $w.f2.list -yscroll \"$w.f2.scroll set\" \
          $Resource(Display) -setgrid true -height 7 -width 40
        pack $w.f2.scroll -side right -fill y
        pack $w.f2.list -side left -expand 1 -fill both
      pack $w.f2 -exp true -fill both -padx 1m

      frame $w.f3 -bd 1 -relief sunken
        pack \
          [button $w.f3.b3 -text "Select" \
            -com "ReturnSelectedName $w $w.f2.list $entryWidget $selType"] \
          [eval button $w.f3.b1 $Resource(ExitButton) \
            -com \"$b configure -state normal\; AudioEffect closeWin\; destroy $w \"] \
          [eval button $w.f3.b2 $Resource(HelpButton) -command \"Help Phonebook\"] \
```

```
            -side left -exp true -fill x
        pack $w.f3 -fill x -padx 1m -pady 1m
        if {$entryWidget == "NULL"} {
          $w.f3.b3 configure -state disabled
        }
#
# Pointer bindings
#
        if {$entryWidget != "NULL"} {
          bind $w.f2.list <Double-1> "
             ReturnSelectedName $w $w.f2.list $entryWidget $selType
            $b configure -state normal
            after 100 destroy $w
          "
        }
#
# read aliases from  /.mailrc
#
        set aliasList {}
        if ![catch {set fd [open  /.mailrc r]} msg] {
          while {[gets $fd line] >= 0} {
            set list [split $line " "]
            if {[lindex $list 0] == "alias" && [llength $list] > 2} {
              set aliasList [linsert $aliasList 0 [lrange $list 1 end]]
              $w.f2.list insert 0 \
                [format "%-12s  %-25s" [lindex $list 1] [lindex $list 2] ]
            }
          }
          close $fd
          $w.f2.list selection set 0
        }
}
################################################################################
proc ReturnSelectedName {w listboxWidget entryWidget selType} {

        global aliasList MAX_ENTRIES

        set liIndex [$listboxWidget curselection]
        if {$liIndex == ""} {
          mkDialog .nothingSelected warning {-text "Nothing selected in directory list" \
            -aspect 800} {Continue ""}
          return
        }
        set receiveList [lindex $aliasList $liIndex]
        if {$selType == "SINGLE"} {
          set addr [lindex $receiveList 1]
          if {[llength [split $addr @]] == 1} {
            set addr "$addr@[idstring "%d"]"
          }
          $entryWidget insert end "$addr"
          $entryWidget xview end
        } else {
          set i 1
          foreach e [lrange $receiveList 1 end] {
            while {[$entryWidget.f$i.e$i get] != "" && $i < $MAX_ENTRIES}    {
              incr i
            }
            if {$i == $MAX_ENTRIES && [$entryWidget.f$i.e$i get] != ""} {
              mkDialog $w.err info {-text "address list full" -aspect 800} {Continue ""}
            }
            if {[llength [split $e @]] == 1} {
              set e "$e@[idstring "%d"]"
            }
            $entryWidget.f$i.e$i insert end "$e"
            $entryWidget.f$i.e$i xview end
          }
        }
}
#### END OF FILE ####
```

B.8 Die Programmkonfiguration: Setup.tcl

```
#
# lib/Setup.tcl
#
#*******************************************************************************
proc SetUp {w} {

        global iconBitmaps Resource setup sounds

        if [winfo exists $w] {
          wm deiconify $w
          raise $w
          return
```

```
}

AudioEffect openWin
toplevel $w
wm title $w "Settings"
wm iconbitmap $w @$iconBitmaps(Setup)

frame $w.headline -bd 1 -relief raised
  eval label $w.headline.l -text "Settings" $Resource(Headline) -anchor w \
    -padx 2m -pady 3m
  eval label $w.headline.b -bitmap @bitmaps/phone35x25.xbm -anchor e -padx 2m -pady 3m
  pack $w.headline.l -side left -anchor nw
  pack $w.headline.b -side right

frame $w.line1 -relief raised -bd 1
  frame $w.line1.f1 -relief groove -bd 2 -width 5c
    pack [label $w.line1.f1.l -text "Miscellaneous"]
    pack [frame $w.line1.f1.f -bd 1 -height 1m -relief ridge] -fill x -side top -padx 1m
    pack [checkbutton $w.line1.f1.c1 -text "audio effects" \
          -variable setup(audioEffects) -relief flat] -anchor w -padx 2m
    pack [checkbutton $w.line1.f1.c2 -text "iconified call notification" \
          -variable setup(iconifiedNotification) -relief flat] -anchor w -padx 2m
    pack [checkbutton $w.line1.f1.c3 -text "save settings on exit" \
          -variable setup(saveSettings) -relief flat] -anchor w -padx 2m
    pack [checkbutton $w.line1.f1.c4 -text "send face bitmap" \
          -variable setup(sendFaceBitmap) -relief flat] -anchor w -padx 2m
    pack [checkbutton $w.line1.f1.c5 -text "ringer on" \
          -variable setup(audibleNotification) -relief flat] -anchor w -padx 2m
    pack [checkbutton $w.line1.f1.c6 -text "terminate pmm on exit" \
          -variable setup(pmmKill) -relief flat] -anchor w -padx 2m
    pack [checkbutton $w.line1.f1.c7 -text "confirm temporary forwarding" \
          -variable setup(confirmFwds) -relief flat] -anchor w -padx 2m
  pack $w.line1.f1 -side left -padx 2m -pady 2m -fill both
  frame $w.line1.f2 -relief groove -bd 2 -width 5c
    pack [label $w.line1.f2.l -text "Sounds:"]
    pack [frame $w.line1.f2.f -bd 1 -height 1m -relief ridge] -fill x -side top -padx 1m
    frame $w.line1.f2.f1
      pack \
        [eval entry $w.line1.f2.f1.e -width 30 -textvar sounds(openWin) $Resource(Entry)] \
        [label $w.line1.f2.f1.l -text "window opening:" -cursor {hand2 red}] \
        -side right
bind $w.line1.f2.f1.l <Button-1> {catch "eval exec $setup(audioPlayer) $sounds(openWin)"}

    frame $w.line1.f2.f2
      pack \
        [eval entry $w.line1.f2.f2.e -width 30 -textvar sounds(closeWin) $Resource(Entry)] \
        [label $w.line1.f2.f2.l -text "window closure:" -cursor {hand2 red}] \
        -side right
bind $w.line1.f2.f2.l <Button-1> {catch "eval exec $setup(audioPlayer) $sounds(closeWin)"}

    frame $w.line1.f2.f3
      pack \
        [eval entry $w.line1.f2.f3.e -width 30 -textvar sounds(warning) $Resource(Entry)] \
        [label $w.line1.f2.f3.l -text "message box:" -cursor {hand2 red}] \
        -side right
bind $w.line1.f2.f3.l <Button-1> {catch "eval exec $setup(audioPlayer) $sounds(warning)"}

    frame $w.line1.f2.f4
      pack\
        [eval entry $w.line1.f2.f4.e -width 30 -textvar sounds(welcome) $Resource(Entry)] \
        [label $w.line1.f2.f4.l -text "program start:" -cursor {hand2 red}] \
        -side right
bind $w.line1.f2.f4.l <Button-1> {catch "eval exec $setup(audioPlayer) $sounds(welcome)"}

    frame $w.line1.f2.f5
      pack \
        [eval entry $w.line1.f2.f5.e -width 30 -textvar sounds(progExit) $Resource(Entry)] \
        [label $w.line1.f2.f5.l -text "program termination:" -cursor {hand2 red}] \
        -side right
bind $w.line1.f2.f5.l <Button-1> {catch "eval exec $setup(audioPlayer) $sounds(progExit)"}

    pack $w.line1.f2.f1 $w.line1.f2.f2 $w.line1.f2.f3 $w.line1.f2.f4 $w.line1.f2.f5 \
      -side top -exp true -fill x -padx 2m
  pack $w.line1.f2 -side right -padx 2m -pady 2m -fill both

frame $w.line3 -bd 1 -relief raised
  pack [label $w.line3.l -wi 20 -text "audio play command:" -anchor e] \
    -anchor e -side left -fill x -padx 2m
  pack [eval entry $w.line3.e -textvar setup(audioPlayer) $Resource(Entry)] \
    -exp true -fill x -padx 2m -pady 1m

frame $w.line4 -bd 1 -relief raised
  frame $w.line4.f1
    pack [label $w.line4.f1.l -wi 20 -text "help processor:" -anchor e] \
      -anchor e -side left -fill x -padx 2m
    pack [eval entry $w.line4.f1.e -textvar setup(helpProcessor) $Resource(Entry)] \
      -exp true -fill x -padx 2m
```

```
          pack $w.line4.f1 -fill x
        frame $w.line4.f2
          pack [label $w.line4.f2.l -wi 20 -text "help URL:" -anchor e] \
            -anchor e -side left -fill x -padx 2m
          pack [eval entry $w.line4.f2.e -textvar setup(helpURL) $Resource(Entry)] \
            -exp true -fill x -padx 2m
          pack $w.line4.f2 -exp true -fill x

      frame $w.line5 -bd 1 -relief raised
        pack [label $w.line5.l -wi 20 -text "face bitmap file:" -anchor e] \
          -anchor e -side left -fill x -padx 2m
        pack [eval entry $w.line5.e -textvar setup(faceBitmap) $Resource(Entry)] \
          -exp true -fill x -padx 2m -pady 1m

      frame $w.line6 -bd 1 -relief raised
        frame $w.line6.f1
          pack [label $w.line6.f1.l1 -wi 20 -text "ringer audio file:" \
            -cursor {hand2 red} -anchor e] -anchor e -side left -fill x -padx 2m
          pack [eval entry $w.line6.f1.e1 -textvar sounds(callNotify) $Resource(Entry)] \
            -side left -exp true -fill x -padx 2m
          pack $w.line6.f1 -expand true -fill x
        frame $w.line6.f2
          pack [label $w.line6.f2.l2 -wi 20 -text "intervall (msec):" -anchor e] \
            -side left -anchor e -fill x -padx 2m
          pack [eval entry $w.line6.f2.e2 -width 5 \
            -textvar setup(ringerIntervall) $Resource(Entry)] \
            -side left -padx 2m
          pack $w.line6.f2 -exp true -fill x
      bind $w.line6.f1.l1 <Button-1> {catch "eval exec $setup(audioPlayer) $sounds(callNotify)"}

      frame $w.line7 -bd 1 -relief raised
        frame $w.line7.f1
          pack [label $w.line7.f1.l -wi 20 -text "pmm startup command:" -anchor e] \
            -anchor e -side left -fill x -padx 2m
          pack [eval entry $w.line7.f1.e -textvar setup(pmmCmd) $Resource(Entry)] \
            -exp true -fill x -padx 2m
          pack $w.line7.f1 -fill x
        frame $w.line7.f2
          pack [label $w.line7.f2.l1 -wi 20 -text "pmm hostname:" -anchor e] \
            -anchor e -side left -fill x -padx 2m
          pack [eval entry $w.line7.f2.e1 -textvar setup(pmmHost) $Resource(Entry)] \
            -side left -exp true -fill x -padx 2m
          pack [label $w.line7.f2.l2 -text "port #:"] -side left -fill x
          pack [eval entry $w.line7.f2.e2 -wi 6 -textvar setup(pmmPort) $Resource(Entry)] \
            -side left -padx 2m
          pack $w.line7.f2 -exp true -fill x

      frame $w.line2 -bd 1 -relief raised
        frame $w.line2.f1 -bd 1 -relief sunken
          pack \
            [button $w.line2.f1.save -text "Save & Dismiss" \
              -com "SaveSettings; AudioEffect closeWin; destroy $w"] \
            [button $w.line2.f1.reasons -text "Reasons" \
              -com "ReasonList .reasons"] \
            [button $w.line2.f1.phonebook -text "Phone Directory" \
              -com "Phonebook .phonebook $w.line2.f1.phonebook "] \
            [eval button $w.line2.f1.exit $Resource(ExitButton) \
              -com \"AudioEffect closeWin\; destroy $w\"] \
            [eval button $w.line2.f1.help $Resource(HelpButton) \
              -com \"Help Setup $w.status.l \"] \
            -side left -exp true -fill x
          pack $w.line2.f1 -exp true -fill x -padx 1m -pady 1m

      frame $w.status -bd 1 -relief raised
        pack [eval label $w.status.l $Resource(Statusline) \
          -text \"Change setup parameters and finish by pressing the DISMISS button...\"] \
          -side left -exp true -fill x

      pack \
        $w.headline \
        $w.line1 \
        $w.line3 \
        $w.line4 \
        $w.line5 \
        $w.line6 \
        $w.line7 \
        $w.line2 \
        $w.status \
        -side top -exp true -fill x
}
#### END OF FILE ####
```

B.9 Speichern der Einstellungen: `Terminate.tcl`

```
#
# lib/Terminate.tcl
#
###############################################################################
proc Terminate {} {

        global agents0 agents1 lockfile pmmId sounds setup

        AudioEffect progExit NoWait

#       for {set i 80} {$i > 20} {incr i -4} {
#         tk_setPalette grey$i
#         update
#       }

#       dp_shutdown $fd both

        if $setup(saveSettings) {
          SaveSettings
        }
        if {$setup(pmmKill) && $pmmId != 0}  {
          puts "CoCo: killing pmm $pmmId..."
          exec kill $pmmId
        }

        destroy .
        catch "exec rm $lockfile"
}
###############################################################################
proc SaveArray {_a {f stdout}} {
        upvar $_a a

        set id [array startsearch a]
        while {1} {
          set el [array nextelement a $id]
          if {$el == ""} {break}
          puts $f "set $_a\($el\) \"$a($el)\""
          }
        array donesearch a $id
}
###############################################################################
proc SaveSettings {} {

        global agents0 agents1 lockfile pmmId sounds setup MUCS_SETUP_FILE

        if {![catch {set f [open $MUCS_SETUP_FILE w]} Msg]} {
          puts $f "#"
          SaveArray setup $f
          puts $f "#"
          SaveArray sounds $f
          puts $f "#"
          SaveArray agents0 $f
          puts $f "#"
          SaveArray agents1 $f
          close $f
        }
}
###############################################################################
```

B.10 Hilfsfunktionen: `Utils.tcl`

```
#
# Utils.tcl
#
###############################################################################
proc AudioEffect {which {waitOpt NoWait}} {

        global setup sounds

        if $setup(audioEffects) {
          if {$waitOpt == "NoWait"} {
            catch "eval exec $setup(audioPlayer) $sounds($which) &"
          } else {
            catch "eval exec $setup(audioPlayer) $sounds($which)"
          }
        }
}
###############################################################################
proc BindWidgetExpl {widget status msg} {

        global tk_version
```

```
        bind $widget <Enter> "if {$tk_version < \"4.0\"} {tk_butEnter %W}
          $status configure -text \"$msg\""
        bind $widget <Leave> "if {$tk_version < \"4.0\"} {tk_butLeave %W}
          $status configure -text \"\""
}
################################################################################
proc mkDialog {w bitmap msgArgs args} {
#
# usage:        mkDialog w bitmap msgArgs list list ...
#
# purpose:      Create a dialog box with a message and any number of
#               buttons at the bottom.
#
# Arguments:
#    w -        Name to use for new top-level window.
#    msgArgs -  List of arguments to use when creating the message of the
#               dialog box (e.g. text, justifcation, etc.)
#    list -     A two-element list that describes one of the buttons that
#               will appear at the bottom of the dialog.  The first element
#               gives the text to be displayed in the button and the second
#               gives the command to be invoked when the button is invoked.

        global audioPlayer sounds

        AudioEffect warning
        catch {destroy $w}
        toplevel $w -class Dialog
        wm title $w "Info"
        wm iconname $w "Dialog"
        wm positionfrom $w user

        set swidth [expr ([winfo screenwidth .] - 250 ) / 2]
        set sheight [expr ([winfo screenheight .] - 120 ) / 2]
        wm geometry $w +$swidth+$sheight

        # Create two frames in the main window. The top frame will hold the
        # message and the bottom one will hold the buttons.  Arrange them
        # one above the other, with any extra vertical space split between
        # them.

        frame $w.top -relief raised -border 1
        frame $w.bot -relief raised -border 1
        pack $w.top $w.bot -side top -fill both -expand yes

        # Create the message widget and arrange for it to be centered in the
        # top frame.

        if {$bitmap != ""} {
          label $w.top.icon -bitmap $bitmap
          pack $w.top.icon -side left -padx 8 -pady 8
        }
        eval message $w.top.msg -justify center $msgArgs
#-font -Adobe-times-medium-r-normal--*-180*
        pack $w.top.msg -side top -expand yes -padx 8 -pady 8

    # Create as many buttons as needed and arrange them from left to right
    # in the bottom frame.  Embed the left button in an additional sunken
    # frame to indicate that it is the default button, and arrange for that
    # button to be invoked as the default action for clicks and returns in
    # the dialog.

    if {[llength $args] > 0} {
        set arg [lindex $args 0]
        frame $w.bot.0 -relief sunken -border 1
        pack $w.bot.0 -side left -expand yes -padx 10 -pady 10
        button $w.bot.0.button -text [lindex $arg 0] -borderwidth 3 \
                -command "[lindex $arg 1]; destroy $w" -padx 2m -pady 1m
        pack $w.bot.0.button -expand yes -padx 6 -pady 6
        bind $w <Return> "[lindex $arg 1]; destroy $w"
        focus $w

        set i 1
        foreach arg [lrange $args 1 end] {
            button $w.bot.$i -text [lindex $arg 0] -borderwidth 3 \
                    -command "[lindex $arg 1]; destroy $w" -padx 2m -pady 1m
            pack $w.bot.$i -side left -expand yes -padx 10
            set i [expr $i+1]
        }
    }
    bind $w <Any-Enter> [list focus $w]
    focus $w
    grab set $w

}
#### END OF FILE ####
```

Anhang C

mucsp Grammatik

Die hier benutzte BNF-ähnliche Grammatik lehnt sich an die in [1] verwendete an.

```
request-packet ::=  "Call " username "MUCS/1.0"
                    [ info-field ]
                    [ origin-field ]
                    media-field
                    [ priority-field ]
                    [ face-field ]
                    [ host-field ]

response-packet ::= "MUCS/1.0 " code msg
                    [ dest-field ]
                    [ reason-field ]

code ::=            digit digit digit

msg ::=             string

dest-field ::=      "Location: " username@domainname

reason-field ::=    "Reason: " string

origin-field ::=    "Origin: " username@domainname

info-field ::=      "Info: " string

priority-field ::=  "Priority: " priority

media-field ::=     "Media: " media-type protocol format rate addr port

host-field ::=      "This-Host-Only: " { TRUE | FALSE }

face-field ::=      "Face: " xfacestring

priority ::=        "emergency" | "urgent" | "normal"

media-type ::=      "audio" | "video" | "whiteboard" | "text"

protocol ::=        see SDP description

format ::=          see SDP description
```

```
digit ::=            1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

string ::=           (ALPHA-NUMERIC)+
```

Literaturverzeichnis

[1] HANDLEY, M., AND JACOBSON, V. Sdp:session description protocol (draft). In *IETF Internet-Draft of SDP, spec 01* (July 1995), IETF.

[2] HOPPER, A. Communication at the desktop. 1253–1265.

[3] JACOBSON, V. sd, the LBL session directory. Manual page, Nov. 1992.

[4] MCCANNE, S., AND JACOBSON, V. vic: A flexible framework for packet video. In *Proc. of ACM Multimedia '95* (Nov. 1995).

[5] OUSTERHOUT, J. K. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, Massachusetts, 1994.

[6] SCHULZRINNE, H. Dynamic configuration of conferencing applications using pattern-matching multicast. In *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* (Apr. 1995), GMD Fokus.

[7] SCHULZRINNE, H. *Format of Messages Exchanged between Conferencing Tools*. GMD Fokus, Aug. 1995.

[8] SCHULZRINNE, H. *NEVOT - A network voice terminal (3.32)*. GMD Fokus, Aug. 1995.

[9] SMITH, B. C., AND ROWE, L. A. Tcl distributed programming. Tech. rep., Department of Computer Science, Cornell University, June 1995. ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/Tcl-DP/tcl-dp3.3b1.tar.gz.

[10] WANT, R., AND HOPPER, A. Active badges and personal interactive computing objects. Technical Report ORL 92-2, Olivetti Research, Cambridge, England, Feb. 1992. also in *IEEE Transactions on Consumer Electronics*, Feb. 1992.

[11] WANT, R., HOPPER, A., FALCAO, V., AND GIBBONS, J. The active badge location system. 91–102. also Olivetti Research Limited Technical Report ORL 92-1.

[12] ZAHL, C. *Entwicklung einer Internet Multimedia-Telekommunikationsanlage*. TU-Berlin, GMD Fokus, Nov. 1995.