Entwicklung einer Internet-Multimedia-Telekommunikations-Anlage

Studienarbeit von
Christian Zahl (Matrikelnr. 12 66 80)

Betreut durch Prof. Adam Wolisz (TU Berlin)
und Dr. Henning Schulzrinne (GMD Fokus)

Berlin, den 22. November 1995

Fachgebiet Telekommunikationsnetze
Institut für Offene Kommunikationsysteme
Fachbereich Informatik (FB 13)
Technische Universität Berlin

Forschungsgebiet für offene Kommunikation
GMD Fokus Berlin

Danksagungen: An dieser Stelle möchte ich all denen danken, die mich während der Erstellung dieser Arbeit unterstützt haben.

Zuerst möchte ich unseren Betreuer *Henning Schulzrinne* recht herzlich für seine Unterstützung während der Arbeit danken. Er war stets bereit die doch oft aufkommenden Fragen meinerseits sachlich und hilfreich zu beantworten. Auch die konstruktiven Rücksprachen, die im Verlauf dieser Arbeit eigentlich viel zu selten von uns initiiert wurden, waren stets hilfreich und fruchtbar.

Einen besonderen Dank gebührt drei Mitarbeitern der `IBM` in aus Austin, Texas. *Mathew Accapadi* bemühte sich mir einen kompetenten Ansprechpartner für meine Problemen mit dem Audioadapter zu nennen. Er verwies mich an *Steve Hatcher* dem Entwickler des Audioadapters, der mir bereitwillig die genauen Gründe für die Probleme auf unserer `RS/6000` erklärte. Nach Austausch einiger Mails bot er an, zwei aktuelle Adapter zu uns nach Berlin zu senden, die dann auch prompt funktionierten. Der dritte `IBM` Mitarbeiter war *Paul Wadehra*, der Entwickler des Device-Drivers für den Audioadapter. Er scheute keine Mühe meine große Menge an Fragen hinsichtlich der Programmierung des Adapters zu beantworten. So schickte er mir im Laufe der Zeit aktuelle Include-Files, sowie ein Fax mit dem schematischen Aufbau des auf dem Board verwendeten Audio-Chips `CS4231`. Als ich ihm auf zwei kleine Probleme im Treiber hinwies war er auch bereit mir zwei neue Treiber zukommen zu lassen. Zusammengenommen erstreckte sich die Korrespondenz auf über 150 Mails über einen Zeitraum von fast 3 Monaten.

Auch *Bob Olson* half ein wenig bei der Portierung von NEVOT. So war als Beta-Tester bereit einige Versionen von mir zu testen, zumal er auf einer anderen `RS/6000` arbeitete, die zudem auch über eine andere Audio Hardware verfügte.

Dave Shield half mir bei der Benutzung seiner `resparse` Library, mit der DNS-Pakete in C Strukturen umgewandelt werden können. Im Gegenzug half ich ihm im Laufe der Zeit noch einige Fehler in seiner Library zu beseitigen.

Ungenannt bleiben natürlich alle Autoren, deren Programme während der ganzen Zeit von uns verwendet wurden, angefangen von LaTeX über `xfig` bis hin zu `tcl/tk`.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Gegenwärtig hat sich ein Trend hin zu multimedialen Anwendungen entwickelt. Fast jeder heute zu kaufende PC oder Notebook verfügt bereits über ein integriertes Audio-System, mit Mikrofon, Lautsprecher, Anschlußmöglichkeit für ein CD-ROM Laufwerk etc. Die Integration von Hardware-MPEG-Dekodern auf einer Grafikkarte, lassen einen PC zu einem Multimediastudio werden. Die Hersteller von UNIX-Workstations folgen ebenfalls diesem Trend, so daß fast alle neu zu kaufenden Geräte über eine derartige Ausstattung verfügen.

Was kann man jedoch mit diesen Komponenten machen? Fast alle gegenwärtig vorhandenen Ansätze nutzen diese Komponenten nur für untergeordnete Dinge. Besitzt man beispielsweise eine SoundBlaster Audio Karte, so wird durch die Installation der zugehörigen Software die MS-Windows Umgebung akustisch untermalt. Das Öffnen eines neuen Fenster wird mit einem „drip“ angekündigt, und über einen Fehler lacht es laut aus dem Lautsprecher.

Neben diesen Spielereien existiert jedoch auch ein anderes Anwendungsgebiet, das der multimedialen Kommunikation zwischen Menschen. Im Internet wird bereits seit ca. fünf Jahren versucht, Audio- und Videokonferenzen zwischen mehreren Teilnehmern zu realisieren. Es existieren verschiedene Programme wie NEVOT, `vic`, `nv`, `wb`, etc. die mittels geeigneter Übertragungsprotokolle wie `RTP` diese Daten „sicher“ über das Internet geleiten. Auch auf dem PC-Sektor tut sich einiges, insbesondere seit dem Beginn der Öffnung des Internets für den Privatmenschen, sowie für die kommerzielle Nutzung. So entstanden auch Produkte wie InternetPhone [Ltd95], mit dem ein Gespräch mittels Modem über das Internet abgewickelt werden kann.

Alle diese Ansätze beschäftigen sich jedoch mehr mit der Frage der technischen Durchführbarkeit, als mit der des einfachen und komfortablem Gebrauchs. Bei allen derzeit verfügbaren Anwendungen müssen sich die beteiligten Gesprächspartner vorab über das zu führende Gespräch abstimmen. So müssen die Internet-Adressen der jeweiligen Rechner bekannt sein, die verwendete Audiokodierung und TCP / UDP Portnummer, etc.

In dieser Arbeit verfolgen wir einen anderen Ansatz. Wir betrachten keine Multimedia-Agenten, sondern beschäftigen uns ausschließlich mit der Frage nach einer komfortablen Benutzung. Wir stellen vor, wie wir direkt die Teilnehmer adressieren und nicht einen Rechner, wie ein „Anruf“ ankommen und behandelt werden kann, ohne daß der Angerufene zuvor Kenntnis hiervon haben muß. Weiterhin beschreiben wir eine beispielhafte Implementierung unseres Ansatzes, der die Verwendung dieser Medien ebenso einfach macht, wie das tagtägliche Telefonieren.

Das von uns realisierte und beschriebene Konzept wurde ausschließlich in Hinblick auf die Verwendung auf vernetzten UNIX Workstations betrachtet. Auf die Einbeziehung von PCs oder ähnlichem wurde hier verzichtet. Dennoch ist es möglich, auch diese mit in das Konzept einzubeziehen, jedoch entstehen hier andere und neue Probleme, als bei der Verwendung von UNIX Workstations. So stellt sich die Frage

nach den Identifizierung eines Benutzers auf einem PC. Derzeit existiert nur im OS/2 ein geeigneter Mechanismus (User Profile Management), der einen Benutzer identifiziert.

Aufbau des Textes

Kapitel 2 stellt unsere neue MUCS Architektur vor, die es uns ermöglicht, die oben angerissenen Probleme zu lösen. Da es sich um einen recht großen Themenkreis handelt, wurde die Arbeit in zwei Teile aufgeteilt, deren Abgrenzung gegen Ende des Kapitels erfolgt.

Kapitel 3 beschreibt den Aufbau des zuvor vorgestellten Conference Daemons, dem Kern der Idee. Es wird der detailierte Ablauf, sowie die von uns beispielhafte Implementierung vorgestellt.

Kapitel 4 stellt einige von uns implementierte Shell-Scripts vor, die dem Conference Deamon helfen, den Anruf zu bearbeiten.

Kapitel 5 stellt das von uns verwendete MUCS Protokoll zur Anrufsignalisierung vor.

Kapitel 6 begründet die Einführung eines Location Services, und beschreibt die von uns vorgeschlagene und beispielhafte implementierte Variante.

Kapitel 7 gehört nicht mehr zu der eigentlichen Problemlösung. Um jedoch während der Testphase einen real benutzbaren Audio-Agenten zur Verfügung zu haben, wurde im Vorfeld der Agent NEVOT auf AIX portiert. Dieses Kapitel beschreibt die wesentlichen Änderungen, die nötig waren.

Kapitel 8 bewertet abschließend die Arbeit und gibt Hinweise für weitere Entwicklungen und Verbesserungen, die sich während des Projektes gezeigt haben.

Anhang A-D enthält der Vollständigkeit halber den Quell-Code, der von uns beispielhaft implementierten Programme.

Einige Bemerkungen zur Wortwahl: Wir verwenden die Fachtermini in englischer Sprache und verzichten bewußt auf eine Übersetzung. In der Vergangenheit mußten wir sehr oft verstellen, daß das Lesen von ins Deutsche übersetzte Fachbüchern zu einer Qual wird. Konstellationen wie *...werden die Arbeitsseiten in den Kachelspeicher ausgelagert...* halten wir daher für unangebracht.

Kapitel 2

Das Design

Das folgende Kapitel gibt einen kurzen Überblick über das von uns entwickelte Design (eine detailliertere Beschreibung, mit Hinweis auf die von uns geforderten Möglichkeiten ist in [Oer95], Kapitel 3 gegeben). Aufgrund der Größe des Projektes wurde dieses in zwei Teile zerlegt. Diese Aufteilung wird am Ende des Kapitels vorgestellt.

2.1 Die beteiligten Instanzen

Als erstes wird ein Multimedia-Agent (MMA) benötigt, mit Hilfe dessen das Gespräch[1] durchgeführt werden kann. Zu beachten ist hierbei, daß die Funktion des gerichteten Anrufs unabhängig davon ist, welches Medium im nachhinein verwendet wird (Audio, Video, White-Boards, etc.). Aus diesem Grunde haben wir das gesamte System nicht auf einen oder mehreren Multimedia-Agenten aufgebaut, sondern diese völlig als eigenständige Programme betrachtet. Sie werden also nur von uns zur Übertragung der Mediendaten genutzt und für nichts anderes.

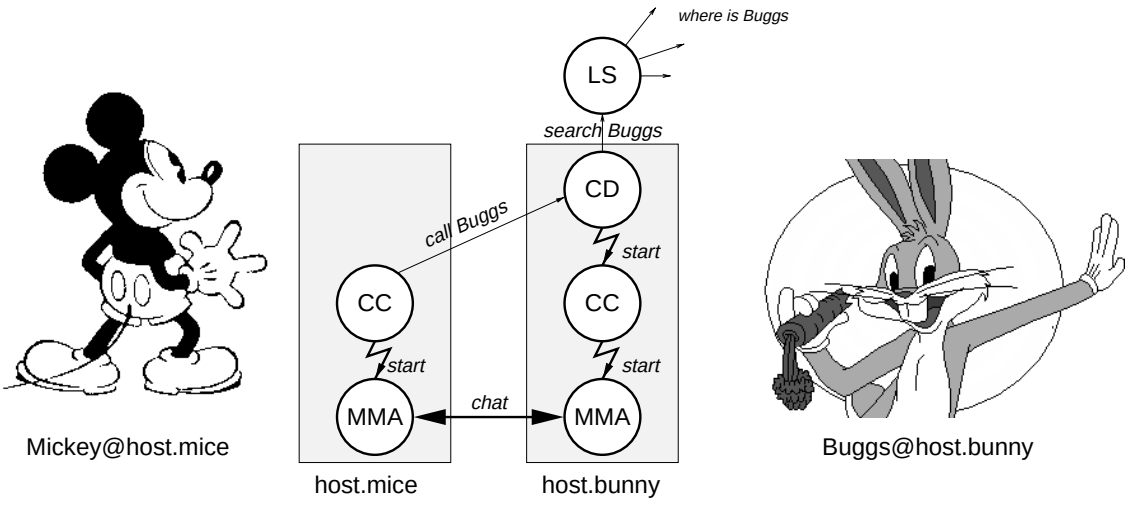

Abb. 2.1: Vereinfachte Darstellung der beteiligten Instanzen

Um einen anderen Benutzer anrufen zu können benötigen wir somit ein eigenständiges Programm, den sogenannten Conference Controller (CC). Dieser stellt die in den bisherigen Agenten fehlende Fähigkeit zur Verfügung, direkt einen speziellen Gesprächspartner anzurufen.

[1] Unter einem „Gespräch“ sind alle Formen von Multimedia-Konferenzen gemeint.

Um einen Anruf zu tätigen, wird auch eine Instanz benötigt, die einen derartigen Anruf entgegennehmen kann. Diese Aufgabe wird von dem Conference Daemon (CD) übernommen. Der CD läuft als Daemon-Prozeß auf allen Rechnern in der jeweiligen Umgebung und nimmt die Anrufe entgegen. Wurde der angerufene Benutzer identifiziert und am lokalen Rechner gefunden, so wird für ihn der Anruf durch Starten eines eigenen CC signalisiert. Der Benutzer kann mit Hilfe dieses CC auf den Anruf in verschiedenster Weise reagieren (Annahme, Ablehnung, Rufumleitung, etc.).

Ein wesentlicher Punkt bei der komfortablen Verwendung eines derartigen Systems ist die Möglichkeit, einen Benutzer und nicht einen Rechner anrufen zu können. Zu diesem Zweck muß der Benutzer in der jeweiligen Umgebung ausfindig gemacht werden, so daß ihm an dem derzeitig von ihm benutzten Rechner, der Anruf signalisiert werden kann. Zu diesem Zweck wird ein Lokalisierungsdienst (Location Service, kurz LS) benötigt, der den derzeitigen Aufenthaltsort eines Benutzter bestimmen kann. Mit Hilfe dieses LS ist der CD nun in der Lage einen Anruf für einen Benutzer zum richtigen Rechner weiterzuleiten (in unser derzeitigen Realisierung durch Informieren des Anrufers).

Die Abbildung 2.1 zeigt eine vereinfachte Darstellung der beteiligen Instanzen.

2.2 Abgrenzung der Arbeit

Aus der Beschreibung des von uns entwickelten Designs ist sicherlich ersichtlich, daß die Realisierung den Rahmen einer Studienarbeit für eine Person übersteigen würde. Aus diesem Grunde wurde das Projekt in zwei Teile aufgeteilt, wobei der eine Teil von Frank Oertel [Oer95], der andere Teil von mir behandelt wird. Zur Abgrenzung der beiden Teile sind hier die Bereiche aufgezeigt, die in dieser Studienarbeit von mir abgedeckt werden:

- Entwicklung eines Conference Daemon (CD), Beschreibung seiner Funktionalität und dessen Implementierung;
- Implementierung einige Shell-Scripte, auf die der CD zurückgreift;
- Überlegungen zum Thema „Lokalisieren eines Benutzers“ und die Implementierung eines einfachen Location-Servers;
- Portierung des Audio-Agenten NEVOT auf `AIX`.

Kapitel 3

Der Conference Daemon

In Kapitel 2 stellten wir den Conference Daemon (CD) als Zugangspunkt für einen eingehenden Anruf vor. Dieser CD übernimmt somit die Funktion einer Telefonzentrale in der herkömmlichen Welt des Telefons. Um einen „Anruf“ nun auch entsprechend bearbeiten zu können, sind die folgenden Punkte zu klären:

1. Wie wird der CD, und welcher erreicht.
2. Empfangen des Anrufes, dekodieren.
3. Überprüfen, ob der angewählte Benutzer bekannt ist, Auflösen von Aliase.
4. Der CD muß anhand globaler und lokaler Einstellungen entscheiden, ob der Anruf „durchgestellt“ werden darf.
5. Der angerufene Benutzer muß in der lokalen Umgebung gefunden werden.
6. Der Angerufene muß über den Anruf informiert werden.
7. Behandlung von Timeouts.

In den folgenden Abschnitten werden wir die oben genannten Punkte Schritt für Schritt beschreiben. Die angegebene Reihenfolge entspricht der, in der die entsprechenden Probleme im CD gelöst werden. Abbildung 3.1 zeigt den Programmfluß in unserer Implementierung des CD. Es ist hilfreich ihn parallel zu den folgenden Abschnitten zu betrachten.

3.1 Zugang zum CD

Der CD sollte auf allen Rechnern der Umgebung laufen, damit das MUCS System von außen aus über alle Rechner zu erreichen ist. Insbesondere ist es wichtig, daß der CD auf den Mail-Servern läuft. Der Grund dafür ist, daß der Angerufene durch Angabe seiner E-Mail-Adresse angesprochen werden soll und somit der erste Anruf[1] über einen der Mail-Server erfolgen wird. Ein anderer Grund ist, daß im Falle von Firewalls die Mail-Server in jedem Fall angesprochen werden können und somit MUCS auch in einer solchen Umgebung funktioniert. Somit sollte der CC in jedem Fall die Mail-Server ansprechen können, auch mehrere, wenn sie definiert sind.

Weiterhin besteht die Möglichkeit, einen oder mehrere Rechner als MUCS Server zu bestimmen und im Domain Name System (DNS) [Moc87a][Moc87b], ähnlich wie bei FTP oder WWW als Alias auf den entsprechenden Rechner einzutragen. Die dritte Möglichkeit ist die Verwendung von SRV Resource Records [GV95] im DNS, die auf MUCS Server hindeuten. Jedoch war der neue Resource Record während

[1] Wir haben einen Caching-Mechanismus entwickelt, bei dem der CC die folgenden Anrufe direkt über den Rechner abwickelt, über den die letzte Konferenz stattgefunden hat.

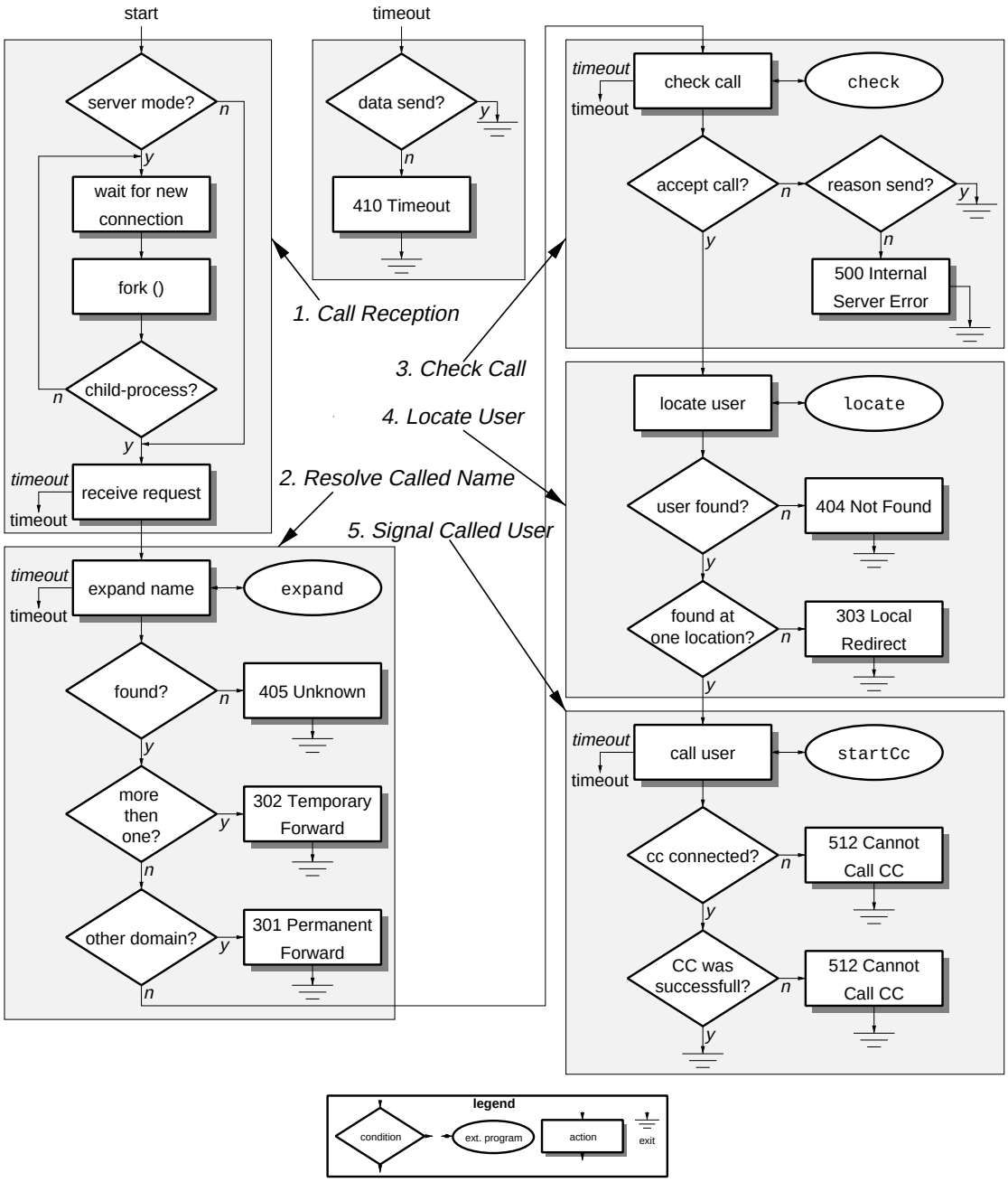

Abb. 3.1: Programmablauf des `CD`, ohne Fehlerbehandlung

der Implementationsphase noch nicht genau spezifiziert. Desweiteren hätte die Verwendung auch eine Änderung in der im allgemeinen verwendeten DNS Implementation `bind` [Vix95] zur Folge gehabt. Aus diesem Grunde haben wir diese Variante nicht weiter berücksichtigt.

Um den `CD` über eine TCP-Verbindung ansprechen zu können, muß die zu benutzende Portnummer bekannt sein. Wir schlagen die Registrierung eines neuen „Well Known Ports“ oder einer „Registered Port Number“ beim `IANA` (Internet Assigned Numbers Authority, ein Gremium zur offiziellen Vergabe von zugewiesenen Parametern, wie IP Portnummern, Multicast-Adress-Gruppen etc.) mit dem Namen `mucs` vor. Es bietet sich ebenfalls die Verwendung des `WKS` Records (Well Known Service, Zuordnung eines Servicenamen zu einer Portnummer) im `DNS` (Domain Name Service) an. Jedoch wird in RFC 1123 [Bra89] erwähnt, daß derartige Einträge in der Vergangenheit so gut wie nie gepflegt wurden, so daß es in sehr vielen Domains keine derartigen Einträge gibt. Dieser Umstand führte sogar zu einer Änderung

in der Beschreibung, wie eine Internet E-Mail [Cro82] zugestellt werden soll [Bra89]. Demzufolge soll der WKS in keinem Fall mehr angefragt und vor allem auch nicht mehr benutzt werden. Inwieweit dieser Hinweis beim MUCS beachtet werden sollte ist nicht ganz klar. Insbesondere würde mit der selben Begründung auch der SRV Record von der weiteren Benutzung ausgeschlossen werden. In der derzeitigen Implementierung haben wir auch diesen Eintrag nicht weiter berücksichtigt.

Zwar böte sich auch die Verwendung des HTTP Ports an da wir, wie in Kapitel 5 beschrieben nur eine Erweiterung dieses Protokolls vornehmen. Jedoch führt das zu erheblichen Problemen während der Einführungsphase, da alle HTTP Server dann diese Erweiterung ebenfalls verstehen müßten. Außerdem müßten wir dann MUCS bereits jetzt in HTTP einbetten, was doch den Rahmen dieser Arbeit sprengen würde! Es stellt sich ohnehin die Frage, inwieweit das Zusammenführen der beiden Protokolle sinnvoll ist, da beide ein anderes Ziel verfolgen.

3.1.1 DomainExpand

Die Bestimmung der möglichen Hosts und deren Adressen, auf denen der CD eventuell erreicht werden kann, ist nicht ganz so einfach, wie es zunächst scheint. So existieren beispielsweise keine Routinen um eine Liste der Mail-Exchange-Hosts mit ihren Adressen und Preference-Werten zu erhalten. Eine derartige Anfrage kann entweder mit Hilfe des Programms `nslookup` oder aber durch direktes Programmieren einer Anfrage an einen Nameserver beantwortet werden. Die Verwendung von `nslookup` erscheint uns eher ungeeignet, es sei denn, es geht um die Verwendung in einer Script-Sprache wie `tcl/tk`. Jedoch ist die Auswertung der Antworten von `nslookup` mittels Shell-Scripts ebenfalls nicht ganz trivial.

Aus diesen Gründen entschieden wir uns eine kleine Bibliothek zu implementieren, die für einen angegebenen Domainnamen eine sortierte Liste von zuständigen Rechnern mit deren Adressen generiert. Um diese auch im `tcl/tk` verwenden zu können, wurde außerdem noch eine neue `tcl` Extension geschrieben, die ein neues `tcl` Kommando zur Verfügung stellt (DomainExpand).

DNS-Pakete sind binär aufgebaut und besitzen einige Mechanismen zur einfachen Komprimierung der Daten. So werden wiederkehrende Teildomains als Referenz auf eine andere Domain abgelegt. Es existiert zwar die `resolver` Library, die das Dekodieren eines DNS Paketes unterstützt, jedoch ist es recht mühselig damit umzugehen. *Dave Shield* entwickelte hierfür die `resparse` Library [Shi94]. Diese wandelt empfangene DNS Pakete in eine C-Struktur um. Zwar enthielt sie noch einige kleine Fehler, die glücklicherweise sehr schnell zu einem „coredump“ führten, so daß diese Fehler von uns schnell bemerkt und behoben werden konnten.

3.2 Empfang des Anrufes

Damit der CD den Anruf erhalten und entsprechend reagieren kann, muß der CD auch auf dem jeweiligen Rechner laufen. Es stellt sich somit die Frage, wie der CD gestartet werden sollte.

3.2.1 Starten des CDs

Damit unser MUCS funktionieren kann, muß der CD auf jedem Rechner, auf dem MUCS benutzbar sein soll, gestartet werden. Dieser Vorgang sollte verständlicherweise automatisch ausgeführt werden, damit das System auch ein gewisses Maß an Stabilität besitzt. Weiterhin hilft diese Methode MUCS zu verbreiten. So kann ein Benutzer über MUCS angerufen werden, der Anruf wird ihm signalisiert, und er kann das „Gespräch“ annehmen und führen, ohne daß er jemals etwas von MUCS gehört haben muß. Dies ist vergleichbar mit dem `talk` System, bei dem zwar nicht der lokale `talk` Prozeß gestartet wird, sondern nur eine Nachricht auf dem Bildschirm erscheint.

Der CD muß in jedem Fall root-Privilegien besitzen, damit er zum einen ohne Einschränkungen auf die lokalen Einstellungen beim Angerufenen zugreifen kann, zum anderen auch dessen MUCS-Umgebung unter seiner eigenen User-Id starten kann. Desweiteren muß die MUCS- Umgebung auf das X-Display[2] des Angerufenen zugreifen können, welches in der Regel mittels X-Authorizations gegen unbefugten Zugriff geschützt sein wird. Alle diese Probleme lösen sich von selbst wenn der CD währenddessen unter der selben User-Id läuft wie die des Angerufenen. Da ein CD nicht nur einen User bedient, ist dies nur dann möglich wenn der CD mittels setuid die User-Id des Angerufenen annimmt. Dies geht wiederum nur dann, wenn der CD auch root Privilegien besitzt.

Es existieren zwei verschiedene Varianten, mit denen der CD automatisch gestartet werden kann. Zum einen könnte der CD über ein lokales „boot-startup-script“ wie /etc/rc.local oder durch den init Prozeß beim Systemstart gestartet werden. Dies würde bedeuteten, daß der CD ständig als Daemon Prozeß im Hintergrund läuft[3]. Als zweite Variante kann der CD auch sozusagen „on demand“ gestartet werden, sobald ein Anruf eingeht. Dies ist mit Hilfe des inetd-Servers möglich. Beide Varianten haben ihre Vor- und Nachteile:

	Start über init	**Start über** inetd
Start	automatisch beim Booten	automatisch bei jedem Anruf
Läuft	immer	nur wenn er benötigt wird
Verzög. bei Anruf	sofort ansprechbar	kleine Verzögerung durch Start
Last bei Anruf	kleinere	größere wegen Starten
Besser für	Server wie Mail Server	„normale“ Maschinen

Aus diesen Gründen haben wir den CD derart gestaltet, daß er über beide Methoden gestartet werden kann. Das Starten über den inetd sollte nach Möglichkeit auf allen Maschinen benutzt werden. Somit läuft der Prozeß nicht unnötigerweise auf einem Rechner, bei dem nicht ein einziger Anruf eingeht. Die einzige Ausnahme bilden die Mail-Server. Bei ihnen sollte der CD immer laufen und somit auch über den init Prozeß beim Systemstart gestartet werden. Somit ist das schnelle Reagieren auf einen Anruf sichergestellt.

Aus Sicherheitsgründen haben wir das Executable des CD mit dem setuid und setgid Bit versehen, der Besitzer sollte entweder nobody, oder ein neu eingetragener Dummy-User mit dem Namen mucs sein. In keinem Fall sollte root der Besitzer sein! Somit ist sichergestellt, daß zum einen, für den Fall daß der CD von einem nicht autorisierten Benutzer gestartet wird, nicht unerwartet Zugriff auf diverse Informationen erhält. Weiterhin kann der CD während einer Test-Phase von nicht-autorisierten Benutzern gestartet und mit eingeschränkter Funktionalität benutzt werden, ohne Gefahr zu laufen, daß durch Fehler im eigenen Bereich Daten verloren gehen.

Die dennoch benötigten root-Privilegien (gegeben wenn die real-user-id oder saved-set-user-id root sind) erhält der CD automatisch, wenn er durch den init-Prozeß gestartet wird. Wird der CD über den inetd gestartet, muß in der Konfigurationsdatei /etc/inetd.conf als User-Id root eingetragen werden.

3.2.1.1 Starten über inetd

Der inetd ist ein Server-Prozeß, der einen „listening-Socket“ für alle Services (Ports), die er bedienen soll besitzt. Wird von einem Client eine Verbindung zu einem der Services aufbaut, so startet dieser im Allgemeinen einen speziellen Server für diesen Service, gemäß den Einstellungen in

[2] Unter der Annahme, daß wir unter X11 arbeiten, was jedoch die Regel ist.

[3] Solange keine Anrufe eingehen verbraucht er jedoch keine Zeit, nur einige Ressourcen wie Speicher, Slot in der Prozeßtabelle etc.

`/etc/inetd.conf`[4]. Wird ein `CD` über diesen Weg gestartet, so ist die Verbindung vom Anrufer bereits aufgebaut und der neu gestartete `CD` kann mit diesem über die File-Deskiptoren 0 (stdin) und 1 (stdout) kommunizieren. Hat er diesen Anruf abgearbeitet, so terminiert er. Wichtig ist hierbei, daß der durch den `inetd` gestartete `CD` **keinen** eigenen listening-Socket besitzt und von daher auch nur diesen einen Anrufer bedienen kann. Für jeden Verbindungsaufbau muß demzufolge ein neuer Prozeß gestartet werden.

3.2.1.2 Start über init

Wird der `CD` über den `init` Prozeß gestartet, so läuft er wie jeder andere normale Prozeß. Er hat keine bereits aufgebauten Verbindungen, die File-Deskriptoren 0 und 1 (und auch 2) sind in der Regel mit der „console“ „verbunden“. Der so gestartete `CD` muß nun selber einen listening-Socket öffnen und bedienen. Um den Ablauf einfach und übersichtlich zu halten, teilt sich der `CD` für jede neue Verbindung. Der Child-Prozeß übernimmt dann die Bearbeitung des Anrufes, hingegen der Parent-Prozeß wieder auf neue Verbindungen wartet.

Der Child-Prozeß übernimmt exakt die selbe Aufgabe, wie der `CD`, der über den `inetd` gestartet wurde. Aus diesem Grunde ist der Child-Prozeß derart realisiert, daß er die gleiche Funktion zum Bearbeiten des Anrufes benutzt wie der durch den `inetd` gestartete `CD`. Die Kommunikation mit dem Client wird ebenfalls über die beiden File-Deskriptoren 0 und 1 realisiert.

Der `CD` muß nun nur noch unterscheiden können, wie er gestartet wurde, über den `inetd` oder als eigenständiger Server durch `init`. Der Einfachheit halber haben wir das durch ein Flag realisiert (-s), das beim Starten mit angegeben werden kann (vgl. Abb. 3.1, Punkt 1). Zum Vergleich, der `httpd`-Server [Mea95] vom NCSA der University of Illinois at Urbana-Champaign, kann ebenfalls über den `inetd` Prozeß gestartet werden und somit einen einzigen Request bearbeiten, oder derart, daß er als eigenständiger Server läuft. Dies wird jedoch nicht wie beim `CD` mittels eines Flags unterschieden, sondern durch die Konfiguration des „`ServerType`“ Eintrags („`inetd`“ oder „`standalone`“) in der Konfigurationsdatei `httpd.conf`.

3.2.2 Empfang des Anrufpakets

Nachdem der `CD` gestartet wurde empfängt er den Request des Anrufers und arbeitet ihn ab. Hierbei stellt sich die Frage, wann der `CD` mit seiner weiteren Arbeit beginnen, nachdem der die *Request-Line* (s. 5.1.1), oder den ganzen Request empfangen hat. Damit der Angerufene so schnell wie möglich informiert werden kann, sollte der `CD` bereits nach Empfang der *Request-Line* anhand der Einstellungen beim Angerufenen den Anruf weiterleiten. Jedoch scheitert dies daran, daß wir auch Attribute wie „Dringlichkeit“ etc. des Anrufes angeben können wollen, was aus Gründen der Flexibilität im `MIME` Teil des Requests angegeben wird. Der `CD` muß daher den gesamten Request empfangen, bevor er weitere Aktivitäten ausführen kann.

Auf den ersten Blick scheint dies eine unnötige Bremse zu sein. Jedoch wollen wir die weiteren Entscheidungen bzgl. des Anrufes (durchstellen, ablehnen etc.) in externe Programme verlagern, so daß der `CD` unverändert in verschiedenen Umgebungen mit verschiedenen Anforderungen eingesetzt werden kann. Aus diesem Grunde müssen diese Programme auf die Informationen im `MIME` Teil zugreifen können, um den Anruf auch bewerten zu können. Ein Performancegewinn durch frühzeitiges Starten der Programme, die ihrerseits weiterhin auf Daten vom Anrufer warten, ist daher nicht zu erwarten.

Sollte der Request keinen gültigen Aufbau besitzen, so wird der Anruf mit dem Status „`400 Bad Request`“ abgebrochen. Sollte der Request zu lang sein (derzeit maximal 16 KByte), so wird der Anruf

[4] Die Tatsache, daß der `inetd` auch selber einige Services erbringt, bzw. das es eine Unterscheidung zwischen UDP, TCP und RPC-Calls gibt, ist hierbei nicht weiter von Bedeutung.

ebenfalls abgebrochen. Als Status erhält der Anrufer „`510 Limit Exceeded`“. Der Anruf wird auch abgebrochen wenn der `CD` die angegebene `MUCS` Version (derzeit 1.0) nicht verarbeiten kann (mit dem Status „`511 Version Unsupported`“).

3.3 Identifizieren des Angerufenen, Aliase

Nachdem der vollständige Anruf-Request vom `CD` eingelesen wurde, muß dieser den Angerufenen identifizieren (Abb 3.1, Punkt 2). Der Angerufene wurde durch Angabe seiner E-Mail-Adresse, bzw. einer entsprechenden Adresse spezifiziert. Dem Anrufer sind in der Regel die internen Umsetzungen von einem kanonischen Namen zu einer umgebungsspezifischen Adresse relativ egal, so daß diese E-Mail Adresse 1:1 vom Anrufer an den `CD` gesendet wird. Es ist dessen Aufgabe, diesen Namen in einen internen User-Name[5] aufzulösen. In jedem Fall benötigen wir die User-Id des Angerufenen, um seine eigene Umgebung unter seiner User-Id starten zu können.

Aus Gründen der Flexibilität haben wir das Auflösen der Adresse in ein externes Programm, mit dem Namen `expand` verlagert. Somit kann die Methode, wie und mit Zuhilfenahme welcher Informationsquellen die Adresse aufgelöst wird, beliebig realisiert werden. Die genaue Beschreibung der von uns implementierten Version ist in Abschnitt 4.1 zu finden, der Quell-Code befindet sich in B.1.

Das Programm wird mit `root`-Privilegien ausgeführt. Somit ist sichergestellt, daß es zu allen nötigen Informationsquellen Zugang hat, auch wenn sie aufgrund lokaler Anforderungen für den „normalen“ Benutzer nicht zugänglich wären (wie z.B. in der TU-Berlin). Das Programm erhält den zu identifizierenden Namen als Parameter mit angegeben. Als Name wird der gesamte Name des Anrufers angegeben, so wie er vom Anrufer spezifiziert wurde. Er beinhaltet also auch den Domainnamen, falls dieser angegeben wurde. Somit können auch verschiedene Namensräume in der gleichen Umgebung verwaltet werden.

Das Programm und der `CD` sind über eine Pipe miteinander verbunden. Der File-Deskriptor 1 (stdout) des Programms wird mit dem schreibbaren Ende der Pipe verbunden, die Deskriptoren 0 (stdin) und 2 (stderr) sind nicht zugeordnet. Das Programm muß die Liste mit den neuen Namen nach stdout ausgeben, wobei in jeder Zeile immer nur ein Name enthalten sein darf. Der `CD` liest die Liste der neuen Namen über die Pipe ein und speichert sie zunächst intern. Das Programm muß mit einem der folgenden Exit-Codes terminieren:

Exit-Code	Bedeutung
0	Der Name konnte aufgelöst werden.
1	Keine mögliche Entsprechung gefunden.
>1	Programmfehler etc.

Nachdem das Programm terminierte, wertet der `CD` das Ergebnis wie folgt aus:

- Sollte ein Fehler aufgetreten sein (Exit-Code $>$ 1), so wird der Anruf mit dem Status „`500 Internal Server Error`“ abgebrochen.
- Konnte der Benutzer nicht identifiziert werden (Exit-Code = 1), so bricht der `CD` den Anruf mit dem Status „`405 Unknown`“ ab.
- Sollte die Liste der Namen größer sein als eins, so informiert der `CD` den Anrufer über die Auflösung der Liste mittels des Status „`302 Temporary Forward`“. Die neuen Namen sind dann in dem `MIME`-Feld `Location:` aufgeführt.
- Beinhaltet die Liste exakt einen Namen, enthält dieser einen Domainnamen und ist dieser nicht der der eigenen Domain, so wird der Anruf ebenfalls abgebrochen. Als Status wird „`301`

[5] Wir beziehen uns hierbei ausschließlich auf `UNIX` Umgebungen, in denen jeder Benutzer einen sogenannten User-Name besitzt. Dieser User-Name, bzw. die User-Id sind der Schlüssel, mit dem alle systeminternen Informationen gespeichert sind.

`Permanent Forward`“ zurückgeliefert, wobei im `MIME`-Feld `Location:` die gefundene Adresse angegeben ist. Dies ist dann sinnvoll, wenn ein Benutzer (z.B: `hgschulz`) umgezogen ist (von `att.com` nach `fokus.gmd.de`), die alte Adresse (`hgschulz@att.com`) aber für einige Zeit noch aktiv sein soll, bis sich alle an die neue (`schulzrinne@fokus.gmd.de`) gewöhnt haben. Ein Anrufer der einen derartigen Forward erhält sollte in Zukunft immer die neue Adresse verwenden!

- Anderenfalls wurde der Angerufene in der lokalen Umgebung eindeutig identifiziert. Der `CD` bearbeitet den Anruf nun weiter.

3.4 Überprüfung des Anrufes

In unserer Implementierung wird zuerst überprüft, ob der Anruf durchgestellt werden soll, bevor er in der lokalen Umgebung gesucht wird. Auf den ersten Blick scheint dies ein wenig unsinnig zu sein, denn weshalb sollten zuerst die Einstellungen für einen Anruf überprüft werden, wenn der Benutzer vielleicht nicht vor Ort ist. Jedoch besteht durch die Verwendung dieser Reihenfolge auch die Möglichkeit, Anrufe für einige Zeit automatisch umleiten zu lassen, z.B. während des Urlaubs. Dies wäre nicht möglich, wenn die Vorbedingung für die Überprüfung des Anrufes der Benutzer vor Ort sein müßte.

Zur Überprüfung des Anrufes (Abb. 3.1, Punkt 3) haben wir wiederum auf ein externes Programm zurückgegriffen, `checkCall` (vgl. refScripts:checkCall), ebenfalls um möglichst flexibel zu sein. Das Programm muß jedoch das Wissen über den vollen Anruf-Request haben, um die Entscheidungen treffen zu können. Zur Übergabe des Anruf-Requests gibt es hierbei die folgenden Möglichkeiten:

- er wird als Parameter beim Aufruf übergeben;
- er wird in exportierte Environment-Variablen abgelegt;
- er wird in einer Datei abgelegt, auf das das Programm zugreift;
- er wird dem Script über eine Pipe nach `stdin` übergeben.

Die erste Variante, Übergabe als Parameter scheidet aus, da in den meisten UNIX-Umgebungen eine obere Schranke für den Parameterraum existiert. Auch ist der Zugriff recht kompliziert zu realisieren. Die Übergabe als Environment-Variablen scheidet aus den selben Gründen aus. Das Ablegen in eine Datei bringt nur den Vorteil mit sich, daß ein einfaches Programm so mehrmals auf die Daten zugreifen kann. Jedoch kann es den Request auch selber in einer Datei ablegen, wenn er über `stdin` zum Programm „gepiped“ wird. Wir entschieden uns daher, die letzte Variante zu verwenden.

Soll der Anruf durchgestellt werden, so muß das Programm mit einem Exit-Code von 0 terminieren. Bei einem Exit-Code von größer 1 wird davon ausgegangen, daß ein Fehler aufgetreten ist, und der `CD` bricht daraufhin den Anruf mit dem Status „`500 Internal Server Error`“ ab. Terminiert das Programm mit 1, so wird der Anruf ebenfalls abgebrochen. Jedoch wird hierbei davon ausgegangen, daß das Programm eine gültige `MUCS`-Response generierte und nach `stdout` geschrieben hat. Nur wenn keine Daten vom Programm generiert wurden, sendet der `CD` den Status „`500 Internal Server Error`“.

„Weshalb soll jedoch das Programm im Falle einer Ablehnung eine vollständige `MUCS`-Response generieren müssen?“ Nur so ist sichergestellt, daß jegliche Form von „Verleugnung“ möglich ist. Soll zum Beispiel ein ungewünschter Vertreter dauerhaft abgelehnt werden, so scheint es sehr wirkungsvoll zu sein, den Status „`405 Unknown`“ zurückzuliefern. Der Vertreter muß nun annehmen, daß der Angerufenen in der Umgebung nicht mehr tätig ist und leider keine weiteren Informationen hinterlassen hat. Weiterhin können somit auch erzwungene Umleitungen ermöglicht werden. Möchte z.B. der Chef einer Firma nie direkt angerufen werden können, so könnte in diesem Fall der Status „`302 Temporary Forward`“ zurückgesendet werden, wobei als `Location:` dann seine Sekretärin angegeben wird. Es ist also möglich jede nur erdenkliche Reaktion auszuführen.

Ein Problem, das jedoch sofort auftritt ist die Frage nach der Authentizität des Anrufers. Wie kann der Angerufene erstens wissen wer angerufen hat, und zweitens sicherstellen, das es auch derjenige ist, für den er sich ausgibt? Da die beiden nur durch eine einfache IP-Verbindung miteinander gekoppelt sind, kann diese Information nur im `MIME`-Feld übertragen werden. Um die Authentizität sicherzustellen müßte ein Verfahren wie die Übermittlung eines PGP-Schlüssels oder ähnlichem verwendet werden. Wie dies gelöst wird, liegt jedoch außerhalb des Rahmens dieser Arbeit.

3.5 Finden des Angerufenen

Nachdem der Anruf nicht durch diverse Einstellungen abgelehnt wurde, muß der Angerufene in der Umgebung gefunden werden (Abb 3.1, Punkt 4). Genaugenommen wird nur ein Rechner benötigt, über den der Benutzer erreicht werden kann, um ihn einerseits den Anruf zu signalisieren, und um ihm die Möglichkeit zu geben, das Gespräch zu führen.

Zum Lokalisieren des gegenwärtigen Aufenthaltsortes existieren jedoch keine programmatischen Funktionen, die hierbei helfen können. Desweiteren soll das System durch weitere Zusatzsysteme einfach ergänzt werden können, wie z.B. durch ein Active-Badge-System [WHFG92, WH92, Hop94]. Um den `CD` nicht neu anpassen zu müssen, und um auf verschiedene Informationsdienste zurückgreifen zu können, realisierten wir diesen Punkt ebenfalls mit Hilfe eines externen Programms, mit dem Namen `locate` (vgl. Abschnitt 4.3).

Das Programm erhält beim Start einen Parameter mit angegeben, den UNIX User-Name des angerufenen Benutzers. Dieses Programm versucht nun den Benutzer, genauer gesagt einen Rechner ausfindig zu machen, an dem der Benutzer angemeldet ist, und gibt sie nach stdout aus. Dieser File-Deskriptor ist über eine Pipe mit dem `CD` verbunden, über die er die Namen einließt. Dabei muß in jeder Zeile exakt ein Rechnername enthalten sein.

Bei der Suche nach dem Benutzer können folgende Ergebnisse auftreten:

1. Der Benutzer wurde in der Umgebung nicht gefunden. Das Programm muß in diesem Fall mit einem Exit-Code von 1 terminieren. Der `CD` informiert den Anrufer mit dem Status „`404 Not Found`“ über die erfolglose Suche des Benutzer. Diese Verbindung wird dann vom `CD` abgebaut und der `CD` terminiert.
2. Der Benutzer konnte gefunden werden. Hierbei müssen die folgenden Fälle weiter unterschieden werden:

   (a) Der Benutzer wurde an mehreren Rechnern gefunden. Der Anrufer erhält nun den Status „`303 Local Redirect`“ und die Liste der Rechner, an denen der Benutzer gefunden wurde wird durch `Location:` Tags im `MIME`-Header angegeben. (Dies gilt auch dann, wenn der lokale Rechner einer derer ist, an denen sich der Benutzer gerade aufhält!) Der `CD` beendet dann den Anruf. Der Anrufer kann nun mit Hilfe dieser Liste jeden der potentiellen Rechner direkt anrufen.

   (b) Der Benutzer wurde an einem Rechner gefunden, jedoch nicht am eigenen. In diesem Fall verhält sich der `CD` so wie oben bei mehreren Rechnern.

   (c) Der Benutzer wurde nur an diesem Rechner gefunden. In diesem Fall bearbeitet der `CD` den Anruf weiter.

3. Sollte ein Fehler bei der Suche aufgetreten sein, so bricht der `CD` den Anruf mit dem Status „`500 Internal Server Error`“ ab.

3.5.1 Probleme mit zyklischen Anrufversuchen durch Rufumleitung

Bei der Betrachtung des oben beschriebenen Ablaufes zeigt sich ein kleines Problem. Sollte ein Benutzer an den Rechnern `hostA` und `hostB` angemeldet sein, und geht ein Anruf bei `hostA` ein, so wird der Anrufer auf `hostA` und `hostB` verwiesen. Er würde nun wiederum versuchen `hostA` anzusprechen, mit dem gleichen Resultat. Das selbe Ergebnis bekommt er ebenfalls zurückgeliefert, wenn er sich an `hostB` wendet. Es entsteht eine endlose Umleitung.

Um dieses zu vermeiden, wurde der `MIME`-Tag `This-Host-Only: true` von uns eingeführt. Sollte der Anrufer diesen Tag angegeben haben, so wird nur versucht, den Benutzer auf diesem Rechner zu finden. Das Programm bekommt in diesem Fall noch den optionalen Parameter `-l` angegeben. Somit reduziert sich die Liste der möglichen Fälle auf „gefunden“ oder nicht, einen Deadlock wie zuvor beschrieben kann nun nicht mehr eintreten.

Damit dieses Verfahren auch zuverlässig funktioniert, muß der `CC` eine Bedingung erfüllen. Sollte er vom `CD` einen Status „`303 Local Redirect`“ erhalten, im Gegensatz zu „`301 Permanent Forward`“ oder „`302 Temporary Forward`“, so **MUß** er bei einen daraus abgeleiteten weiteren Anrufversuch dieses Tag mit angeben, um die Umleitung zu vermeiden. Das ist auch der Grund dafür, daß es die Unterscheidung in „`303 Local Redirect`“ und „`302 Temporary Forward`“ gibt.

3.6 Signalisierung des Anrufes

Nachdem der Anruf die oben genannten Stellen erfolgreich passiert hat, wird dieser dem Angerufenen signalisiert (Abb. 3.1, Punkt 5). Dies wurde ebenfalls mit Hilfe eines externen Programms realisiert, `startCc` (beschrieben in Abschnitt 4.4), das jedoch unter den User-Id und Group-Id des angerufenen Benutzers gestartet wird (vgl. Abschnitt 3.2.1). In diesem wird der Conference Controller (CC) für den Empfänger gestartet, in der Regel derart, daß er den Anruf in geeigneter Form signalisiert (vgl. [Oer95]). Das Programm bekommt einen Parameter mit übergeben, die Portnummer eines Sockets des `CD`. Der `CD` wartet darauf, daß entweder das Programm terminiert oder daß sich der Conference Controller mittels der Portnummer beim `CD` anmeldet. Sollte das Programm terminieren, so bricht der `CD` den Anruf mit dem Status „`512 Cannot Call CC`“ ab.

Im Gegensatz zu den anderen externen Programmen, die wir zur Auswertung des Anrufes heranziehen, benutzen wir in diesem Fall keine Pipe, sondern einen Socket. Zum einen muß der `CC` bei einem Anruf ohnehin über einen Socket den entfernten `CD` erreichen. Da der Ablauf beim Anrufen und Entgegennehmen eines Anrufes sehr ähnlich ist, scheint die Verwendung eines Sockets durchaus legitim zu sein. Desweiteren soll die Möglichkeit bestehen, daß die Benutzer einen eigenen `CC` erstellen. Unserer Meinung nach ist es sinnvoller von vornherein mit einem Socket zu arbeiten. Hinzu kommt, daß die Verbindung zwischen dem anrufenden und dem angerufenen `CC` bidirektional ist (auch wenn in der derzeitigen Implementation hiervon noch kein Gebrauch gemacht wird), wozu zwei Pipes benötigt werden würden.

Nachdem sich der `CC` beim `CD` angemeldet hat, sendet der `CD` den vom Anrufer empfangenen Anruf 1:1 (den vollständigen Anruf-Request, angefangen von der Request-Line, bis hin zur abschließenden Leerzeile, vgl. auch Abschnitt 5.1.1) an den `CC` weiter. Dieser kann nun anhand der lokalen Einstellungen oder den Aktionen des Benutzers auf diesen reagieren. Alle Informationen, die der `CC` zum Anrufer zurücksenden möchte, werden zunächst an den `CD` über die Socket Verbindung geschickt. Dieser sendet die Daten dann transparent zum Anrufer zurück.

Der Grund für dieses „proxiing“ der Verbindung ist der, daß wir eine größere Kontrolle über den `CC` besitzen wollten und ihm deshalb die Verbindung vom Anrufer nicht übergeben wollten. Es entstünde dann ohnehin das Problem, daß der `CD` den Anruf Request zuerst an den `CC` senden müßte, wozu ebenfalls eine eigene Verbindung zwischen den Prozessen nötig wäre.

Auch hier können mehrere Situationen entstehen, die wie folgt vom CD behandelt werden:

- Sollte der CC die Socket-Verbindung schließen ohne zuvor irgendwelche Daten zum Anrufer zurückgesendet zu haben (es wird nicht überprüft, ob es sich um ein korrektes Response-Paket handelt, vgl. Abschnitt 5.1.2), so beendet der CD den Anruf mit dem Status „`512 Cannot Call CC`“.
- Sollte der CC terminieren und bereits Daten gesendet haben, so beendet der CD die Verbindung zum Anrufer ohne weitere Informationen zum Anrufer zu senden (Einhaltung des Formates des MUCS-Protokolls). In diesem Fall geht der CD davon aus, daß der Anruf vollständig abgewickelt wurde.
- Sollte der CC terminieren und sich zuvor gar nicht beim CD angemeldet haben, so beendet der CD den Anruf ebenfalls mit dem Status „`512 Cannot Call CC`“.

3.7 Behandlung von Timeouts

Es ist leicht einzusehen, daß während der Anrufphase eine Reihe von Ereignissen eintreten können. So sollte z.B. der Anruf beim Angerufenen nicht unendlich lange signalisiert werden, was bei der Verwendung akustischer Verfahren obendrein Anrufende stören würde. Zum anderen ist zu berücksichtigen, daß bei der Überprüfung des Anrufes ein lokales Programm vom Angerufenen ausgeführt wird. Sollte dieses fehlerhaft sein und nicht terminieren, würde der CD unendlich lange auf eine Antwort warten.

Aus diesen Gründen entschieden wir uns einen Timeout für einen Anruf zu implementieren. Sobald zum CD eine neue Verbindung aufgebaut wird, startet dieser einen Timer. Sobald der Timer abgelaufen ist (der derzeitiger Default beträgt 3 Minuten), beendet der CD die Verbindung. Je nachdem in welchen Stadium sich der Anruf befindet wird dies dem Anrufer mit dem Status „`410 Timeout`“ mitgeteilt. Somit kann der Anruf, angefangen vom Zugang zum CD, über das Identifizieren und Auffinden des Angerufenen, über das Signalisieren des Anrufes durch Starten der lokalen MUCS-Umgebung, bis hin zur Annahme des Anrufes, diese Zeit nicht überschreiten.

Durch die Verwendung des Timeouts ergeben sich eine Reihe von Vorteilen, die zur Robustheit des Systems maßgeblich beitragen:

- Falls der vom Anrufer verwendete Client auf einen internen Fehler stößt und sich nicht mehr ordnungsgemäß verhält, wird die TCP-Verbindung nach dem Timeout durch den CD beendet. Der CD (genauer die für diesen Anruf zuständige Instanz des CD) terminiert ebenfalls. Es bleiben also keine Prozeßleichen zurück, auch nicht nach längerer Benutzung.
- Sollte bei einem der Programme (zum Auflösen des Angerufenen, zur Überprüfung des Anrufes, zum Auffinden des Angerufenen in der lokalen Umgebung, oder zum Starten des lokalen Umgebung) ein Fehler auftreten, so wird dieser in jedem Fall nach dem Timeout erkannt werden.
- Dem Angerufenen wird der Anruf durch Starten seiner lokal eingestellten MUCS Umgebung signalisiert. Sollte diese nicht richtig konfiguriert sein und sich „aufhängen“, so stellt der CD dies ebenfalls anhand des Timeouts fest. Der Anruf wird dann abgebrochen, dem Anrufer wird mitgeteilt daß es sich um ein Problem beim Angerufenen handelt und kann ihm über andere Wege auf das Problem aufmerksam machen.
- Es kann durchaus passieren, daß der Angerufene gerade keine Zeit hat auf den Anruf zu reagieren oder kurzzeitig abwesend ist. In diesem Fall wäre es sehr lästig, wenn ein hartnäckiger Anrufer diesen Anruf einfach nicht beendet und es eine Stunde lang „klingeln“ läßt. Durch den Timeout wird dieses ebenfalls wirkungsvoll verhindert. Dies entspricht auch dem Verhalten einiger Telefonanlagen und Telefone (wie einigen Geräten von Siemens, die nach 30 Sekunden den Anruf beenden).

Kapitel 4

Beschreibung der Shell-Scripts

Zur Erinnerung, der `CD` muß mit `root`-Privilegien laufen um in jedem Fall an die lokalen Einstellungen der Benutzer zu gelangen, und zum anderen um die Möglichkeit zu haben, eine X11-Applikation zu starten. Letzteres setzt den Zugang zu den X-Authorizations voraus und da es nicht akzeptabel sein kann, daß die Benutzer diese immer abgeschaltet haben, muß der `CD` auf diese Zugriff haben.

Damit ergibt sich zwangsweise die Frage nach der Sicherheit. Der `CD` muß von daher derart gestaltet sein, daß keine Sicherheitslücken vorhanden sind. Dies führte unter anderem auch zu der Entscheidung, den `CD` vollständig in C zu programmieren, und auf die Verwendung von tcl/tk [Ous94] zu verzichten. Somit ist die Akzeptanz bei den Sysops am ehesten gegeben.

Weiterhin wurde darauf geachtet den `CD` so einfach und kurz wie möglich zu halten. Aus diesem Grund entschieden wir uns, alle weiteren umgebungsspezifischen und konfigurierbaren Teile in externe Programme zu verlagern (in unserem Fall haben wir hierzu auf Shell-Scripts zurückgegriffen, da dies zum ersten am schnellsten und einfachsten zu realisieren war, und zum anderen für die erste Testphase völlig ausreichend erscheint, auch wenn diese weitaus langsamer sind, als entsprechende Programme in C). Hierdurch steigt die Flexibilität des `CD` in hohem Maße, und es nunmehr möglich den CD mehr oder weniger zu „verifizieren“. Immerhin stellt der CD die Basis für das ganze System dar, so daß die Ausfallsicherheit im besonderen Maße berücksichtigt werden muß.

In den folgenden Abschnitten stellen wir die von uns implementierten Shell-Scripts und deren genauen Ablauf vor. Sie dienen unter anderem als Templates, so daß sie später an die jeweilige Umgebung angepaßt werden können. Die Reihenfolge entspricht der, in der sie durch den `CD` gestartet werden (vgl Abb. 3.1).

4.1 Identifizierung des Angerufenen

Als erstes betrachten wir das Script, das die Auflösung des Namens des „angerufenen“ Benutzer (oder Benutzergruppe), im folgenden *E-Mail Adresse* genannt, übernimmt. Zur Auflösung können eine Reihe von verschiedenen Informationsquellen herangezogen werden, z.B:

- die Aliase von einem Mail-System (z.B. `/etc/aliases`);
- die UNIX-User-Einträge, angegeben `/etc/passwd`;
- verschiedene lokal vorhandene Datenbanken;

Unser Shell-Script versucht die folgenden Informationsquellen in der angegebenen Reihenfolge zur Auflösung der E-Mail Adresse heranzuziehen. Die erste Quelle, in der Informationen zu dieser Adresse zu finden sind wird verwendet, die restlichen werden nicht weiter berücksichtigt. Die Ergebnisse der

Auflösung werden nach `stdout` geschrieben, damit der `CD` diese für die weitere Verarbeitung einlesen kann. Zu beachten ist, das als Ergebnis unter Umständen eine Liste mit Namen entsteht, und nicht nur einer. Wir verzichteten bei unserer Implementation aus Gründen der Einfachheit darauf, mehrere Namensräume anhand des Domainnamen zu unterscheiden:

1. Zuerst wird ein `MUCS` lokales Alias-File hinsichtlich der Adresse durchsucht. Hiermit wird es möglich vorhandene Aliase im Mail-System, sowie dedizierte Benutzer „umzumappen“. Durch Angabe einer leeren Liste können somit explizit Adressen verboten werden wie Studenten, große Listen oder „Dummy“-Benutzer wie `ftp` oder `nobody`.
2. Als nächstes wird das lokale Alias-File `/etc/aliases` zur Auflösung herangezogen. Dies ist in der Regel nur dann sinnvoll, wenn auf verschiedenen Rechnern verschiedene Alias-Listen existieren, was jedoch in der Regel nicht zu empfehlen ist. Das Ergebnis des Anrufes wäre in einem solchen Falle abhängig von dem erreichten Rechner.
3. In Anschluß daran wird das lokale Password-File wie folgt untersucht:
   (a) Der angegebene Name wird mit den Eintrag des User-Namen[1] exakt, mit Ausnahme von Groß-Klein-Schreibung, verglichen. Somit kann der Anrufer auch den User-Name verwenden („hgs“ anstellt von „Schulzrinne“). Bei dieser Variante kann maximal ein Benutzer als Angerufener als Ergebnis ermittelt werden, da die User-Names innerhalb der Umgebung immer eindeutig sein müssen!
   (b) Wortweiser Vergleich mit dem Personal-Information Feld (dieses Feld enthält eine genauere Beschreibung des Benutzers, wie „Christian Zahl“). Hat der Anrufer den exakten Namen vergessen, und erinnert sich nur noch an den Nachnamen wie „Müller“, so erhält er als Ergebnis eine Forward-Liste zurückgeliefert, in der alle „Müllers“ aufgeführt sind. Er kann nun seine Auswahl genauer spezifizieren.
   (c) Sollte der Name einen oder mehrere Punkte enthalten so wird angenommen, daß diese als Trennzeichen zwischen Namensteilen zu verstehen sind. Das Personal-Information Feld wird daraufhin für jeden Namenteil (ohne den Punkt) durchsucht. Sollte das Feld zu jedem der Namensteile eine Entsprechung gefunden haben (ungeachtet der Reihenfolge), so wird dieser Eintrag ausgewählt. Somit ist es möglich, den Angerufenen genauer zu spezifizieren, wie z.B. „R.Stevens“ anstelle von nur „Stevens“. Ein Eintrag wie z.B. „Doctor W. Richard Stevens“ würde dem Namen somit entsprechen.
4. Als letztes wird über `NIS` das domainglobale Password-File untersucht. Hierbei wird genauso vorgegangen, wie beim lokalen Password-File.

Durch einfache Konfigurationsänderungen in dem Shell-Script kann jede der oben angegebenen Informationsquellen ausgeschaltet werden, je nach den Anforderungen der jeweiligen Umgebung. Ebenso kann die Reihenfolge, in der diese Quellen befragt werden variiert werden.

Es gibt es noch weitere Quellen, deren Benutzung wir jedoch nicht implementiert haben. So könnte das Mail-System zur Auflösung der Adresse herangezogen werden. Es würden so Mailing-Listen und Aliase für Benutzer aufgelöst werden, mit exakt dessen Fähigkeiten. Dies kann durch Senden des `SMTP` Kommandos `EXPN`, bzw. `VRFY` ausgelöst werden. Leider führte der erste Versuch, etwas derartiges in der `GMD` zu benutzen in eine Sackgasse. So wird zum Beispiel der UNIX-Benutzer `hgs` im Mail-System als `schulzrinne` geführt. Der Versuch, den Namen `schulzrinne` auflösen zu lassen endete wiederum mit dem Ergebnis `schulzrinne`. Noch schlimmer war der Versuch `hgs` aufzulösen zu lassen. Hier kam kurioser Weise ebenfalls `schulzrinne` als Ergebnis heraus. Der Benutzer würde demzufolge nie erreichbar sein. Es bliebe somit nur die Möglichkeit, Mailing-Lists auflösen zu lassen.

Desweiteren ist es in einem „normalen“ Shell-Script nicht möglich, eine IP-Verbindung zum `SMTP` aufzubauen. Dies ginge nur durch die Verwendung von tcl-dp [SR95]. Jedoch ist das Executable der Shell

[1] Es ist der UNI* User-Name, nicht das Feld „Personal Informations“ gemeint!

`dptcl` ein symbolic Link auf die X-Windows Implementierung `dpwish`, so daß wir aus Sicherheitsgründen dieses nicht verwenden wollten.

4.2 Überprüfung des Anrufes

Nachdem der Benutzer identifiziert wurde, soll der eingehende Anruf „überprüft“ werden. Da wir dem Benutzer die größtmögliche Flexibilität bieten wollen, wird der Anruf hinsichtlich einiger Parameter überprüft werden können, wie:

- Maximale Anzahl gleichzeitiger Gespräche;
- Verleugnungsfunktion, wenn die Ehefrau anruft;
- Beachtung der Mittagspause;
- ...

Bei der Betrachtung der oben aufgezählten Punkte wird sehr schnell deutlich, daß sich diese beliebig ausbauen lassen. So könnte auch die folgende Forderung bestehen: „Der Anruf soll nur dann durchgestellt werden, wenn es nach 16:00 ist, ich alleine im Zimmer bin, keine anderen Gespräche führe und der Anrufer sich als Fräulein Feinbein ausgewiesen hat“. Da derartige Verknüpfungen nicht durch Tabelleneinträge oder Listen in trivialer Art und Weise realisiert werden können, entschieden wir uns wiederum ein Shell-Script zur Lösung des Problems heranzuziehen. Somit kann jeder Benutzer jede beliebig komplexe Form von Bedingung innerhalb des Scripts formulieren, zugeschnitten auf seine persönlichen Bedürfnisse, und ist nicht an starre Vorgaben durch uns gebunden.

Zweifelsohne ist nicht jeder in der Lage solche Scripts zu erzeugen, jedoch sollte es durchaus möglich sein, eine Art Generator mit graphischer Oberfläche zu entwickeln, mit dessen Hilfe derartig komplexe Bedingungen schnell und einfach formuliert werden können.

Aus zeitlichen Gründen haben wir nur ein sehr simples Script geschrieben, das immer mit 0 terminiert (Annahme des Anrufs). Die Zeit reichte nicht, um ein vernünftiges zu implementieren, das auch als Vorlage für weitere Scripts genommen werden könnte.

4.3 Auffinden des Angerufenen

Nachdem der Anruf nicht durch etwaige Einstellungen abgebrochen wurde, muß der Angerufene in der lokalen Umgebung gefunden werden. Zu diesem Zweck stehen in einer UNIX Umgebung nicht sehr viele Informationsquellen direkt zur Verfügung. Die einzige direkte Information die existiert ist die, ob ein Benutzer an dem lokalen Rechner angemeldet ist und seit wann und wie (über eine asynchrone Terminalleitung, mittels `telnet` oder `rlogin`, oder ob er direkt an der Konsole des jeweiligen Rechners sitzt).

Informationen über andere Rechner lassen sich nur mit Hilfe von Zusatzprogrammen erfragen (wie `rwho` und `rusers`). Diese bringen jedoch eine Reihe von Nachteilen (vgl. Abschnitt 6.1.1) mit sich, was zu der Einführung unseres *Location-Servers* (`LS`) führte (s. Kapitel 6). Um jedoch auch in Umgebungen, in denen der von uns vorgeschlagene `LS` nicht oder noch nicht eingesetzt wird unser `MUCS`-System einfach einführen zu können, war es nötig diese „alten“ Informationsdienste ebenfalls mit benutzen zu können, falls sie installiert sind. Somit bot sich zwangsläufig die Verwendung eines weiteren Shell-Scripts (`locate`) für das Auffinden des angerufenen Benutzers an.

Das Script erhält einen Parameter, den Namen des zu suchenden Benutzers. Wurde der Benutzer in der lokalen Umgebung gefunden, terminiert des Script mit einem Exit-Code von 0. Die Liste mit den Rechnernamen, an denen sich der Benutzer derzeit aufhalten könnte, wird nach stdout geschrieben. Sollte

der Benutzer nicht gefunden worden sein, so terminiert das Script mit 1. Im Falle eines Fehlers liefert es einen Wert größer als 1.

Unser Shell-Script versucht den angerufenen Benutzer mittels der folgenden Informationsdienste innerhalb der lokalen Umgebung ausfindig zu machen. Jede der Informationsquellen kann hierbei durch einfache Konfiguration von der Benutzung ausgeschlossen werden:

- Zuerst wird die Datenbank des Location-Servers nach Informationen über den angerufenen Benutzer durchsucht. Hierbei werden nur die Datensätze berücksichtigt, die angeben auf welchen Rechnern er angemeldet ist (über ein ASCII-Terminal, über die Konsole oder über eine Netzverbindung wie `telnet` oder `rlogin`.
- Sollte die Suche in der Datenbank des `LS` nicht erfolgreich gewesen sein, so wird die `rwho` Datenbank nach Informationen abgesucht. Auf die Verwendung von `rusers` haben wir hierbei verzichtet. Der Grund hierfür ist der, daß die Behandlung der sich ergebenen Probleme mit dem Warten auf die Antwort (vgl.6.1.2) in einem Shell-Script nicht befriedigend gelöst werden kann.
- Ansonsten wird nur die lokale `who` Datenbank nach dem Benutzer durchsucht. Somit ist auch sichergestellt, daß im Falle eines Ausfalls des `LS` oder der Datenbank des `LS`, die Benutzer immer noch erreichbar sind, sofern sie direkt über den Rechner angerufen werden, an dem sie sich gerade befinden. Dieses sollte aufgrund des von uns vorgeschlagenen Cachings der Rechnernamen, über die der Benutzer zuletzt erreicht wurde, mit großer Wahrscheinlichkeit auch zum Erfolg führen.

Damit auch die Möglichkeit besteht nachzufragen, ob ein Benutzer an dem lokalen Rechner angemeldet ist, erhält das Script einen optionalen Parameter `-l`. Wurde dieser angegeben, so werden nur Informationen über den lokalen Rechner ausgewertet. Hierbei werden jedoch nur Datenbanken des Location-Server und von `rwho` ausgewertet.

4.4 Starten der lokalen MUCS Umgebung

Hat der `CD` entschieden den Anruf zum Benutzer „durchzustellen“, so geschied dies durch Starten der `MUCS` Umgebung, genauer des Conference Controllers (`CC`) beim Angerufenen. Hierzu wird das Script „`startCc`“ vom `CD` aus gestartet. Zu beachten ist dabei, daß es unter der User-Id und Group-Id des Angerufenen von seinem Home-Directory aus ausgeführt wird, und daß die File-Deskriptoren 0 (stdin), 1 (stdout) und 2 (stderr) mit `/dev/null` verbunden sind. Aus diesem Grund werden zuerst stdout und stderr in ein lokales Logfile („`~/.mucs-log`“) umgeleitet.

Als Parameter wird dem Script exakt ein Parameter vom `CD` übergeben, die Portnummer, mit Hilfe derer der `CC` beim `CD` Kontakt aufnimmt. Das Script hat nun die Aufgabe den `CC` für den Benutzer zu starten. Der Benutzer sollte jedoch auch die Möglichkeit haben, selber zu definieren welchen `CC` und sonstige Programme beim Anruf gestartet werden sollen. Dies wird dadurch ermöglicht, daß das Script zuerst nach einem benutzerlokalen Script mit dem Namen „`~/.mucs-startCc`“ sucht. Existiert ein solches und ist dieses ausführbar, so wird es gestartet und erhält ebenfalls die Portnummer als einigen Parameter mit übergeben. Dieses Script muß mit einem Exit-Code von ungleich `0` terminieren, wenn es auf einen Fehler stößt. In diesem Fall wird eine E-Mail an den (angerufenen) Benutzer gesendet mit dem Hinweis, daß scheinbar irgendein Fehler in seiner `MUCS` Installation vorliegt. Nur anhand dieser E-Mail kann der Benutzer in der Regel über eine Fehlkonfiguration informiert werden, da er den Anruf dann selber nicht bemerken kann.

Im Falle das ein Fehler aufgetreten ist, oder der Benutzer kein lokales Startup-Script besitzt, wird versucht eine Default-Umgebung zu starten. Sollte dieses ebenfalls nicht erfolgreich sein, so erhält, je nach Konfiguration, der angerufenen Benutzer oder ein `MUCS`-Administrator eine E-Mail mit weiteren Informationen.

Bei dem Starten des `CC` für den angerufenen Benutzer auf seinem X11-Display muß noch ein Problem berücksichtigt werden. So müßte durch geeignete Methoden das richtige Display ausgewählt werden. Das Problem wird noch größer, wenn der Benutzer mehrere Screens besitzt oder gar mehrere X11-Server auf einem Rechner gestartet hat[2]. Wir haben dieses Problem derzeit außer acht gelassen, und starten den `CC` fest für das Display „`:0.0`“, was in den meisten Fällen stimmen dürfte.

[2] Dies ist durchaus möglich, z.B. unter dem HFT (High Function Terminal) von `AIX`.

Kapitel 5

Das MUCS Protokoll

Bei der Entwicklung unseres MUCS' stellte sich die Frage nach dem zu verwendenden Protokoll zwischen dem Anrufer und dem CD. Zum einen sollte es relativ einfach sein, zum anderen wollten wir nicht „das Rad neu erfinden“ und nach Möglichkeit ein bereits Vorhandenes benutzen. Im Internet Multimedia Bereich wird bereits seit einigen Jahren der Vorläufer das „Session Description Protocols“ SDP [HJ95] verwendet, um Konferenzen im vorhinein anzukündigen. Die Verwendung von SDP böte sich somit an, jedoch waren wir der Meinung, daß dieses aus mehreren Gründen für MUCS eher ungeeignet ist.

- Bei SDP handelt es sich um ein Protokoll, daß zur Ankündigung von Konferenzen im Vorhinein benutzt wird. Es ist nicht zum direkten Signalisieren geeignet.
- Es ist unidirektional.
- Es basiert auf dem Austausch von UDP Paketen[1]. Wir benötigen jedoch eine gesicherte bidirektionale Verbindung.
- Sehr viele Features, die wir einbauen wollten oder zumindest angedacht haben, sind im SDP nur über die a= Attribute realisierbar, was eher wie eine „Krücke“ wirken würde.

Zusammenfassend kamen wir zu dem Schluß daß das SDP nur sehr eingeschränkt direkt benutzbar sei und da es keinen speziellen Grund gab, unbedingt dieses Protokoll zu benutzen (ein SDP Client würde unsere Anrufe ohnehin nicht vollständig darstellen können), verwarfen wir diesen Ansatz.

Ein großer Kritikpunkt beim SDP ist die Tatsache, daß es kaum um neue Features erweiterbar ist. Da wir uns bei den eventuell in der Zukunft noch aufkommenden Wünschen bzgl. weiterer Features nicht einschränken wollten, schied auch eine etwas geänderte SDP Variante aus. Statt dessen entschieden wir uns für eine MIME [BF93] ähnliche Darstellung aller Parameter, wie Übertragungsmedien, Dringlichkeit etc. Allerdings fehlt hier noch eine geeignete Signalisierungsmethode, die wir schließlich im HTTP [BLFF95] gefunden haben. Die Tatsache, daß bei HTTP die Einkapselung weiterer Informationen ohnehin durch MIME Messages vorgegeben ist, war dabei ebenfalls sehr hilfreich. Es ergeben sich somit folgende Vorteile:

- MIME Messages sind beliebig erweiterbar und unterliegen nicht den Beschränkungen wie in SDP.
- Keine Beschränkung auf 1 kByte, wie beim SDP.

Das von uns verwendete Protokoll MUCS[2] wird getrennt beschrieben, zum einen in den folgenden Abschnitten, zum anderen in der Arbeit von *Frank Oertel* [Oer95].

[1] Laut Internet-Draft ist dies nicht ganz richtig, da die SDP Spezifikation nichts darüber sagt.

[2] Wir verwenden den Namen MUCS, da es sich um eine Erweiterung bzw. Änderung des bestehenden HTTP Protokolls handelt.

5.1 Paketformat

Wie im HTTP gibt es beim MUCS zwei verschiedene Arten von Paketen, die zwischen den beteiligten Prozessen ausgetauscht werden, *Request* und *Response* Pakete (vgl. Abbildung 5.1). Ein Request Paket wird vom anrufenden CC an einen CD gesendet (1), der den Header an den lokalen CC weitersendet (2). Das Response Paket wird vom CD an den anrufenden CC gesendet (4) und enthält u.U. den Header, den der lokale CC in Folge des Anrufes generiert hat (3).

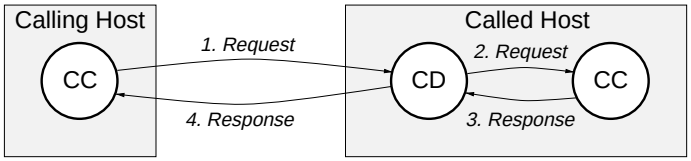

Abb. 5.1: Ausgetauschte MUCS Nachrichten beim Anruf

Für das weitere Verständnis setzen wir das Wissen des HTTP Protokolls voraus. Das gesamte Protokoll ist zeilenorientiert und jede Zeile wird mit der Folge CRLF abgeschlossen[3].

5.1.1 Request Paket

Request-Line
*(General-Header | Request-Header | Entity-Header)
CRLF
[Entity-Body]

Abb. 5.2: Request

Wie im HTTP besteht ein Full-Request-Paket (Abb. 5.2) aus einer RequestLine, einer beliebigen Folge aus General-Header, Request-Header und Entity-Header, der trennenden Leerzeile, sowie dem optionalen Entity-Body. Im Gegensatz zum HTTP verzichten wir auf den Simple-Request, der nur aus Kompatibilitätsgründen zu der Vorgängerversion erhalten blieb. Beim MUCS ist der *Entity-Body* in jedem Fall leer, da alle Daten die für den Anruf wichtig sind, in den Headern enthalten sind. Ein MUCS Paket wird somit durch die Leerzeile abgeschlossen, so daß der CD ohne den Inhalt des Paketes analysieren zu müssen das Paket vollständig empfangen kann. Die Bedeutung der drei Header ist in [Oer95] beschrieben.

Die Request-Line besitzt folgenden Aufbau:

RequestLine ::= *Method Request-URI Version*

Die drei Felder werden durch Whitespaces (vgl. Glossar) voneinander getrennt. In *Method* ist der auszuführende Request angegeben, wobei *Request-URI* das angesprochene Objekt beschreibt. Wir haben für MUCS eine weitere *Method* definiert, die dementsprechend als *extension-method* zu bezeichnen ist:

extension-method ::= CALL

Der anzurufende Benutzer wird durch den Request-URI spezifiziert. Wir verwenden hierzu nicht exakt den in RFC 1738 [BLMM94] angegebenen, vollständigen Mail-URL:

Request-URI ::= mailto:RFC822-Email-Address,

sondern nur die RFC822-E-Mail-Address. Die Verwendung des vollständigen URI erschien uns unsinnig. Eine genaue Beschreibung des Aufbaus einer E-Mail Adresse ist im RFC 822 [Cro82] enthalten.

[3] Analog zu HTTP.

Analog zu HTTP besteht das Versions-Feld aus dem Protokoll gefolgt durch die Versionsnummer, wobei beide durch einen Slash voneinander getrennt sind:

Version ::= MUCS/1.0

5.1.2 Response Paket

Wie schon beim Request Paket beschränken wir uns auch hierbei auf eine Full-Response (Abb. 5.3) und lassen den Entity-Body leer. Die *Status-Line* dient zur Übermittlung des Ergebnisses des Anrufes. Derzeit sind wir jedoch dabei in Bezug auf diese das Protokoll ein wenig zu ändern. So sollte die „*Status-Line*“ aus einer oder mehreren *Status-Lines* bestehen können, wobei die letzte keine *Informational* sein darf, und alle vorherigen *Informational* sein müssen. In dieser Hinsicht unterscheidet sich dieses Protokoll von dem eigentlichen HTTP, bei dem immer nur eine *Status-Line* existieren darf. Wir halten es jedoch für sinnvoller mehrere *Informational Status-Lines* übertragen zu dürfen, um somit den jeweils aktuellen Zustand des CD mitzuteilen. Da sich beim MUCS ein Anruf durchaus über mehrere Minuten hinziehen kann, ist dies sehr von Vorteil. Somit kann der Anrufer wenigstens sehen was gerade passiert und entscheiden, wann er den Anruf abbrechen möchte. Eventuell könnte man dieses zur Diskussion in die HTTP Working Group Mailing List stellen.

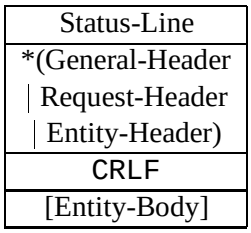

Abb. 5.3: Response

5.1.2.1 Status-Lines

Wie in HTTP beschrieben besteht eine Statusline aus den folgenden Feldern:

Status-Line ::= *Version Status-Code Reason-Phrase* CRLF

Das Feld *Version* besitzt den gleichen Aufbau wie beim Request-Paket (s. 5.1.1). Sollen mehre Status-Lines zurückgesendet werden, so müssen alle das gleiche Version Feld besitzen. Das Feld *Status-Code* besteht aus exakt drei ASCII Ziffern, wobei die erste Ziffer die Klasse des Status beschreibt. Das Feld *Reason-Phrase* ist ein beliebiger ASCII String, der für den Benutzer bestimmt ist und den Status genauer erklärt. Derzeit haben wir die folgenden *Status-Codes* mit dazugehörigen *Reason-Phrases* definiert, angelehnt an die des HTTP:

1xx *Informational*
: Übermittlung von Debugging Informationen

202 **Accepted**
: Der Anruf wurde vom Benutzer angenommen.

301 **Permanent Forward**
: Der Benutzer wird in der lokalen Umgebung nicht länger geführt, kann jedoch in Zukunft über die in Location: angegebene Adresse erreicht werden. Die Idee ist, daß der Anrufer sich die neue Adresse merken und in Zukunft verwenden sollte, oder daß der CC die Adresse transparent durch die neue ersetzt.

302 **Temporary Forward**
: Es handelt sich entweder um eine Mailing-List, oder der Angerufene hat den Anruf zu einer anderen Adresse weitergeleitet. Die neue(n) Adresse(n) sind in den Location: Feldern angegeben.

`303` **Local Redirect**
Der Angerufene „sitzt“ nicht an dem Rechner, an dem der Anruf einging. Eine Liste mit möglichen Orten, wo der Angerufene zu finden ist ist in den `Location:` Feldern enthalten. Der `CC` sollte hierbei transparent für den Anrufer diese neuen Adressen anrufen.

`400` **Bad Request**
Der übermittelte Anruf hatte ein falsches Format.

`404` **Not Found**
Der anzurufende Benutzer konnte derzeit nicht gefunden werden.

`405` **Unknown**
Der anzurufende Benutzer ist in der Domain nicht bekannt.

`410` **Timeout**
Der `CD` hat in der vorgeschriebenen Zeit keine Verbindung vermitteln können.

`430` **Reject**
Der Anrufer hat den Anruf abgelehnt, die Begründung ist im „`Reason:`“ Feld enthalten.

`500` **Internal Server Error**
Der `CD` ist auf einen internen Fehler gestoßen, wie z.B. Fehler beim fork oder malloc.

`501` **Not Implemented**
Dieser Request ist im `CD` nicht implementiert. Kann derzeit eigentlich gar nicht passieren, da es ja nur einen gibt...

`510` **Limit Exceeded**
Ein im `CD` existierendes Limit wurde überschritten (der Request war zu groß oder ähnliches).

`511` **Version Unsupported**
Die Version des `MUCS` Protokolls wird von diesem `CD` nicht unterstützt.

`512` **Cannot Call CC**
Der `CC` konnte nicht gestartet werden. Entweder ist die `MUCS`-Umgebung nicht richtig installiert oder der Angerufene hat ein nicht korrektes Startup-Script.

5.2 Limits

Aus praktischen Gründen seien noch die folgenden Limits genannt, die in Bezug auf das Protokoll eingehalten werden sollten[4]:

1. Die Größe eines Requests **SOLLTE** 64 kByte nicht übersteigen [Bra89]. Somit soll sichergestellt werden, daß im Falle eines Fehlers nicht unnötig viele Megabytes an Daten über das Netz fließen.

2. Jeder `CD` **MUß** mindestens in der Lage sein, ein derartig großes Paket zu verarbeiten. Das gleiche gilt für den `CC`.

3. Die Zeilen im Header **SOLLTEN** nicht länger sein als 8 kByte[5]. Das ist unter Umständen auch wichtig, da evtl die Shell-Scripts mit den Daten in Environment-Variablen arbeiten, und die sind in der Regel in der Länge beschränkt.

4. Jeder Prozess **MUß** mindestens Zeilen von einer Länge von 255 Bytes verarbeiten können (ohne `CRLF`).

5. Jeder Prozess **MUß** in der Lage sein auch `LF` als Zeilenende zu akzeptieren, ohne vorheriges `CR`. Jeder Prozess **SOLLTE** jedoch immer `CRLF` als Zeilenabschluß verwenden.

[4] Wir haben hier die Formulierungen den RFCs des `IETF`s nachempfunden.
[5] In Anlehnung an den `httpd` Server von `NCSA`[Mea95].

Kapitel 6

Location Service

„Wo befindet sich gerade XYZ, und wie kann ich ihn erreichen?“ Das sind die beiden Hauptfragen, die sich bei unserem `MUCS` innerhalb der lokalen Umgebung ergeben. Wie bei der Beschreibung des Shell-Scripts `locate` bereits erwähnt, existieren bereits einige Informationsquellen, die für die Lösung des Problems herangezogen werden können. Jedoch zeigt sich recht schnell, daß diese in dieser derzeit existierenden Form in keiner Weise ausreichend sind, um die gewünschte Flexibilität zu erhalten. Ebenso unsinnig wäre es jedoch einen weiteren dedizierten Dienst dieser Art zu entwickeln, der neben den anderen installiert ist. Viel sinnvoller scheint es, einen abstrakten Dienst zu implementieren, der durch geeignete Mechanismen alle diese anderen Dienste mit erbringen kann.

In den nächsten Abschnitten stellen wir den Location Service vor, mit Hilfe dessen wir das Auffinden des Angerufenen in der jeweiligen Umgebung realisierten.

6.1 Aufgabenstellung

Generell lassen sich bei den gewünschten Informationen zwei Kategorien unterscheiden, zum einen solche die relativ statisch sind wie an welchen Rechner ist wer eingeloggt, und solche die sich dynamisch ändern. Zu letzteren gehört zum Beispiel die sogenannte „Idle-Time“, die angibt wie lange an dem entsprechenden Terminal keine Eingaben mehr gemacht wurden.

Ein Mitarbeiter in einem Unternehmen wird sich höchstwahrscheinlich einloggen wenn er morgens kommt, und erst wieder ausloggen wenn er abends nach Hause geht. Die Information bleibt also zu z.B. 10 Stunden statisch. Betrachten wir die „Idle-Time“, so ist davon auszugehen daß sie im Allgemeinen recht niedrig sein sollte, es sei denn er ist gerade unterwegs zum Kaffeeautomaten, zu einer Besprechung oder ähnlichem. Die Information schwankt demzufolge recht stark.

Damit diese beiden Sorten von Informationen auch eine reale Bedeutung haben, müssen sie mehr oder weniger aktuell sein. So ist es wenig hilfreich zu wissen, ob der Benutzer irgendwann in den letzten Stunde einen Taste gedrückt hat oder nicht. Ebenso ist es unsinnig zu wissen, daß er nun seit einer Stunde, 43 Minuten und 42,624 Sekunden angemeldet ist.

Betrachten wir zunächst zwei klassische Vertreter für Lokalisierungsdienste, `rwho` und `rusers`.

6.1.1 Rwho

Das Rwho System ist sehr einfach zu installieren (starten des `rwhod` Daemons auf jedem Rechner) und zu benutzten (durch Aufruf von `rwho`). Die Informationen sind immer recht aktuell und werden sehr schnell erzeugt. Der Grund hierfür ist der, daß jeder partipizierende Rechner eine lokale Datenbank hält,

in der die Informationen aller anderen Rechner enthalten sind. Sollte ein Teil des Netzwerkes kurzzeitig unterbrochen sein, so stehen die älteren Informationen noch immer zur Verfügung. Die Tatsache, daß die Daten auf jedem Rechner lokal gehalten werden sorgt für eine schnelle Antwort, da keine anderen Server befragt werden müssen.

Problematisch ist jedoch die erzeugte Netz- und Rechnerlast. Jeder `rwhod`-Server sendet alle drei Minuten die Daten des lokalen Rechners (vereinfacht gesagt die Datei /etc/utmp) mittels IP-Broadcast in das Netz. Als Folge empfangen alle Rechner im Netz dieses Datenpaket und alle `rwhod`-Server speichern die Information in ihrer lokalen Datenbank ab. In einer Umgebung mit ca. 200 Rechnern bedeutet dies in etwa ein Paket mit Update der Datenbank pro Sekunde. Berichten zu Folge konnte somit in einigen Umgebungen der `rwhod` bis zu 20 % der CPU und des Netzes belegen. Desweiteren empfangen auch Rechner die Datenpakete, auf denen kein `rwhod` läuft oder laufen soll (wie dedizierte Server Rechner), da es sich um einen Broadcast handelt. Aus diesen Gründen ist in den meisten größeren Umgebungen dieser Dienst nicht aktiviert.

Seit der Einführung von IP-Multicast [Dee88, Dee89a] wurde eine neue Version des `rwhod` Servers entwickelt [Dee89b]. Dieser sendet die Daten nicht mehr via Broadcast, sondern mittels IP-Multicast übers Netz. Somit empfangen nur noch die Rechner derartige Pakete, die ebenfalls an der Benutzung von `rwho` interessiert sind. Jedoch bleibt das Problem der Netz- und CPU-Last nach wie vor bestehen.

6.1.2 rusers

Um das Problem mit der hohen CPU- und Netz-Last zu umgehen, wurde ein weiteres System entwickelt, genannt `rusers`. Im Gegensatz zu `rwho` läuft hier nicht ständig ein Server im Hintergrund, der die Informationen sammelt und die eigenen Informationen verteilt. Beim Aufruf von `rusers` sendet dieser einen Broadcast (genauer mittels RPC) in das Netz und fragt somit alle Rechner nach den aktuellen Informationen ab. In der Regel wird daraufhin auf jedem Rechner der `rusersd` Server über den `inetd` Server gestartet, und sendet die Informationen über den eigenen Rechner an den anfragenden `rusers` Prozeß zurück.

Hierbei zeigt sich, daß unter der Annahme, daß keine Anfragen gestartet werden, weder die CPU, noch das Netzwerk mit unnötigen Daten belastet wird, ganz im Gegenteil zum `rwho`. Allerdings erzeugt die Anfrage eines Rechners eine Lawine von Antworten von jedem `rusersd` Server, womit eine hohe Anzahl von Kollisionen auf Broadcasting-Netzwerken wie Ethernet vorprogrammiert sind. Hierbei ist noch zu berücksichtigen, daß die Übertragung mittels `RPC` ausgeführt wird, womit für jede Antwort mehrere Netzpakete anfallen. Dieser Effekt läßt sich auch ganz einfach anhand der Antwortzeiten des `rusersd` sehen. So benötigen die Antworten eines `rusers` ca. fünf Sekunden in einer Umgebung wie der `GMD`.

Ein weiteres Problem besteht darin, daß keine Information dafür zur Verfügung stehen, wann alle Rechner geantwortet haben, oder ob noch einer fehlt. So müßte ein suchender Prozess wie der `CD` z.B. 10-15 Sekunden lang auf Antworten warten, und dann den `ruser` Prozeß selber terminieren.

6.2 Unsere Version

Bei der Betrachtung der bisher existierenden Dienste konnten wir bereits zwei Extremfälle sehen, zum einen eine Verschwendung von Ressourcen beim `rwho`, zum anderen hohe Antwortzeiten beim `rusers`. Desweiteren stehen diese Dienste isoliert da. Eine Erweiterung selbiger ist eigentlich nicht möglich. Aus diesen Gründen entschlossen wir uns das im folgenden vorgestellte Konzept einzusetzen.

Um die benötigten Informationen so schnell wie möglich zu erhalten kommen im Grunde nur zwei verschiedene Zugriffsmethoden in Frage:

- Durch Stellen einer Anfrage an einen dedizierten Server,
- oder durch Zugriff auf eine verteilte oder zentrale Datenbank.

Die Verwendung eines dedizierten Servers birgt immer die Gefahr der Überlast, bzw. einer potentiellen Engstelle. So müßte dieser Server die Daten durch geeignete Verfahren erhalten und nach Möglichkeit auch speichern, damit im Falle eines Ausfalls die Wiederanlaufzeit so gering wie möglich ist. Um die Ausfallsicherheit zu erhöhen könnten auch mehrere Server installiert sein. Dies führt aber ebenfalls zu weiteren Problemen. Zuerst stellt sich die Frage woher ein Client weiß, welche Server es gibt und an welchen er sich wenden soll. Desweiteren sollten die Informationen auch konsistent sein, womit sich wiederum die gleichen Probleme ergeben wie beim Zugriff auf eine verteilte Datenbank. So müssen sich die Server beim Zugriff auf die Datenbank untereinander synchronisieren, was in jedem Fall nicht trivial zu lösen ist. Alternativ könnte man auch eine kommerziellen Datenbank wie `Informix` oder `Oracle` hierzu verwenden, jedoch erscheint uns dieses ein wenig überdimensioniert und zu eingeschränkt nutzbar zu sein.

6.2.1 Die verteilte Datenbank

Unsere Lösung stellt eine Art Kompromiß aus den oben genannten Punkten dar. Damit der Zugriff auf die Informationen so schnell wie möglich erfolgen kann, schied ein oder mehrere dedizierte Server aus. Stattdessen bevorzugen wir eine verteilte Datenbank, die von verschiedenen Informations-Servern gefüllt werden. Um die Probleme, die beim konkurrierenden Zugriff entstehen zu minimieren, besitzt jeder Server seine eigene Datenbank, auf die er schreibend zugreift. Alle anderen Clients greifen immer nur lesend auf diese Informationen zu.

Der Trick hierbei ist, daß unsere Datenbank in Wirklichkeit ein Directory in einem gemounteten `NFS` Filesystem ist. Alle Rechner innerhalb der lokalen Umgebung, die entweder als Informationsträger agieren oder Anfragen für solche haben, müssen dieses Filesystem gemountet haben. Jeder Informations-Server erzeugt ein oder mehrere „Datenbankfiles“, in denen diverse Informationen gespeichert werden können. Diese Files müssen der folgenden Namenskonvention folgen:

FileName	::=	ServerId “|“ Infomation
ServerId	::=	*Eindeutige ID eines Servers wie “*`who`*“*
Information	::=	(Information Information) | InfoType “=“ Info “|“
InfoType	::=	“user“ | “ host“ | “line“ | “remote“
	::=	| (*Eindeutige ID des Types der Information*)
Info	::=	*zum Type gehörende Information*
ReservedChars	::=	“%“ | “/“ | “|“ | “=“
Escape	::=	“%“ Hex Hex
Hex	::=	“0“ | “1“ | “2“ | “3“ | “4“ | “5“ | “6“ | “7“ | “8“ | “9“
		| “A“ | “B“ | “C“ | “D“ | “E“ | “F“ “a“ | “b“ | “c“ | “d“ | “e“ | “f“

Es ist unbedingt darauf zu achten, daß weder im *InfoType* noch im *Info* Feld die reservierten Zeichen auftauchen. Sollte es dennoch nötig sein, ein derartiges Zeichen verwenden zu müssen, so muß das Zeichen mit Hilfe eines Escape-Mechanismus dargestellt werden, angelehnt an RFC1738 [BLMM94]. Das Zeichen wird somit durch ein `%`, gefolgt durch den zweistelligen hexadezimal Wert des Zeichens aus dem `ASCII` Zeichensatz, dargestellt. Ein “`/`“ würde somit als “`%2f`“ oder “`%2F`“ dargestellt werden.

Der obigen Beschreibung folgend, könnte der Eintrag von einem Informationsdienst mit dem Kennung `who` für den Benutzer `hgs`, der an den Rechnern `lupus` und `ursa` angemeldet ist, wie folgt aussehen:

```
who|host=lupus|user=hgs|line=console|
who|host=lupus|user=hgs|line=pts%2f0|
who|host=ursa|user=hgs|line=pts%2f3|remote=lupus|
```

Sollte noch ein Server installiert sein, der mit einem Active-Badge-System [WHFG92, WH92, Hop94] zusammenarbeitet, so würde sich die Datenbank beispielsweise um folgende Einträge erweitern:

```
badge|room=toilet.4.1|user=hgs|
badge|room=423|user=hgs|
```

Ein Client, der beispielsweise nach allen derzeitigen möglichen Rechnernamen sucht, an denen des Benutzers `hgs` angemeldet ist, muß somit nur in dem Directory, welches die eigentliche Datenbank darstellt, nach Dateien suchen, die den Informationstype `host` besitzen, sowie den Informationstyp `user` mit dem Wert `hgs`. Die Rechnernamen können alsdann aus dem Werteteil des Informationstype `host` bestimmt werden.

Die Server können jedoch noch weitere Informationen ablegen, wie z.B. die Idle-Time an den jeweiligen Pseudo-TTYs. Diese können dann in dem File selber abgelegt werden, wobei das Format völlig frei gewählt werden kann. Es muß nur bei allen Servern des gleichen Types einheitlich sein, so daß ein Client anhand der ServerId das Format bestimmen kann.

Ein weiter wichtiger Punkt, den die Server in jedem Fall leisten sollten, ist ein „Clean-Up“ beim Start. Sollte der Rechner oder aber auch der Server selber abgestürzt sein, so befinden sich falsche Daten in der Datenbank. Die Server sollten deshalb bei jedem Neustart ihre eigenen Datensätze (erkennbar an der Server-Id) löschen. Weiterhin sollten alle Server über ein geeignetes Verfahren zum ordnungsgemäßen Terminieren zur Verfügung stellen. So sollten sie das Signal `SIGTERM` richtig interpretieren, daß vom `init` Prozeß bei Änderung des „Run-Levels“ gesendet werden abfangen, und daraufhin ihre Datensätze löschen. Somit wird sichergestellt, daß alte Daten nicht unendlich lange in der Datenbank verweilen.

6.2.2 Die Server

Wir haben beispielhaft den einfachen Server `lswhod` implementiert, der ähnlich wie der `rwhod` arbeitet. Er fragt alle drei Minuten die Datei `/etc/utmp` ab und füllt mit diesen Informationen die Datenbank (`/usr/local/mucs/ls-db`).Es werden nur die Datensätze von `utmp` verwendet, die als „USER_PROCESS“ Einträge gekennzeichnet sind. Der `lswhod` (seine Server-Id lautet `who`) stellt die folgenden Informationen zur Verfügung:

`user=`*Name des Benutzers*
`host=`*Name des Rechners, an dem er angemeldet ist*
`line=`*Name des TTYs*
`remote=`*Name des entfernten Rechner, nur vorhanden bei remote-login*

Beim Starten und Beenden des Servers entfernt dieser alle Einträge aus der Datenbank, die von ihm für den Rechner, auf dem er gerade läuft, erstellt wurden. Für jeden sich seit dem letzten Durchlauf neu angemeldeten Benutzer, schreibt er einen Eintrag in die Datenbank, für jeden wieder abgemeldeten Benutzer wird der Eintrag gelöscht. Somit wird der `NFS`-Server, auf dem sich die Datenbank befindet, so wenig wie möglich belastet, da nur Änderungen zu einem Zugriff auf die Datenbank führen.

6.3 Bewertung

Da noch keine Erfahrungswerte für dieses neue Verfahren existieren, kann eine genaue Bewertung derzeit noch nicht gegeben werden. In jedem Fall können jedoch einige theoretische Überlegungen angestellt werden, die in den folgenden Abschnitten dargestellt sind.

6.3.1 Vorteile

Mit dem vorgeschlagenen Verfahren ergeben sich eine Reihe von Vorteilen gegenüber herkömmlichen Informationsdiensten:

- Im Gegensatz zu `rwho` werden andere Rechner nicht durch den Empfang und der Verarbeitung von Informationen beschäftigt, die sie zum großen Teil nicht interessieren. Ein Rechner bekommt nur dann Informationen, wenn er sie wirklich benötigt, ausgelöst durch die Anfrage eines Clients.
- Die Anfrage eines Clients für eine Information belastet die anderen Rechner ebenfalls nicht, anders als beim `rusers`. Die Anfrage wird direkt (mittels unicast, via `NFS`) an den Server des Filesystems gerichtet, andere Rechner empfangen diesen nicht.
- Durch den einfachen Zugriff über den Eintrag in einem Directory sind die nötigen Wartezeiten minimal. Es entstehen weder große Wartezeiten wie beim `rusers`, noch hat der Client Schwierigkeiten festzustellen, ob alle Informationen angekommen sind.
- Durch die Namensgebung und die Möglichkeit, weitere Informationen in den Files abzulegen, kann das System im Prinzip um beliebige Informationsdienste erweitert werden. Active-Badges, elektronische Schlösser oder ähnliches können somit ebenfalls detailierte Informationen bereitstellen, die von allen Applikationen auf einfache Art und Weise abgefragt werden können.

6.3.2 Nachteile

Allerdings existieren auch einige Nachteile, die ebenfalls nicht verschwiegen werden sollen.

- Aufgrund der Natur der Datenbank, die als Directory in einem über `NFS` gemounteten Filesystem vorliegt, liegt diese auf einen zentralen Rechner. Sollte dieser ausfallen, so ist das ganze System unbenutzbar.
- Aufgrund der Natur der Speicherung der Daten in einer Datenbank kommt es zu einem Problem, wenn einer der Server oder Rechner abstürzt. In diesem Fall stehen die alten Daten noch immer in der Datenbank, so daß anfragende Clients mit falschen Informationen beliefert werden. Im Falle eines Rechnerabsturzes dürfte dieses nicht so schlimm sein, der weitere Programme dieses sehr schnell merken werden (sie können zu diesen Rechner keinen IP Verbindung aufbauen). Wenn der Rechner wieder neu gestartet wird, werden die Informations-Server ihre eigenen alten Datensätze automatisch wieder löschen. Im Falle eines defekten Servers kann dies jedoch zu dauerhaft falschen Informationen führen.

Es sei noch kurz erwähnt, daß das von vorgeschlagenen Verfahren, einen Teil der Informationen in den Dateinamen zu legen zwingend ein UNIX System voraussetzt, das lange Dateinamen verarbeiten kann. Das sollte heutzutage kein Problem mehr sein, obwohl scheinbar noch einige ältere System-V Systeme existieren, bei denen der Dateiname auf 14 Zeichen beschränkt ist.

6.3.3 Zusammenfassung

Zusammenfassend läßt sich feststellen, daß einige der Probleme die bei den herkömmlichen Diensten existieren, flexibel gelöst wurden. Insbesondere besteht die Möglichkeit der beliebigen Erweiterung des Systems durch neue Informationsquellen, deren Informationen alle in einer gemeinsamen Datenbank gehalten werden.

Der von uns vorgestellte Mechanismus zum Lokalisieren verschiedener Objekte, vorzugsweise Benutzer, kann selbstverständlich auch mit anderen Diensten gekoppelt werden. So sei hier am Rande noch auf einige weitere Ansätze hingewiesen, die das Problem zum Teil in einem größeren und globalen Umfeld betrachten [BD93, Man91]. Da für diese Ansätze keine Quellen zur Verfügung standen, wurden diese

auch nicht weiter berücksichtigt. Den Ansatz von René Stolp [Sto94] haben wir ebenfalls nicht weiter verfolgt, da dieser auf Diensten wie `rwho` aufsetzt (vgl. Abschnitt 6.1.1).

Kapitel 7

NeVoT Erweiterungen für AIX

Die gesamte Entwicklungs- und Implementierungsphase des bisher vorgestellten Systems wurde nicht auf den, in der GMD und TU hauptsächlich vertretenen SUN Workstations durchgeführt, sondern auf RS/6000 Rechnern von IBM. Um für unsere Arbeit auch einen realen Audioagenten zu Testzwecken benutzen zu können entschieden wir uns, den Agenten NEVOT von *Henning Schulzrinne* [Sch95] auf AIX[1] zu portieren.

Abschnitt 7.1 beschreibt den wesentlichen Kern, die Ansteuerung der Audio-Hardware. Die Einstellung einiger Audioparameter wie Lautstärke, Balance etc. wurde bei NEVOT mit Hilfe des externen Programms tkaudio vorgenommen. Dies ist jedoch unter AIX nicht möglich, so daß wir hierfür eine andere Möglichkeit schaffen mußten. Das für diesen Zweck neu eingeführte „Control-Panel“ „Audio Control“ ist im Abschnitt 7.2 beschrieben. Der letzte Abschnitt 7.3 beschreibt das neue tcl/tk Kommando „audioctl“, daß die Einstellung der Audioparameter durch das „Control-Panel“ ermöglicht.

7.1 Ansteuerung der Audio-Hardware

Als Workstation für die Audioansteuerung diente uns eine IBM RS/6000 M20. Da diese standardmäßig über keine Audio-Hardware verfügt, wurde eine Audiokarte mit dem Namen „Ultimedia Audio Adapter“ eingebaut. Von *Bob Olson* vom *Argonne National Laboraty* wurde uns jedoch mitgeteilt, daß unser Code auch auf einer 41P mit integriertem Audiosystem funktioniert.

Um bei der Benutzung der Audio-Hardware von den jeweiligen Unterschieden auf den verschiedenen Maschinen unabhängig zu sein, entwickelte IBM eine Bibliothek (Ultimedia Services Library, UMS). Diese wurde unter Verwendung des System Object Management Systems (kurz SOM) erstellt, so daß das gleiche Interface auch auf verschiedenen Hardwarearchitekturen zur Verfügung steht. Leider stellte sich recht schnell heraus, daß diese Bibliothek zum einen recht aufgebläht ist (ca. 3 MB), wodurch die Applikationen recht viel Zeit benötigten, um gestartet zu werden. Das größte Problem war jedoch, daß diese Bibliothek keine Möglichkeit bietet, mittels des select()-Call auf das Anstehen neuer Daten zu warten. Der erste Versuch dies mit Hilfe eines Timers zu umgehen wurde wieder verworfen, da zum einen der Timer nicht genau genug arbeitete und zum anderen die Systembelastung dadurch erheblich anstieg. Schließlich entschieden wir uns, die UMS Bibliothek nicht zu verwenden und die Audio-Hardware wie allgemein üblich direkt anzusteuern.

[1] AIX is das UNIX Betriebssystem von IBM für die RS/6000-Familie

7.1.1 Aufnahme

Der von Henning Schulzrinne entwickelte Agent NEVOT, baut darauf auf, daß alle 20 ms die Audiodaten vom Device gelesen (Aufnahme) und über das Netz zum Empfänger gesendet werden[2]. Dies scheint bei Workstations von `SUN` problemlos zu funktionieren, jedoch nicht bei dem von uns verwendeten Adapter für die `RS/6000`[3]. Dieser arbeitet intern mittels zweier „Direct Memory Access Bereichen“ (DMA-Seiten) die wechselseitig mit Daten gefüllt werden. Sobald eine DMA-Seite gefüllt ist, wird sie dem System zur Verfügung gestellt. Als Folge wird ein wartender `select()` Call terminiert, und die Audiodaten können eingelesen werden. Leider birgt die Benutzung der DMA-Seiten auch ein Problem: So läßt sich die Größe dieser nur auf einen Wert zur Basis 2 einstellen. Bei einer Abtastrate von 8 kHz mit 8 Datenbits mono, fallen jedoch 160 Bytes in den 20 ms an. Versuche haben gezeigt, daß das Setzen der Größe der DMA-Seite auf 128 den Rechner zu stark belastet, so daß wir einen Wert von 256 Bytes benutzen. Dies hat zur Folge, daß die Daten zwischengespeichert werden müssen.

7.1.2 Wiedergabe

Ein weiteres Problem bestand auf der Seite der Wiedergabe. Hier gilt im Prinzip das Gleiche wie oben. Jedoch startet die Audio-Hardware die Wiedergabe erst dann, wenn zwei DMA-Seiten gefüllt sind. Sollte zu dem Zeitpunkt, an dem die Audio-Hardware die beiden DMA-Seiten austauscht, die aktuelle nicht vollständig gefüllt sein, so wird die Wiedergabe beendet. Zum Starten werden wieder zwei gefüllte DMA-Seiten benötigt.

Dies führt dazu, daß sich eine kleine Queue innerhalb des Adapters aufbaut, die demzufolge drei mal der Größe der DMA-Seite entspricht. Bei der oben erwähnten Abtastrate von 8 kHz entspricht dies einer Verzögerung von $125\mu s * 3 * 256 = 96ms$, die bereits recht hoch liegt. Addiert man noch die Verzögerung durch das Netzwerk hinzu und nimmt an, daß beim Empfänger ebenfalls diese Verzögerung auftritt, resultiert dies in einer deutlich hörbaren Gesamtzeit von $> 200ms$[4]. Entsprechend der Empfehlung G.114 der International Telecomunication Union (ITU) [ITU94], liegt diese jedoch noch im akzeptablen Rahmen ($\leq 300ms$).

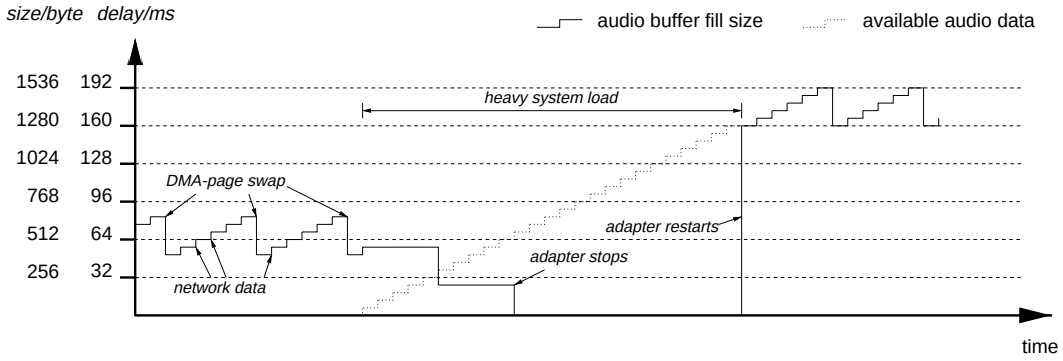

Abb. 7.1: Vergrößerung der Verzögerung durch hohe Systemlast

Ein weiteres Problem tritt dann auf, wenn beispielsweise aufgrund plötzlicher hoher Systemlast die Audiodaten nicht schnell genug zum Audiodevice geschrieben werden können (vgl. Abbildung 7.1). In diesem Fall beendet es kurzzeitig die Wiedergabe, was als Knacken zu hören ist. Da die Audiodaten jedoch nicht verlorengegangen sind sondern nur später abgespielt werden, entsteht plötzlich eine große Verzögerung zwischen der Aufnahme und der Wiedergabe beim Empfänger. In unserer Umgebung betrug diese im Extremfall ein bis zwei Sekunden. Da diese in keinem Fall akzeptabel ist, haben wir einen

[2] Es können auch mehrere Empfänger sein, jedoch spielt das für die weitere Betrachtung keine Rolle.
[3] Das gleiche gilt auch für das in die `41P` integrierte Audiosystem.
[4] Gemeint ist die Zeit die benötigt wird, einen Laut vom Sender zum Empfänger und wieder zurückzuschicken.

Mechanismus implementiert, der für diesen Fall die Verzögerung im Laufe der Zeit wieder verkleinert.

Hierfür gibt es die beiden folgenden Möglichkeiten,

- durch „Löschen“ der angesammelten Audiodaten,
- oder durch partielles entfernen einzelner Audiosamples.

Die erste Methode beinhaltet den Nachteil, daß somit ganze Worte entfernt werden würden. Da es unter Umständen, gerade bei hoher Systemlast, öfters hintereinander vorkommen kann, daß derartige „Aussetzer“ auftreten, beeinträchtigt dies im hohen Maße die Verständlichkeit. Der zweite Ansatz umgeht dieses Problem dadurch, daß nur einige Audiosamples entfernt werden. In unserer Implementierung haben wir bei einer Abtastrate von 8 kHz jeden 32. Audiosample entfernt. Dadurch ergibt sich zwar zwangsläufig ein kleiner Frequenzhub, und der „Sprecher“ hört sich an, als ob er schneller reden würde, jedoch ist dies kaum zu bemerken, insbesondere nicht bei Sprache. Der Quell-Code für `netutil/ibm_audio.c` ist im Anhang D.1 enthalten.

7.2 Einstellung der Audioparameter

Abbildung 7.2 zeigt das erstellte Control-Panel (der Quell-Code für „`netutil/ibm_audioctl.tcl`“ ist im Anhang D.2 enthalten). Es ist unterteilt in zwei Hälften, wobei mit der oberen die Parameter für den Audio-Eingang, mit der unteren die des Audio-Ausganges eingestellt werden können.

Die Datenquelle wird durch Auswahl einer des vier Checkbuttons selektiert. Zur Auswahl stehen der Mikrophoneingang (entweder als High-Gain oder Low-Gain Variante), der externe Eingang Line1, sowie der Eingang Aux1, der, je nach Rechnermodell, mit dem internen CD-ROM Laufwerk verbunden ist. Unabhängig vom gewählten Eingang, kann die Empfindlichkeit für die beiden Kanäle Links und Rechts, im Bereich von 0 bis 100 eingestellt werden.

Da bei dem von uns verwendeten Audioadapter die drei Ausgänge Headphone, External Speaker und Out1 intern zusammengeschaltet sind, existiert keine Auswahlmöglichkeit für den Ausgang, es sind immer alle aktiv. Lediglich der interne Lautsprechner, je nach Modell der in der Tastatur, oder der im Rechner eingebaute, kann ein- und ausgeschaltet werden. Die Lautstärke kann ebenfalls im Bereich von 0 bis 100 eingestellt werden, jedoch nur für beide Kanäle gemeinsam. Jedoch kann die Balance zwischen den beiden Kanälen variiert werden.

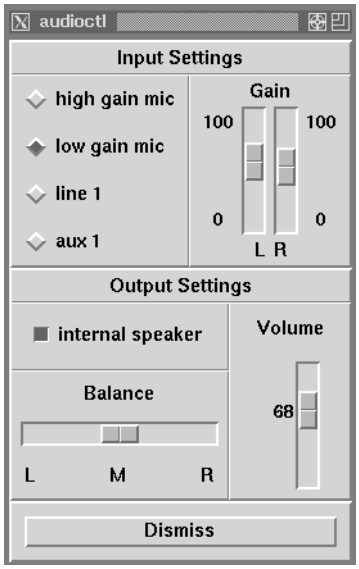

Abb. 7.2: Control Panel

7.3 Schnittstelle zwischen `tcl` und er Audio-Hardware

Damit das zuvor beschriebenen „Control-Panel“ die Einstellungen für die Audio-Hardware machen kann, mußte ein neues `tcl/tk` Kommando „`audioctl`“ implementiert werden. Dieses wurde jedoch nicht an den derzeitigen Gegebenheiten und Möglichkeiten des Audioadapters orientiert, sondern wurde für die allgemeine Verwendung ausgelegt. Welche der im folgenden beschriebenen Parametern bei der jeweiligen Audio-Hardware eingestellt werden kann, ist abhängig von den jeweiligen Möglichkeiten.

Das neue Kommando besitzt die folgenden Optionen und Parameter:

`-input` *input*
: Bestimmt den Audioeingang, wobei immer nur einer aktiv ist (die Eingänge werden also nicht gemischt). Zur Auswahl stehen die Eingänge „`lowgainmike`“, „`highgainmike`“, „`line1`“, „`line2`“ und „`line3`“.

`-output` *output*
: Bestimmt den Ausgang: „`line1`“, „`line2`“, „`line3`“.

`-volume` *volume*
: Bestimmt die Lautstärke des Ausganges im Bereich von 0 (aus) bis 100 (volle Lautstärke).

`-balance` *balance*
: Verschiebt die Balance zwischen dem linken und rechten Kanal für die Wiedergabe. Der Wert muß im Bereich von -100 (links) bis 100 (rechts) liegen. Bei einem Wert von 0 liegt die Balance ganau in der Mitte.

`-leftgain` *leftgain*
: Stellt die Eingangsempfindlichkeit für den linken Kanal im Bereich zwischen 0 (maximale Absenkung) bis (100) (maximale Empfindlichkeit) ein.

`-rightgain` *rightgain*
: Wie `-leftgain`, jedoch für den rechten Kanal.

`-gain` *gain*
: Wie `-leftgain`, jedoch wird hiermit die Empfindlichkeit für beide Kanäle gleichzeitig eingestellt.

`-intspeaker` *value*
: Schaltet den internen Lautsprecher ein „`on`“ oder aus „`off`“.

Der Quell-Code für „`nevot/cmd_audioctl.c`“ ist im Anhang D.3 enthalten.

Kapitel 8

Zusammenfassung

In den vorangegangenen sieben Kapiteln haben wir ein Konferenz-System und deren eine Hälfte der Implementierung vorgestellt und genauer beschrieben. Während der Implementierung, als auch bei der Ausarbeitung des schriftlichen Teils stießen wir auf eine Reihe Problemen, die wir teils vollständig, teils zufriedenstellend gelöst haben. Auch blieben aus zeitlichen Gründen einige Ideen unverwirklicht und wurden nicht genauer untersucht. Wir denken jedoch, daß das von uns soweit entwickelte und vorgestellte System, durchaus eine neue Richtung in den Bereich Multimedia Conferencing aufgezeigt hat.

Aus diesem Grunde seien hier zum Schluß noch einige offene Punkte erwähnt, die in der derzeitig Realisierung nicht vollständig gelöst wurden. Im Anschluß daran geben wir noch eine Reihe von Verbesserungsvorschlägen, die bei zukünftigen Weiterentwicklungen durchaus mit berücksichtigt werden sollten.

8.1 Offene Punkte

8.1.1 Verbindungsabbruch

Die derzeitige Implementierung des `CD` stellt einen Verbindungsabbruch vom Anrufer nach eingegangenen Call-Request nicht fest. In Folge dessen wird der Angerufene zunächst über den Anruf informiert, obwohl der Anrufer es nicht mehr „klingeln“ läßt. Der `CD` müßte demzufolge den Zustand der IP Verbindung stets überwachen und bei Abbruch die weitere Bearbeitung des Anrufes unterbrechen. Wir haben bisher jedoch keine effiziente und einfache Lösung gefunden, dieses zu realisieren.

8.1.2 Protokollieren von Anrufen

Es wäre durchaus hilfreich wenn für jeden Benutzer ein lokales Log-Files geführt werden würde, so daß er auch über Anrufe informiert wird, wenn er nicht da ist. Ebenso könnte für jeden Anruf eine Mail verschickt werden etc.

8.2 Verbesserungsvorschläge

8.2.1 Zum Protokoll

Das von uns verwendete Protokoll könnte durchaus verbessert werden. Zum einen wäre es günstig, wenn mit einem Request auch mehrere Leute angesprochen werden könnten, womit sich die Anzahl der benötigten TCP-Verbindungen und -Pakete verringern ließe. Jedoch ist das für den `CD` nicht so einfach zu

behandeln. In diesem Zusammenhang stellt sich auch die Frage nach der geeigneten Form der Übermittlung der „Antworten“. Immerhin können einige Angerufene entweder nicht da sein, an der Konferenz nicht teilnehmen wollen oder diese annehmen. Damit entfällt jedoch die Möglichkeit, diese verschiedenen Informationen über die Status-Line zurückzuliefern. Eine Möglichkeit bestünde darin, über die gleiche Verbindung für jeden angesprochenen Benutzer eine abgeschlossenen Response zu generieren. Jedoch entfernen wir uns damit noch weiter vom HTTP.

8.2.2 Alias Listen

In der bisherigen Implementierung können wir nur exakt einen Benutzer mit einem Request ansprechen. Handelt es sich hierbei um einen Alias, so darf dieser nur auf einen Benutzer verweisen. Bei mehreren wird der Anruf abgelehnt. Der Grund hierfür ist, daß exakt die gleichen Probleme auftreten, wie auf der vorherigen Seite. Die andere Möglichkeit, die Liste der Benutzer des aufgelösten Aliase an den Anrufer zurückzuschicken ist auch nicht sehr vernünftig. Somit könnte ein (evtl. nur scheinbarer) Anrufer die Informationen über eine Mailing-List oder sonstigen Listen erhalten, was nicht in jeder Umgebung gewünscht ist, da die Anonymität der einzelnen nicht länger gewahrt ist.

8.2.3 „Privacy“

Ein weiteres Problem stellt die sogenannte „Privacy“[1] dar. Mit Hilfe des MUCS ist es möglich festzustellen, ob ein Benutzer da ist, oder nicht. Dies ist ebenfalls eine sehr heikle Information, die nicht ohne weiteres herausgegeben werden sollte. Aus diesem Grunde sind z.B. in den meisten Domains die rwhod oder fingerd Prozesse nicht gestartet, so daß von außen nicht festgestellt werden kann, ob ein entsprechender Benutzer da ist, oder nicht. In der TU sind ebenfalls die wtmp Files für den „normalen“ User nicht zugängig, aus den gleichen Gründen. Auf der anderen Seite kann mittels eines Anrufes auch sehr einfach festgestellt werden, ob die entsprechende Person da ist oder nicht. Der einzige Unterschied besteht darin, daß nicht jeder die Telefonnummer kennt, was im Falle der E-Mail-Adresse sicherlich anders ist.

8.2.4 Authentizität

Damit anhand der benutzerlokalen Einstellungen entschieden werden kann, ob der Anruf überhaupt signalisiert werden soll, muß der Anrufer identifiziert werden können. Derzeit geschied dies durch Übergabe der E-Mail Adresse des Anrufers im MIME Header. Diese Methode ist jedoch alles andere als sicher und die Möglichkeit, eine solche Adresse zu „fälschen“ ist sehr groß und ohne großen Aufwand durchführbar. Das ist jedoch ein generelles Problem im Internet, das auch beim Empfang von E-mails existiert. Abhilfe kann da nur durch die Verwendung eines Verschlüsselungsverfahren wie beispielsweise PGP [ASZ95] geschafft werden.

8.2.5 Forwarding

Derzeit ist der CD nicht in der Lage selber einen forward direkt auszuführen. So sendet beispielsweise der CD auf dem Mail-Server eine *redirect* Message zum Anrufer zurück, in der der oder die Hosts angegeben sind, über die der Angerufene zu erreichen ist. Es wäre jedoch auch denkbar, daß dieser CD den Anruf selbstständig an die anderen CDs weiterreicht. Dieses würde den Ablauf im CD wiederum stark ändern und könnte unter Umständen zu verschachtelten Forwards führen.

[1] Leider existiert kein deutsches Pendant für dieses Wort, wie auch Prof. Lutterbeck einmal unterstrich.

8.2.6 Location Server

Der Location Server sollte noch erweitert werden, so daß auch andere Dienste wie Active Badges etc. verwendet werden können. So sollte lieber ein allgemeinerer Server erstellt werden, der auch für die Verwaltung anderer Informationen verwendet wenden kann.

8.2.7 Fehlerbehandlung

Sollte irgendwo in einem der Shell-Scripts ein Fehler auftreten oder eines wegen eines Timeout terminiert werden, so sollte dieses in geeigneter Form entweder dem Anrufer, dem Angerufenen oder dem MUCS Administrator via E-Mail mitgeteilt werden. Anderenfalls dürfte es recht schwierig sein die vollständige Funktionalität sicherzustellen.

8.2.8 Script zum Auffinden des Benutzers

Es gilt zu überlegen ob im Falle von Hosts, auf denen einen große Zahl von Anrufen eingehen werden (wie die Mail-Exchanger) nicht ein anderes Script oder zumindest eine vereinfachte Suche benutzt werden sollte, bzw. verwendet werden können sollte. So könnte anstelle von Anfragen an den Location-Server eine einfache Konfigurationsdatei nach dem Angerufenen hin durchsucht werden und ein fest eingestellter Forward zum Anrufer zurückgesendet werden. Auf diese Weise könnten derartige Rechner, die ohnehin eine große Last zu tragen haben erheblich entlastet werden. Zwar entsteht hierdurch eventuell ein weiterer Forward, aufgrund des Cachings jedoch nur beim ersten Anruf. Desweiteren könnte davon ausgegangen werden, daß die in der Konfigurationsdatei eingetragenen Rechner genau den Arbeitsplatzrechner entsprechen, an denen sicher dieser Benutzer am meisten aufhält. Das jeweilige Anpassen an die jeweiligen Erfordernisse ist jedoch aufgrund der Verwendung der Shell-Scripts extrem einfach möglich.

8.2.9 Auffinden des Benutzers

Zusätzlich sollte noch eine Möglichkeit bestehen einen Benutzer, der sich gerade dem Arbeitsplatz eines anderen Mitarbeiters befindet, zu erreichen. Dies sollte auch dann möglich sein, wenn der angerufene Benutzer an diesem Rechner nicht angemeldet ist! (Die Informationen, daß er sich dort aufhält muß demzufolge durch ein Active-Badge-System oder ähnlichem bekanntgegeben worden sein.) Dies ist besonders in Notfällen oder extrem wichtigen Gesprächen sinnvoll. Jedoch entstehen hierbei für den Ablauf einige neue Probleme, die jedoch alle relativ einfach zu lösen seien sollten. Da wir jedoch ersteinmal des System im realen Einsatz eine Weile testen wollten, entschieden wir uns diesen Punkt nicht weiter zu berücksichtigen.

8.2.10 Script-Generator

Wie bereits in 4.2 erwähnt, sollte ein Script-Generator entwickelt werden, der die Erstellung eines Scripts für die „Überprüfung“ des Anrufes erleichtert. Mit diesem, am sinnvollsten graphischen Tool sollte dann jeder in der Lage sein, beliebig komplexe Bedingungen für die Annahme eines Anrufes formulieren zu können.

Anhang A

Source-Code für den CD

A.1 cd.c

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           main.c
 *      written by:     Christian Zahl
 *      description:    ...
 *
 * $Id: main.c,v 1.2 1995/10/22 09:22:25 czahl Exp czahl $
 *
 * $Log: main.c,v $
 * Revision 1.2  1995/10/22  09:22:25  czahl
 * *** empty log message ***
 *
 * Revision 1.1  1995/10/01  12:27:35  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: main.c,v 1.2 1995/10/22 09:22:25 czahl Exp czahl $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>
#include <syslog.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/file.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <errno.h>
#include <sys/signal.h>

/*----- user includefiles -------------------------------------------------*/
#include "debug.h"
#include "cd.h"

/*----- defines -----------------------------------------------------------*/
#define MUCS_PORT_NO            9000
#define MUCS_SERVICE_NAME       "mucs"
#define PROG_NAME               "cd"

/*----- type definitions --------------------------------------------------*/

/*----- local functions ---------------------------------------------------*/
static  void                    ParseOptions (int argc, char *argv);
static  void                    Exit (int status);

/*----- global functions --------------------------------------------------*/

/*----- local variables ---------------------------------------------------*/
static  int                     logFacillity = LOG_LOCAL0;
static  int                     serverPort;
static  int                     openedConnections = 0;
static  int                     closedConnections = 0;
static  int                     portNo = -1;
static  int                     serverMode = 0;
static  int                     serverSocket = -1;

/*----- global variables --------------------------------------------------*/
        int                     debugLevel = 0;
        char                    myDomainName[MAXDNAME+1];
        char                    myHostName[MAXDNAME+1];

/*-------------------------------------------------------------------------*/
```

```
static void Exit (int status)
{
        syslog (LOG_DEBUG, "exiting with status of %d", status);
        closelog ();
        exit (status);
} /* Exit */
/*---------------------------------------------------------------------------*/
static void SigChldHandler (int sigNo, int pid)
{
        closedConnections++;
        syslog (LOG_DEBUG, "%d opend / %d closed", openedConnections, closedConnections);
        wait (NULL);
        signal (SIGCHLD, SigChldHandler);
} /* SigChldHandler */
/*---------------------------------------------------------------------------*/
static void SigTermHandler (int sigNo, int pid)
{
        signal (SIGTERM, SigTermHandler);
        syslog (LOG_DEBUG, "SIGTERM catched");
} /* SigTermHandler */
/*---------------------------------------------------------------------------*/
static void SigHupHandler (int sigNo, int pid)
{
        signal (SIGHUP, SigHupHandler);
        syslog (LOG_DEBUG, "SIGHUP catched");
} /* SigHupHandler */
/*---------------------------------------------------------------------------*/
static void SigIntHandler (int sigNo, int pid)
{
        signal (SIGINT, SigIntHandler);
        syslog (LOG_DEBUG, "SIGINT catched, terminating...");
        if (serverMode)
          close (serverSocket);
        else {
          fclose (stdin);
          fclose (stdout);
          }
        Exit (1);
} /* SigIntHandler */
/*---------------------------------------------------------------------------*/
static int ServerMainLoop ()
{
static  char                    func[] = "ServerMainLoop";
        int                     on = 1;
        struct sockaddr_in      sin;
        struct servent          *servEntry;
        int                     clientSocket;
        struct sockaddr_in      remoteAddr;
        int                     remoteAddrLen = sizeof (remoteAddr);

/*** initialize the daemon mode ***/
        switch (fork ()) {
          case -1:
            syslog (LOG_ERR, "%s: fork: %m", func);
            closelog ();
            exit (-1);
          case 0:
            break;
          default:
            closelog ();
            exit (0);
          }
        setsid ();
        chdir ("/");
        umask (0);
        syslog (LOG_DEBUG, "entering server mode...");
/*** lookup for our portno ***/
        if (portNo != -1) {
          syslog (LOG_DEBUG, "%s: using portnumber %d as specified", func, portNo);
          serverPort = htons (portNo);
          }
        else if ((servEntry = getservbyname (MUCS_SERVICE_NAME, "tcp")) == NULL) {
          syslog (LOG_WARNING, "%s: Warning, service %s not defiend, using default %d/tcp", func, MUCS_SERVICE_NAME, MUCS_PORT_NO);
          serverPort = htons (MUCS_PORT_NO);
          }
        else {
          serverPort = servEntry->s_port;                           /* is allready in net byte order! */
          }
/*** get the socket ***/
        if ((serverSocket = socket (AF_INET, SOCK_STREAM, 0)) == -1) {
          syslog (LOG_ERR, "%s: socket: %m", func);
          Exit (1);
          }
/*** reuse the address ***/
        if (setsockopt (serverSocket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof (on)) == -1) {
          syslog (LOG_ERR, "%s: setsockopt: %m", func);
          close (serverSocket);
          Exit (1);
          }
/*** bind the socket ***/
        memset (&sin, 0, sizeof (sin));
        sin.sin_family = AF_INET;
        sin.sin_len = sizeof (sin);
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = serverPort;
        if (bind (serverSocket, &sin, sizeof (sin)) == -1) {
          syslog (LOG_ERR, "%s: bind: %m", func);
          close (serverSocket);
          Exit (1);
```

```
          }
/*** signal the wait for connections ***/
        if (listen (serverSocket, 5) == -1) {
          syslog (LOG_ERR, "%s: listen: %m", func);
          close (serverSocket);
          Exit (1);
          }
/*** the main-loop ***/
        while (1) {
          if ((clientSocket = accept (serverSocket, (struct sockaddr*) &remoteAddr, &remoteAddrLen)) == -1) {
            if (errno == EINTR)
              continue;
            syslog (LOG_ERR, "%s: accept: %m", func);
            return 0;
            }
          switch (fork ()) {
            case -1:
              syslog (LOG_ERR, "%s: fork: %m", func);
              return -1;
/*** child serves the new connection ***/
            case 0:
              close (serverSocket);                           /* we don't need it */
              close (0);
              close (1);
              dup2 (clientSocket, 0);
              dup2 (clientSocket, 1);
              close (clientSocket);
              signal (SIGCHLD, SIG_DFL);
              ProcessRequest ();
              Exit (0);
              break;
/*** parrent waits for new connections ***/
            default:
              close (clientSocket);                           /* we don't need it any longer */
              openedConnections++;
              syslog (LOG_DEBUG, "%s: %d opend / %d closed", func, openedConnections, closedConnections);
            } /* switch */
          } /* while */
        return 0;
} /* ServerMainLoop */
/*-------------------------------------------------------------------------*/
static void Usage ()
{
        syslog (LOG_INFO, "usage: %s [-d debug-level] [-p portno] [-s]", PROG_NAME);
        Exit (1);
} /* Usage */
/*-------------------------------------------------------------------------*/
static void ParseOptions (int argc, char *argv)
{
        int                     c;

        while ((c = getopt (argc, argv, "d:p:s")) != -1)
          switch (c) {
            case 'd':
              debugLevel = atoi (optarg);
              break;
            case 'p':
              portNo = atoi (optarg);
              break;
            case 's':
              serverMode = 1;
              break;
            case '?':
              syslog (LOG_ERR, "Bad option [%c];", c);
              Usage ();
            }
        if (argc != optind)
          Usage ();
} /* ParseOption */
/*-------------------------------------------------------------------------*/
main (int argc, char *argv[])
{
        close (2);
        openlog (PROG_NAME, LOG_PID, logFacillity);
        syslog (LOG_DEBUG, "Started...");
        syslog (LOG_DEBUG, "Version: %s", rcsid);
        syslog (LOG_DEBUG, "Compiled:%s %s", __DATE__, __TIME__);
        signal (SIGINT, SigIntHandler);
        signal (SIGTERM, SigTermHandler);
        signal (SIGHUP, SigHupHandler);
        signal (SIGCHLD, SigChldHandler);
        ParseOptions (argc, argv);
#ifdef yyy
        if (getuid () != 0) {
          syslog (LOG_ERR, "CheckPermissions: sorry, but we have to be root!");
          Exit (1);
          }
#endif
/*** obtain our DNS domain name and our host name ***/
        res_init ();
        strcpy (myDomainName, _res.defdname);
        gethostname (myHostName, sizeof (myHostName)-1);
/*** do the work ***/
        if (serverMode)
          ServerMainLoop ();
        else
          ProcessRequest ();
        syslog (LOG_DEBUG, "Normal termination");
        Exit (0);
```

```
} /* main */
/*--------------------------------------------------------------------------*/
```

A.2 req.c

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           req.c
 *      written by:     Christian Zahl
 *      description:    handles new requests
 *
 * $Id: req.c,v 1.5 1995/11/09 09:35:30 czahl Exp czahl $
 *
 * $Log: req.c,v $
 * Revision 1.5  1995/11/09  09:35:30  czahl
 * - minor changes
 *
 * Revision 1.4  1995/10/22  14:18:10  czahl
 * *** empty log message ***
 *
 * Revision 1.3  1995/10/22  09:22:29  czahl
 * *** empty log message ***
 *
 * Revision 1.2  1995/10/10  11:47:05  czahl
 * - dummy version
 *
 * Revision 1.1  1995/09/30  15:15:01  czahl
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: req.c,v 1.5 1995/11/09 09:35:30 czahl Exp czahl $\n";

/*----- standard includefiles ----------------------------------------------*/
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <syslog.h>
#include <sys/select.h>
#include <stdarg.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/wait.h>
#include <pwd.h>
#include <netinet/in.h>

/*----- user includefiles --------------------------------------------------*/
#include "debug.h"

/*----- defines ------------------------------------------------------------*/
#define DEF_CALL_TIMEOUT        (3*60)
#define REQ_BUF_SIZE            (16*1024)

#define C_LF                    '\x0a'
#define C_CR                    '\x0d'

#define S_NONE                  NULL
#define S_I_HELLO               "101 Hi, I'm Fraggle, are you Gork?"
#define S_I_EXPANDING           "103 Expanding Username"
#define S_I_CHECK_ACCEPTANCE    "105 Checking Call Acceptance"
#define S_I_SEARCH_USER         "106 Searching User"
#define S_I_START_CC            "107 Calling CC"
#define S_I_CC_CONNECTED        "108 Connected To CC"
#define S_R_MOVED_PERM          "301 Permanent Forward"
#define S_R_MOVED_TEMP          "302 Temporary Forward"
#define S_R_LOCAL_REDIRECT      "303 Local Redirect"
#define S_C_BAD_REQ             "400 Bad Request"
#define S_C_NOT_FOUND           "404 Not Found"
#define S_C_UNKNOWN             "405 Unknown"
#define S_C_TIMEOUT             "410 Timeout"
#define S_S_INTERNAL            "500 Internal Server Error"
#define S_S_NOT_IMPLEMENTED     "501 Not Implemented"
#define S_S_LIMIT_EXCEEDED      "510 Limit Exceeded"
#define S_S_BAD_VERSION         "511 Version Unsupported"
#define S_S_START_CC            "512 Cannot Call CC"

#define REQ_CMD_CALL            "CALL"
#define REQ_MUCS_ID             "MUCS/"
#define REQ_MUCS_1_0            "MUCS/1.0"
#define RES_MUCS_1_0            REQ_MUCS_1_0

#define HUGE_LINE_LEN           (8*1024)                /* as in httpd from CERN */

#define EXPAND_SCRIPT           "/usr/local/mucs/etc/expand"
#define EXPAND_SCRIPT_NAME      "expand"
#define START_CC_SCRIPT         "/usr/local/mucs/etc/startCc"
#define START_CC_NAME           "startCc"
#define LOCATE_SCRIPT           "/usr/local/mucs/etc/locate"
#define LOCATE_SCRIPT_NAME      "locate"
        #define DEF_PATH                "/bin:/usr/bin:/usr/sbin:/etc:/usr/ucb"
/*----- type definitions ---------------------------------------------------*/
typedef struct _t_List {
        struct _t_List          *next;
        char                    data[1];
```

```
} t_List;
#define LIST_NEW(len)           malloc (sizeof (t_List) -1 +len)
#define LIST_DEL(lll)           {t_List* tmp; while (lll) {tmp=lll; \
                                lll=lll->next; free (tmp); } }
typedef t_Bool                  int

/*----- local functions ------------------------------------------------------*/

/*----- global functions -----------------------------------------------------*/

/*----- local variables ------------------------------------------------------*/
static  int                     callTimeout = DEF_CALL_TIMEOUT;
static  int                     timeout = 0;                    /* flag, set by signal handler */
static  char                    *req;
static  int                     reqLen;
static  char                    cmdStr[HUGE_LINE_LEN];
static  char                    argsStr[HUGE_LINE_LEN];
static  pid_t                   childPid;                       /* set by signal handler */
static  int                     childStatus;                    /* set by signal handler */
static  int                     version;                        /* obtained from the requestline, but not used */
static  char                    *expandScript = EXPAND_SCRIPT;
static  char                    *startCcScript = START_CC_SCRIPT;
static  char                    *locateScript = LOCATE_SCRIPT;
static  char                    *locateScriptName = LOCATE_SCRIPT_NAME;
static  struct passwd           *userEntry;
static  int                     socketForCc = -1;
static  char                    theDisplay[80] = ":0";          /* X Display of the called user */
static  pid_t                   ccPid;
static  int                     searchLocal = 0;                /* flag, search user on this host only */

/*----- global variables -----------------------------------------------------*/
extern  char                    myDomainName[];                 /* our local domain */
extern  char                    myHostName[];                   /* our local hostname */

/*----------------------------------------------------------------------------*/
static void SendToClient (char* fmt, ...)
{
        va_list                 ap;

        va_start (ap, fmt);
        vfprintf (stdout, fmt, ap);
        va_end (ap);
        fflush (stdout);
} /* SendToClient */
/*----------------------------------------------------------------------------*/
static void ServerStatus (int level, char *status, int final, char* fmt, ...)
/*
 * Logs the given fmt string with options into syslog...
 * Also sends a valid status line to stdout
 */
{
        char                    errMsg[255];
        va_list                 ap;

        va_start (ap, fmt);
        vsprintf (errMsg, fmt, ap);
        va_end (ap);
        syslog (LOG_ERR, errMsg);
        if (status) {
          fprintf (stdout, "%s %s\n", RES_MUCS_1_0, status);
          fflush (stdout);
          }
        if (final) {
          fprintf (stdout, "\n");
          fflush (stdout);
          }
} /* ServerStatus */
/*----------------------------------------------------------------------------*/
static void SigAlarmHandler ()
{
        timeout = 1;                                            /* set the flag */
} /* SigAlrmHandler */
/*----------------------------------------------------------------------------*/
static void SigChildHandler ()
{
        childPid = wait (&childStatus);
/*** if we are waiting for the CC to connect, close the socket ***/
        if (socketForCc != -1) {
          shutdown (socketForCc, 2);
          close (socketForCc);
          socketForCc = -1;                                     /* signal the accept call */
          }
        syslog (LOG_DEBUG, "SIGCHLD catched for pid %d with status %x", childPid, childStatus);
} /* SigChildHandler */
/*----------------------------------------------------------------------------*/
static void ForwardCall (int ccFd)
/*
 * Forwards the call directly to the local CC. Either when
 *      - the caller closes the connection
 *      - or the local CC closes the connection
 *      - or the connection timed out
 * then all connections will be closed.
 */
{
static  char                    func[] = "ForwardCall";
#define BUF_SIZE                1024
        char                    buf[BUF_SIZE];
        int                     n;
        fd_set                  readSet;
```

```
        int                     dataFromCc = 0;                 /* flag */

        ServerStatus (LOG_DEBUG, S_I_CC_CONNECTED, 0, "%s: forwarding call to CC", func);
/*** send the request to the CC ***/
        write (ccFd, req, reqLen);
/*** serve as proxy between caller and called CC ***/
        while (1) {
          FD_ZERO (&readSet);
          FD_SET (0, &readSet);
          FD_SET (ccFd, &readSet);
          if (select (ccFd+1, &readSet, NULL, NULL, NULL) == -1) {
            if (timeout)
              syslog (LOG_DEBUG, "%s: timeout", func);
            else
              syslog (LOG_ERR, "%s: socket: %m", func);
            break;
            } /* if */
/*** data from the caller ? ***/
          if (FD_ISSET (0, &readSet)) {
            if ((n = read (0, buf, BUF_SIZE)) <= 0) {
              syslog (LOG_DEBUG, "%s: caller closed connection", func);
              break;
              }
            if (write (ccFd, buf, n) != n) {
              syslog (LOG_ERR, "%s: unable to send all data to cc: %m", func);
              break;
              }
            }
/*** data from the cc ? ***/
          if (FD_ISSET (ccFd, &readSet)) {
            if ((n = read (ccFd, buf, BUF_SIZE)) <= 0) {
              syslog (LOG_DEBUG, "%s: called CC closed connection", func);
              break;
              }
            dataFromCc = 1;
            if (write (1, buf, n) != n) {
              syslog (LOG_ERR, "%s: unable to send all data to caller: %m", func);
              break;
              }
            }
          } /* while */
/*** close all connections to the local CC ***/
        shutdown (ccFd, 2);
        close (ccFd);
        if (!dataFromCc)
          if (timeout)
            ServerStatus (LOG_DEBUG, S_C_TIMEOUT, 1, "%s: timeout", func);
          else
            ServerStatus (LOG_DEBUG, S_S_START_CC, 1, "%s: CC terminates without saying hello :-)", func);
/*      kill (ccPid, SIGHUP); */
        return;
#undef BUF_SIZE
} /* ForwardCall */
/*-------------------------------------------------------------------------*/
static int StartCc ()
/*
 * returns:
 *      -1      error
 *      >=0     fd for communication with cc
 */
{
static  char                    func[] = "StartCc";
        struct sockaddr_in      sin;
        int                     sinLen;
        struct sockaddr         remoteAddr;
        int                     remoteAddrLen = sizeof (remoteAddr);
        int                     ccFd = -1;
        char                    portNoStr[10];
        char                    buf[4*1024];
        char                    *p;
        char                    *envv[10];
        int                     i;

        ServerStatus (LOG_DEBUG, S_I_START_CC, 0, "%s starting CC...", func);
/*** get a new listening socket and bind it ***/
        if ((socketForCc = socket (AF_INET, SOCK_STREAM, 0)) == -1) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: socket: %m", func);
          return -1;
          }
        memset (&sin, 0, sizeof (sin));
        sin.sin_family = AF_INET;
        sin.sin_len = sizeof (sin);
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = 0;
        if (bind (socketForCc, &sin, sizeof (sin)) == -1) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: bind: %m", func);
          close (socketForCc);
          return -1;
          }
        sinLen = sizeof (sin);
        if (getsockname (socketForCc, &sin, &sinLen)) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: getsockname: %m", func);
          close (socketForCc);
          return -1;
          }
        sprintf (portNoStr, "%d", ntohs (sin.sin_port));
        syslog (LOG_DEBUG, "%s: portno=%d", func, ntohs (sin.sin_port));
        if (listen (socketForCc, 1) == -1) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: listen: %m", func);
```

```
          close (socketForCc);
          return -1;
          }
/*** setup the environment for the child ***/
        i = 0;
        envv[i++] = p = buf;
        sprintf (p, "HOME=%s", userEntry->pw_dir);
        envv[i++] = p += strlen (p) +1;
        sprintf (p, "USER=%s", userEntry->pw_name);
        envv[i++] = p += strlen (p) +1;
        sprintf (p, "PATH=%s", DEF_PATH);
        envv[i++] = p += strlen (p) +1;
        sprintf (p, "DISPLAY=%s", theDisplay);
        envv[i++] = NULL;
/*** start the CC in the background ***/
        childPid = -1;                                          /* clean it! */
        signal (SIGCHLD, SigChildHandler);
        ccPid = fork ();
        if (ccPid == -1) {                                      /* unable to fork? */
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: fork: %%m", func);
          shutdown (socketForCc, 2);
          close (socketForCc);
          return -1;
          }
/*** we are the child, so start the CC ***/
        if (ccPid == 0) {
          if (setgid (userEntry->pw_gid))
            syslog (LOG_WARNING, "%s: warning: cannot setgid: %m", func);
          if (setuid (userEntry->pw_uid))
            syslog (LOG_WARNING, "%s: warning: cannot setuid: %m", func);
          if (setsid ())
            syslog (LOG_WARNING, "%s: warning: cannot setsid: %m", func);
          if (chdir (userEntry->pw_dir))
            syslog (LOG_WARNING, "%s: warning: cannot chdir: %m", func);
          close (0);
          close (1);
!!! 0 < /dev/null
!!! 1 > /dev/null
!!! 2 > /dev/null
          close (socketForCc);
          closelog ();
          execle (startCcScript, START_CC_NAME, portNoStr, NULL, envv);
          ServerStatus (LOG_ERR, S_NONE, 0, "%s: execl: %%m", func);
          exit (1);
          }
/*** wait for connection from CC ***/
        if ((ccFd = accept (socketForCc, &remoteAddr, &remoteAddrLen)) == -1) {
          if (timeout)
            ServerStatus (LOG_DEBUG, S_C_TIMEOUT, 1, "%s: timeout while waiting for CC", func);
          else if (((errno == EINTR) && (childPid == ccPid)) || (socketForCc == -1))
            ServerStatus (LOG_ERR, S_S_START_CC, 1, "%s: cannot start CC or CC terminated", func);
          else
            ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: accept: %%m", func);
          if (socketForCc != -1) {
            shutdown (socketForCc, 2);
            close (socketForCc);
            }
          kill (ccPid, SIGKILL);                        /* be sure */
          return -1;
          }
        shutdown (socketForCc, 2);
        close (socketForCc);
        return ccFd;
} /* StartCc */
/*-------------------------------------------------------------------------*/
static int CheckForCallAcceptance ()
/*
 * returns:
 *      -1      error, timeout, no interest
 *      0       ok, call accepted
 */
{
static  char                    func[] = "CheckForCallAcceptance";
        pid_t                   pid;

        ServerStatus (LOG_DEBUG, S_I_CHECK_ACCEPTANCE, 0, "%s checking for call acception", func);
        return 0;
#ifdef NEW
/*** start the script in the background ***/
        signal (SIGCHLD, SIG_DFL);
        pid = fork ();
        if (pid == -1) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: fork: %%m", func);
          return -1;
          }
/*** we are the child ***/
        if (pid == 0) {
          close (fd[0]);
          close (0);
setuid...
          signal (SIGTERM, SIG_DFL);
          signal (SIGINT, SIG_DFL);
          signal (SIGALRM, SIG_DFL);

env...
          execle (callAcceptanceScript, callAcceptanceScriptName, name, NULL, NULL);
          ServerStatus (LOG_ERR, S_NONE, 0, "%s: exec: %%m", func);
          exit (127);
          }
```

```
/*** wait for the script to terminate ***/
        if (wait (&status) == -1) {
          if (timeout)
            ServerStatus (LOG_DEBUG, S_C_TIMEOUT, 1, "%s: wait: timeout", func);
          else
            ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: wait: %%m", func);
          kill (pid, SIGKILL);
          wait (NULL);
          return -1;
          }
        syslog (LOG_DEBUG, "%s: child exited with status %x", func, status);
/*** check the current status ***/
        if (WIFSIGNALED (status)) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: child terminated by signal %d", func, WTERMSIG (status));
          return -1;
          }
        if ((WEXITSTATUS (status)) == 0) {
          ServerStatus (LOG_DEBUG, S_I_ACCEPTED, 0, "%s: call accepted", func);
          return 0;
          }
        if ((WEXITSTATUS (status)) == 1) {
          ServerStatus (LOG_DEBUG, S_C_NOT_FOUND, 1, "%s: rc=1, user should not be called", func);
          return -1;
          }
        if ((WEXITSTATUS (status)) != 0) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: unexpected retcode (%d) from child", func, WEXITSTATUS (status));
          return -1;
          }
        ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: oops, should never happen", func);
        return -1;
#endif
} /* CheckForCallAcceptance */
/*-------------------------------------------------------------------------*/
static char* LocateCalledUser (char* userName, int locateLocalOnly)
/*
 * "user" has to be the local user-name!
 * returns
 *      0       user found on this host
 *      -1      user not found on this host (redirects were sent)
 */
{
static  char                    func[] = "LocateCalledUser";
        int                     fd[2];
        FILE                    *f;
        char                    line[80];
        int                     n;
        t_Bool                  lineToLong;
        t_List                  *list;
        t_List                  *tmp;
        int                     locations;
        int                     status;
        pid_t                   pid;

        ServerStatus (LOG_DEBUG, S_I_SEARCH_USER, 0, "%s: searching user [%s]", func, userName);
        if (pipe (fd) == -1) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: pipe: %%m", func);
          return -1;
          }
/*** execute the script in the child ***/
        signal (SIGCHLD, SIG_DFL);
        pid = fork ();
        if (pid == -1) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: fork: %%m", func);
          close (fd[0]);
          close (fd[1]);
          return -1;
          }
/*** we are the child, so execute the script ***/
        if (pid == 0) {
/*!!! change uid ??? !!!!! */
          close (fd[0]);
          close (0);
          close (1);
          dup2 (fd[1], 1);
          close (fd[1]);
fd 0,2
          signal (SIGTERM, SIG_DFL);
          signal (SIGINT, SIG_DFL);
          signal (SIGALRM, SIG_DFL);
          if (locateLocalOnly)
            execle (locateScript, locateScriptName, "-l", userName, NULL, NULL);
          else
            execle (locateScript, locateScriptName, userName, NULL, NULL);
          syslog (LOG_ERR, S_NONE, "%s: exec: %%m", func);
          closelog ();
          exit (127);
          }
/*** catch the names send to stdout ***/
        close (fd[1]);
        if ((f = fdopen (fd[0], "r")) == NULL) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: fdopen: %%m", func);
          kill (pid, SIGKILL);
          waitpid (pid, NULL, 0);
          return -1;
          }
        lineToLong = FALSE;
        locations = 0;
        list = NULL;
        while (fgets (line, sizeof (line), f)) {                /* unitl eof or timeout */
```

```
          n = strlen (line);
          if (n <= 1)
            continue;
          if (line[n-1] != '\n') {
            lineToLong = TRUE;
            continue;
            }
          if (lineToLong) {                                   /* skip long lines */
            lineToLong = FALSE;
            continue;
            }
          line[--n] = '\0';                                   /* \n -> \0 */
          if ((tmp = LIST_NEW (n+1)) == NULL) {
            ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: malloc: %%m", func);
            kill (pid, SIGKILL);
            waitpid (pid, NULL, 0);
            fclose (f);
            LIST_DEL (list);
            return -1;
            }
          strcpy (tmp->data, line);
          tmp->next = list;
          list = tmp;
          locations++;
          } /* while */
        fclose (f);
        kill (pid, SIGKILL);                                  /* make sure not having zombies... */
        waitpid (pid, &status, 0);
        syslog (LOG_DEBUG, "%s: child exited with status %x", func, status);
/*** check the current status ***/
        if (timeout) {
          ServerStatus (LOG_DEBUG, S_C_TIMEOUT, 1, "%s: timeout", func);
          LIST_DEL (list);
          return -1;
          }
/*** child killed? ***/
        if (WIFSIGNALED (status)) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: child terminated by signal %d", func, WTERMSIG (status));
          LIST_DEL (list);
          return -1;
          }
/*** user not found? ***/
        if (((WEXITSTATUS (status)) == 1) || (locations <= 0)) {
          ServerStatus (LOG_DEBUG, S_C_NOT_FOUND, 1, "%s: rc=1, user [%s] not found", func, userName);
          LIST_DEL (list);
          return -1;
          }
/*** error ? ***/
        if ((WEXITSTATUS (status)) != 0) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: unexpected retcode (%d) from child", func, WEXITSTATUS (status));
          LIST_DEL (list);
          return -1;
          }
/*** expands if more then one ***/
        if ((locations > 1) || strcmp (list->data, myHostName))  {
          ServerStatus (LOG_DEBUG, S_R_LOCAL_REDIRECT, 0, "%s: user [%s] is not sitting at this host", func, userName);
          while (list) {
            SendToClient ("Location: %s@%s.%s\n", userName, list->data, myDomainName);
            tmp = list;
            list = list->next;
            free (tmp);
            } /* while */
          SendToClient ("\n");
          return -1;
          }
/*** we don't need the list, because we know that the user is sitting at this host ***/
        LIST_DEL (list);
        syslog (LOG_DEBUG, "%s: user [%s] found on this host", func, userName);
        return 0;
} /* LocateCalledUser */
/*---------------------------------------------------------------------------*/
static char* ExpandCalledUserName (char* nameToExpand)
/*
 * Expands the given name into the real usernames or forward mail addresses.
 * Handles all kind of errors and returns only one user-name! Note that the
 * char* returned is static and will be overwritten by every call, and
 * should never be freed.
 * Returns:
 *      NULL    error
 *      name    expanded name
 * LIMITS:
 *      Currently only handles local names without canonical representation
 *              user
 *              user@domain
 *      but not
 *              canonical name <user>
 *              user (canonical name)
 */
{
static  char                    func[] = "ExpandCalledUserName";
static  char                    name[256];                  /* the char* returned */
        pid_t                   pid;
        int                     status;
        int                     fd[2];
        t_List                  *tmp;
        t_List                  *old;
        t_List                  *list = NULL;
        int                     names = 0;
        char                    *p;
```

```
        char                    c;
        int                     n;

        ServerStatus (LOG_DEBUG, S_I_EXPANDING, 0, "%s: expanding %s", func, nameToExpand);
        if (pipe (fd) == -1) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: pipe: %%m", func);
          return NULL;
          }
        strcpy (name, nameToExpand);
        if (p = strchr (name, '@'))
          *p = '\0';
/*** execute the script in the child ***/
        childPid = -1;                                          /* clean it */
        signal (SIGCHLD, SIG_DFL);
        pid = fork ();
        if (pid == -1) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: fork: %%m", func);
          return NULL;
          }
        if (pid == 0) {                                         /* we are the child */
          close (fd[0]);
          close (0);
          close (1);
          dup2 (fd[1], 1);
          close (fd[1]);
          signal (SIGTERM, SIG_DFL);
          signal (SIGINT, SIG_DFL);
          signal (SIGALRM, SIG_DFL);
          execle (expandScript, expandScript, name, NULL, NULL);
          ServerStatus (LOG_ERR, S_NONE, 0, "%s: exec: %%m", func);
          exit (2);
          }
/*** catch the names send to stdout ***/
        close (fd[1]);
        while (read (fd[0], &c, 1) == 1) {
          if (c == '\n') {
            if (n < sizeof (name) -1)  {
              name[n++] = '\0';
              tmp = LIST_NEW (n);
              tmp->next = list;
              strcpy (tmp->data, name);
              list = tmp;
              n = 0;
              names++;
              }
            }
          else if (n < sizeof (name) -1)
            if (c != ' ' && c != '\t' && c != '\r')
              name[n++] = c;
          } /* while */
        close (fd[0]);
        kill (pid, SIGKILL);                                    /* make sure don't haveing zombies */
        waitpid (pid, &status, 0);
        syslog (LOG_DEBUG, "%s: child exited with status %x", func, status);
/*** check the current status ***/
        if (timeout) {
          ServerStatus (LOG_DEBUG, S_C_TIMEOUT, 1, "%s: timeout", func);
          LIST_DEL (list);
          return NULL;
          }
        if (WIFSIGNALED (status)) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: child terminated by signal %d", func, WTERMSIG (status));
          LIST_DEL (list);
          return NULL;
          }
        if ((WEXITSTATUS (status)) == 1) {
          ServerStatus (LOG_DEBUG, S_C_UNKNOWN, 1, "%s: rc=1, cannot expand [%s]", func, nameToExpand);
          LIST_DEL (list);
          return NULL;
          }
        if ((WEXITSTATUS (status)) != 0) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: unexpected retcode (%d) from child", func, WEXITSTATUS (status));
          LIST_DEL (list);
          return NULL;
          }
/*** name unknown? ***/
        if (names <= 0) {
          ServerStatus (LOG_DEBUG, S_C_UNKNOWN, 1, "%s: names=0, cannot expand [%s]", func, nameToExpand);
          LIST_DEL (list);
          return NULL;
          }
/*** expands to more then one ***/
        if (names > 1) {
          ServerStatus (LOG_DEBUG, S_R_MOVED_TEMP, 0, "%s: name [%s] expands to %d names", func, nameToExpand, names);
          tmp = list;
          while (tmp != NULL) {
            SendToClient ("Location: %s\n", tmp->data);
            old = tmp;
            tmp = tmp->next;
            free (old);
            } /* while */
          SendToClient ("\n");
          return NULL;
          }
/*** now we don't need the list, because the only name is still in "name" ***/
        LIST_DEL (list);
/*** in our local domain ***/
        if (!(p = strchr (name, '@')))                  /* user */
          return name;
```

```
        if (!stricmp (p+1, myDomainName))               /* user@domain */
          return name;
        if ((p = strchr (p, '.'))
                && !(stricmp (p+1, myDomainName))) {    /* user@host.domain */
          return name;
          }
/*** not in our domain ***/
        ServerStatus (LOG_DEBUG, S_R_MOVED_PERM, 0, "%s: user [%s] moved to [%s]", func, nameToExpand, name);
        SendToClient ("Location: %s\n", name);
        SendToClient ("\n");
        return NULL;
} /* ExpandCalledUserName */
/*---------------------------------------------------------------------------*/
static void ProcessCall (char* cmd, char* user)
{
static  char                    func[] = "ProcessCall";
        char                    *name;
        char                    userName[100];
        int                     ccFd;
        char                    *p;

/*** 1. expand the user name, and check if the name is ok ***/
        if ((name = ExpandCalledUserName (user)) == NULL)
          return;
        strncpy (userName, name, sizeof (userName)-1);
        if (p = strchr (userName, '@'))
          *p = '\0';
        if (!(userEntry = getpwnam (userName))) {
          ServerStatus (LOG_DEBUG, S_C_UNKNOWN, 1, "%s: the expanded user [%] is unknown", func, name);
          return;
          }
/*** 2. locate the user ***/
        if (LocateCalledUser (name, searchLocal))
          return;
/*** 3. check the call ***/
        if (CheckForCallAcceptance (name))
          return;
/*** 4. signal the call ***/
        if ((ccFd = StartCc (name)) == -1)
          return;
/*** 5. serve as proxy ***/
        ForwardCall (ccFd);
} /* ProcessCall */
/*---------------------------------------------------------------------------*/
static char* SkipWhiteSpaces (char* p)
{
/*** skip any leading whitespaces ***/
        while (1) {
          if (*p == '\0')
            break;
          if ((*p != ' ') && (*p != '\t'))
            break;
          p++;
          }
        return p;
} /* SkipWhiteSpaces */
/*---------------------------------------------------------------------------*/
static char* GetField (char* p, char* str, int len)
{
/*** skip any leading whitespaces ***/
        p = SkipWhiteSpaces (p);
/*** read up to next whitespace or eol ***/
        while (len-- > 0) {
          if (*p == '\0')
            break;
          if (*p == ' ')
            break;
          if (*p == '\t')
            break;
          if (*p == '\n')
            break;
          *str++ = *p++;
          }
        if (len <= 0)
          return NULL;
        *str = '\0';
        return p;
} /* GetField */
/*---------------------------------------------------------------------------*/
static int CheckRequestLine ()
/*
 * Returns:
 *      0       ok
 *      -1      error, which is allready signaled.
 */
{
static  char                    func[] = "CheckRequestLine";
        char                    *p;
        char                    versionStr[HUGE_LINE_LEN];

/*** split the requestline ***/
        p = req;
        if (((p = GetField (p, cmdStr, sizeof (cmdStr) -1)) == NULL)
                || ((p = GetField (p, argsStr, sizeof (argsStr) -1)) == NULL)
                || ((p = GetField (p, versionStr, sizeof (versionStr) -1)) == NULL)) {
          ServerStatus (LOG_DEBUG, S_C_BAD_REQ, 1, "%s: bad requestline", func);
          return -1;
          }
        p = SkipWhiteSpaces (p);
```

```
        if (*p != C_LF) {
          ServerStatus (LOG_DEBUG, S_C_BAD_REQ, 1, "%s: bad requestline", func);
          return -1;
          }
/*** check the protocoll and version ***/
        if (strnicmp (versionStr, REQ_MUCS_ID, sizeof (REQ_MUCS_ID)-1) != 0) {
          ServerStatus (LOG_DEBUG, S_C_BAD_REQ, 1, "%s: invalid protocoll: [%s]", func, versionStr);
          return -1;
          }
        if (!stricmp (versionStr, REQ_MUCS_1_0))               /* its MUCS/1.0? */
          version = 1000;                                      /* save the version */
        else {                                                 /* unknown version? */
          ServerStatus (LOG_DEBUG, S_S_BAD_VERSION, 1, "%s: invalid version: %s", func, versionStr);
          return -1;
          }
/*** ***/
        return 0;
} /* CheckRequestLine */
/*------------------------------------------------------------------------*/
static char* GetRequest ()
/*
 *      stdin / 0       data received from the caller
 *      stdout / 1      data send to the caller
 * Returns
 *      -1              error, timeout etc.
 *      0               ok, request is in req
 */
{
static  char                    func[] = "GetRequest";
        t_Bool                  lastWasLF;                     /* flag */
        int                     i;

/*** get some memory for holding the request ***/
        if ((req = malloc (REQ_BUF_SIZE)) == NULL) {
          ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: malloc: %m", func);
          return -1;
          }
/*** receive the whole request ***/
        lastWasLF = FALSE;
        reqLen = 0;
        while (!timeout) {                                     /* until timeout */
          i = read (0, &req[reqLen], 1);
          if (i == -1)                                         /* error? */
            if (errno == EINTR)
              continue;
            else {
              ServerStatus (LOG_ERR, S_S_INTERNAL, 1, "%s: read: %m", func);
              free (req);
              req = NULL;
              return -1;
              }
          if (i == 0) {                                        /* connection closed? */
            ServerStatus (LOG_DEBUG, S_C_BAD_REQ, 1, "%s: connection closed by peer", func);
            free (req);
            req = NULL;
            return -1;
            }
          if (req[reqLen] == C_CR)                             /* skip CR */
            continue;
          if (++reqLen >= REQ_BUF_SIZE) {
            ServerStatus (LOG_DEBUG, S_S_LIMIT_EXCEEDED, 1, "%s: request overflow", func);
            free (req);
            req = NULL;
            return -1;
            }
          if (req[reqLen-1] == C_LF)
            if (lastWasLF)
              break;
            else
              lastWasLF = TRUE;
          else
            lastWasLF = FALSE;
          } /* while */
/*** post checks ***/
        if (timeout) {
          ServerStatus (LOG_DEBUG, S_C_TIMEOUT, 1, "%s: timeout", func);
          free (req);
          req = NULL;
          return -1;
          }
        req[reqLen] = '\0';                            /* terminate, it easier to deal with */
        return 0;
} /* GetRequest */
/*------------------------------------------------------------------------*/
static void HandleRequest ()
{
static  char                    func[] = "HandleRequest";

        ServerStatus (LOG_DEBUG, S_I_HELLO, 0, "%s sending greeting", func);
        if (GetRequest ())
          return;
        if (CheckRequestLine ())
          return;
/*** ***/
        if (!stricmp (cmdStr, REQ_CMD_CALL))
          ProcessCall (cmdStr, argsStr);
        else
          ServerStatus (LOG_DEBUG, S_C_BAD_REQ, 1, "%s: invalid command", func);
} /* HandleRequest */
```

```
/*------------------------------------------------------------------------*/
void ProcessRequest ()
/*
 * This is the main entry for serving any kind of new client connection.
 *      stdin / 0       data received from the caller
 *      stdout / 1      data send to the caller
 *      stderr / 2      UNDEFINED but not closed, and should never be used!
 */
{
/*** for timeout checks ***/
        timeout = FALSE;                                        /* clear the flag */
        signal (SIGALRM, SigAlarmHandler);                      /* install the timeout handler */
        alarm (callTimeout);                                    /* start the timer */
/*** do the work ***/
        HandleRequest ();
        if (req != NULL)
          free (req);
/*** shutdown all the open stuff ***/
        alarm (0);                                              /* disable timeout */
        shutdown (0, 1);                                        /* ugly because it could be a pipe... */
        shutdown (1, 1);
        close (0);
        close (1);
        return;
} /* ProcessRequest */
/*------------------------------------------------------------------------*/
```

Anhang B

Quelle-Code der Shell-Scripts

B.1 expand

```
#!/bin/ksh
#
# Shell script to expand a call name to one ore more email adresses
# and / or usernames.
# - Expects only the email-address part, not the comment!
# - All names SHOULD be lower-case, because we check case sensitive!
#-----------------------------------------------------------------
# Configureation section
exec 2>/dev/null
PATH=/bin:/usr/bin:/usr/ucb
MUCS_HOME=/usr/local/mucs
MUCS_ALIASES=$MUCS_HOME/etc/aliases
MAIL_EXCHANGE=becks
MAIL_ALIASES=/etc/aliases
NIS_PASSWD=passwd
PASSWD=/etc/passwd
DEBUG=false
#-----------------------------------------------------------------
Debug ()
{
        if $DEBUG; then
                echo $* >&2
        fi
}
#-----------------------------------------------------------------
# ExpandMucsAlises <name>
# checks local alias listings
#       0       found,
#       1       not found
ExpandMucsAliases ()
{
        test "$MUCS_ALIASES" || return 1
        test ! -r "$MUCS_ALIASES" || return 1
        found=""
        for i in `egrep -i "^$1:" $MUCS_ALIASES`; do
                test "$i" = "$1:" && continue
                found=yes
                echo $i
        done
        test "$found" || return 1
        return 0
}
#-----------------------------------------------------------------
# Trys to expand alises via SMTP
ExpandSmtpAliases ()
{
#       (echo "EXPN $1"; echo "QUIT") | telnet $MAIL_EXCHANGE smtp
        return 1
}
#-----------------------------------------------------------------
# Expands Alises via local mail alises file
ExpandMailAliases ()
{
        test "$MAIL_ALIASES" || return 1
        test ! -r "$MAIL_ALIASES" || return 1
        rc="1"
        for i in `egrep -i "^$1:" $MAIL_ALIASES|sed -e 's/,/ /g'`; do
                test "$i" = "$1:" && continue
                rc=0
                echo $i
        done
        return $rc
}
#-----------------------------------------------------------------
# Expands the name using the local passwd file
ExpandPasswd ()
{
```

```
        test "$PASSWD" || return 1
        test ! -r "$PASSWD" || return 1
        egrep "^$1:" $PASSWD >/dev/null && echo $1 && return 0
        rc=1
        for n in `sed -e 's/\:/ /g' $PASSWD | grep -i " $1 " |
        awk '{print $1}'`; do
                echo $n
                rc=0
        done
        return $rc
}
#---------------------------------------------------------------
# Expands the name using NIS passwd
ExpandNisPasswd ()
{
        test "$NIS_PASSWD" || return 1
        ypcat $NIS_PASSWD | egrep "^$1:" >/dev/null && echo $1 && return 0
        rc=1
        for n in `ypcat $NIS_PASSWD | awk '{FS=":"; print $1 " " $5}' |
        grep -i "$1" | awk '{print $1}'`; do
                echo $n
                rc=0
        done
        return $rc
}
#---------------------------------------------------------------
#main ()
        NAME=$1
        if [ "@$NAME" = "@" ]; then
                exit 1
        fi
Debug "\tNAME=$NAME"
Debug "\tmucs aliases..."
        ExpandMucsAliases "$NAME" && exit 0
Debug "\tsmtp aliases..."
        ExpandSmtpAliases "$NAME" && exit 0
Debug "\tmail aliases..."
        ExpandMailAliases "$NAME" && exit 0
Debug "\tpasswd..."
        ExpandPasswd "$NAME" && exit 0
Debug "\tnis passwd..."
        ExpandNisPasswd "$NAME" && exit 0
Debug "\tnot found!"
        exit 1
#---------------------------------------------------------------
```

B.2 checkCall

```
#!/bin/sh
#
# This script has to checks the call. It should terminate with
#       0       to accept the call
#       1       to reject it
###############################################################################
exec 2>/dev/null

#------------------------------------------------------------------------------
# First get the request it-self
cat > /dev/null

#------------------------------------------------------------------------------
# No accept the call
exit 0
```

B.3 locate

```
#!/bin/sh -f
#
# Locates the given user in the local domain.
# Usage: locate <user> [-l]
#       <user>  UNI* user-name of user to locate
#       -l      look only at this host
# Returns
#       0       user found, locations written to stdout
#       1       not found
#       2       error
# $Log$
#------------------------------------------------------------------------------
# setup
PATH=/bin:/usr/bin:/usr/ucb:
#------------------------------------------------------------------------------
# check the options
ParseOptions ()
{
        while [ $# -gt 0 ]; do
                case $1 in
                        -l)     LOCATE_LOCAL=true;;
                        *)      USER_NAME=$1;;
                esac
```

```
                shift
        done
        if [ "@$USER_NAME" = "@" ]; then
                exit 2
        fi
}
#------------------------------------------------------------------------
# ask the location-server if not local only
AskLocationServer ()
{
        test $LOCATE_LOCAL && return
        found=""
        LC_ALL=C; export LC_ALL
        for host in `rwho -a \
                | egrep "^$USER_NAME " \
                | sort +5 \
                | egrep 'console|hft/0' \
                | sed -e 's/:/ /' \
                | awk '{print $2}' \
                | sort -u`
        do
                echo $host
                found=0
        done
        test $found && exit 0
}
#------------------------------------------------------------------------
# look at the local host
SearchLocal ()
{
        for tty in `who | awk "/^$USER_NAME /"'{print $2}'`; do
                case $tty in
                        console | hft/0)
                                echo `hostname`; exit 0 ;;
                esac

        done
        local=`who | awk "/^$USER_NAME /"'{if ($6 == "") print "local"}'`
        if [ "@$local" != "@" ]; then
                echo `hostname`
                exit 0
        fi
}
#------------------------------------------------------------------------
#main
        ParseOptions $*
        AskLocationServer
        SearchLocal
        exit 1
### EOF ###
```

B.4 startCc

```
#!/bin/ksh -f
#
# This script starts the conference controler (CC)
# 1995/10/16 V1
#-----------------------------------------------------------------------
# $1    portno for the CC to connect to
#-----------------------------------------------------------------------
PATH=/bin:/usr/bin:/usr/ucb:.
LOG=$HOME/.mucs-log
exec 0</dev/null
exec 1>$LOG
exec 2>&1
#$LOG

#trap 'rm $TMP' 0

PORT_NO=$1

MUCS_HOME=/usr/local/mucs
MUCS_CC=$MUCS_HOME/bin/coco
MUCS_PERSONAL_CC=$HOME/.mucs-startCc
MUCS_ADMIN=czahl
ARGS="$*"
#-----------------------------------------------------------------------
Log ()
{
        echo $* >&2
}
#-----------------------------------------------------------------------
# Notify <user> <greetings>
Notify ()
{
        sendmail $1 << EOF-EOF-EOF
Subject: MUCS Config Error
From: The MUCS system

*** THIS IS AN AUTOMATIC GENERATED MAIL FROM THE MUCS SYSTEM, DON'T REPLY. ***

Dear $2,

I received a call for the user "$USER" and I wasn't able to start the
```

```
MUCS environment to inform the user of the call. Neither the personal
setting, nor the system default settings worked at this time. Please
take a look at the settings, because we want to use the MUCS system
in the future also. If you cannot detect the cause for the error, or
if you have found a new bug, please contact one of the following MUCS
develloper:
        mailto:zahl@fokus.gmd.de (Christian Zahl)
        mailto:schulzrinne@fokus.gmd.de (Henning Schulzrinne)

Here are some more detailed informations about the variables:
        MUCS_HOME=$MUCS_HOME
        MUCS_CC=$MUCS_CC
        MUCS_PERSONAL_CC=$MUCS_PERSONAL_CC
        HOST=`hostname`
The environment:
`for x in \`env\`; do
echo "\\t\$x"
done`
Commandline options:
        $ARGS
The following errors were generated:
*******************************************************************************
`cat $TMP`
*******************************************************************************
Thanks for your help,
the MUCS system
EOF-EOF-EOF
}
#------------------------------------------------------------------
# Try to start the personal setup
if [ -r $MUCS_PERSONAL_CC ]; then
        $MUCS_PERSONAL_CC $PORT_NO
        rc=$?
        if [ $rc != 0 ]; then
                Log "Error while starting $MUCS_PERSONAL_CC"
                Log "Returncode: $rc"
                Notify $USER "MUCS User"
                exit 1
        else
                exit 0
        fi
fi
Log "Warning: no personal startup script found."
#------------------------------------------------------------------
# Use the site global setting for starting the MUCS env
if [ -x $MUCS_CC ]; then
        $MUCS_CC $PORT_NO
        rc=$?
        if [ $rc != 0 ]; then
                Log "Error while starting $MUCS_CC"
                Log "Returncode: $rc"
                Notify $MUCS_ADMIN "MUCS Admin"
                exit 1
        else
                exit 0
        fi
fi
Log "Error: no way found to startup the MUCS system."
Notify $MUCS_ADMIN "MUCS Admin"
exit 1
```

Anhang C

Source-Code für den LocationService

C.1 lswhod.c

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           lswhod.c
 *      written by:     Christian Zahl
 *      description:    the whod for the location server
 *
 * $Id: lswhod.c,v 1.2 1995/11/19 14:16:22 czahl Exp $
 *
 * $Log: lswhod.c,v $
 * Revision 1.2  1995/11/19  14:16:22  czahl
 * - first official version for AIX and Solaris
 *
 * Revision 1.1  1995/11/11  14:46:31  cz
 * Initial revision
 *
 */
static  char                    rcsid[] = "$Id: lswhod.c,v 1.2 1995/11/19 14:16:22 czahl Exp $\n";

/*----- standard includefiles ---------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <sys/signal.h>
#include <fcntl.h>
#include <sys/types.h>
#include <dirent.h>
#ifdef USE_UTMPX
#  include <utmpx.h>
#else
#  include <utmp.h>
#endif
#include <limits.h>
#include <sys/utsname.h>

/*----- user includefiles -------------------------------------------------*/

/*----- defines -----------------------------------------------------------*/
#ifndef DEF_DB_DIR
#  ifdef _AIX
#    define DEF_DB_DIR          "/usr/local/mucs/ls-db"
#  else
#    define DEF_DB_DIR          "/net/u1/cz/mucs/ls-db"
#  endif
#endif
#ifndef DEF_UPDATE_INTERVAL
#  define DEF_UPDATE_INTERVAL   (3*60)
#endif
#define SERVER_NAME             "who"
#ifdef WITH_DEBUG
#  define DEBUG(xxx)            xxx
#else
#  define DEBUG(xxx)            {}
#endif
#ifndef TRUE
#define TRUE                    1
#define FALSE                   0
#endif

/*----- type definitions --------------------------------------------------*/
typedef int                     t_Bool;
typedef struct _t_info {
        struct _t_info          *next;                  /* chained list */
        t_Bool                  updated;                /* flag if changed */
        char                    id[1];                  /* placeholder for item */
} t_Info;

/*----- local functions ---------------------------------------------------*/
static  void                    CleanDatabase ();
```

```
/*----- global functions ----------------------------------------------------*/

/*----- local variables -----------------------------------------------------*/
static  char                    myHostName[256];
static  char                    *serverName;
static  int                     updateInterval = DEF_UPDATE_INTERVAL;
static  char                    *dbDir = DEF_DB_DIR;
static  t_Info                  *infos = NULL;

/*----- global variables ----------------------------------------------------*/

/*---------------------------------------------------------------------------*/
SigHandler ()
{
        CleanDatabase ();
        exit (0);
} /* SigHandler */
/*---------------------------------------------------------------------------*/
char *Id2Fn (char *id)
{
static  char                    tmp[PATH_MAX];
        char                    *fn;

        fn = tmp;
        while (*id) {
          if ((*id == '/') || (*id == '%')) {
            sprintf (fn, "%%%02x", *id++);
            fn += 3;
            }
          else
            *fn++ = *id++;
          }
        *fn = '\0';
        return tmp;
} /* Id2Fn */
/*---------------------------------------------------------------------------*/
static void UpdateDatabase ()
{
        DIR                     *dir;
        struct dirent           *dirent;
        char                    id[256];
        char                    fn[PATH_MAX +1];
#ifdef USE_UTMP
        struct utmpx            *utmp;
#else
        struct utmp             *utmp;
#endif
        char                    userName[sizeof (utmp->ut_user)+1];
        char                    lineName[sizeof (utmp->ut_line)+1];
        char                    hostName[sizeof (utmp->ut_host)+1];
        t_Info                  *newInfos = NULL;
        t_Info                  *tmp;
        t_Info                  *last;
        int                     fd;

        DEBUG (printf ("Updating the database...\n");)
/*** read all user utmp entries ***/
        while (1) {
#ifdef USE_UTMPX
          if ((utmp = getutxent ()) == NULL)
#else
          if ((utmp = getutent ()) == NULL)
#endif
            break;
          if (utmp->ut_type != USER_PROCESS)
            continue;
          memcpy (userName, utmp->ut_user, sizeof (utmp->ut_user));
          userName[sizeof (utmp->ut_user)] = '\0';
          memcpy (lineName, utmp->ut_line, sizeof (utmp->ut_line));
          lineName[sizeof (utmp->ut_line)] = '\0';
          memcpy (hostName, utmp->ut_host, sizeof (utmp->ut_host));
          hostName[sizeof (utmp->ut_host)] = '\0';
/*** build up the id string ***/
          strcpy (id, serverName);
          strcat (id, "|host="); strcat (id, myHostName);
          strcat (id, "|user="); strcat (id, userName);
          strcat (id, "|line="); strcat (id, lineName);
          if (hostName[0] != '\0') {
            strcat (id, "|remote=");
            strcat (id, hostName);
            if (strlen (hostName) >= sizeof (utmp->ut_host))
              strcat (id, "...");
            }
          strcat (id, "|");
          DEBUG (printf ("id=%s\n", id);)
/*** search the id in the old infos ***/
          for (last = NULL, tmp = infos; tmp != NULL; last = tmp, tmp = tmp->next)
            if (!strcmp (id, tmp->id))
              break;
/*** id found, remove from old infos ***/
          if (tmp != NULL) {
            if (last == NULL)
              infos = tmp->next;
            else
              last->next = tmp->next;
            tmp->updated = FALSE;
            }
/*** id not found, create new item ***/
```

```
          else {
            if ((tmp = malloc (sizeof (t_Info) -1 + strlen (id) +1)) == NULL) {
              DEBUG (perror ("!!!malloc:");)
              return;
              }
            strcpy (tmp->id, id);
            tmp->updated = TRUE;
            }
/*** insert in new list ***/
          tmp->next = newInfos;
          newInfos = tmp;
          } /* while */
#ifdef USE_UTMPX
        endutxent ();
#else
        endutent ();
#endif
/*** remove the old DB entries and items ***/
        for (tmp = infos; tmp != NULL; tmp = tmp->next) {
          last = tmp;
          sprintf (fn, "%s/%s", dbDir, Id2Fn (tmp->id));
          DEBUG (printf ("removing old entry %s\n", tmp->id));
          unlink (fn);
          tmp = tmp->next;
          free (last);
          }
/*** create the updated information DB entries ***/
        for (tmp = newInfos; tmp != NULL; tmp = tmp->next) {
          if (!tmp->updated)
            continue;
          sprintf (fn, "%s/%s", dbDir, Id2Fn (tmp->id));
          DEBUG (printf ("creating new entry %s\n", tmp->id);)
          fd = creat (fn, 0644);
          if (fd == -1)
            DEBUG (perror ("!!! creat:");)
          else
            close (fd);
          }
        infos = newInfos;
        DEBUG (printf ("Done!\n");)
} /* CleanDatabase */
/*-------------------------------------------------------------------------*/
static void CleanDatabase ()
{
        DIR                     *dir;
        struct dirent           *dirent;
        char                    hostId[256];

        DEBUG (printf ("Cleaning database...\n");)
        if ((dir = opendir (dbDir)) == NULL)
          return;
        sprintf (hostId, "|host=%s|", myHostName);
        for (dirent = readdir (dir); dirent != NULL; dirent = readdir (dir))
          if (strstr (dirent->d_name, hostId)
          && !strncmp (dirent->d_name, serverName, strlen (serverName)))
            unlink (dirent->d_name);
        closedir (dir);
        DEBUG (printf ("Done!\n");)
} /* CleanDatabase */
/*-------------------------------------------------------------------------*/
Init ()
{
        struct utsname          utsname;

        DEBUG (printf ("Initalizing...\n");)
        signal (SIGTERM, SigHandler);
        signal (SIGINT, SigHandler);
        signal (SIGQUIT, SigHandler);
        signal (SIGHUP, SigHandler);
        uname (&utsname);
        strcpy (myHostName, utsname.nodename);
        serverName = SERVER_NAME;
        chdir ("/");
        DEBUG (printf ("Done!\n");)
} /* Init */
/*-------------------------------------------------------------------------*/
main ()
{
        Init ();
        CleanDatabase ();
        while (1) {
          UpdateDatabase ();
          sleep (updateInterval);
          }
} /* main */
/*-------------------------------------------------------------------------*/
```

C.2 Makefile

```
#############################################################################
#               optional settable defines
# WITH_DEBUG
#       prints some usefull debug informations
# DEF_DB_DIR
#       the directory where the DB resides
```

```
# DEF_UPDATE_INTERVAL
#       interval in which the server updates the DB

##############################################################################
# USE_UTMPX
#       use utmpx function group insteat of utmp

##############################################################################
#               AIX 3.2.5
DEFS=   -DWITH_DEBUG

##############################################################################
#               Solaris
# DEFS= -DUSE_UTMPX
# CC=   gcc

lswhod:         lswhod.c
        $(CC) -o lswhod lswhod.c $(DEFS)
```

Anhang D

Source-Code für NeVoT

D.1 netutil/ibm_audio.c

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           ibm_audio.c
 *      creation date:  06-Mar-1995
 *      last update:    <obsolete>
 *      written by:     Frank Oertel, Christian Zahl
 *      description:    Device dependent part of AUDIO support for the
 *                      IBM ULTIMEDIA Adapter
 *
 * $Id: ibm_audio.c,v 1.1 1995/08/17 13:58:26 hgs Exp $
 *
 * $Log: ibm_audio.c,v $
 * Revision 1.1  1995/08/17  13:58:26  hgs
 * Initial revision
 *
 * Revision 1.18  1995/06/30  16:17:18  czahl
 * - bug: while searching for an audio device filename, the comparison
 *   was wrong! Result: no audio :-(
 *
 * Revision 1.17  1995/06/28  11:40:27  czahl
 * - minor changes so that it can be compiled in other environments than
 *   mine.
 *
 * Revision 1.16  1995/06/27  15:21:51  czahl
 * - After a long time of changes, trail and so on, we are now in the
 *   happily position to make this release to an alpha version!!!
 *
 * Revision 1.15  1995/06/25  16:19:06  czahl
 * - alpha 0. Before we begin to support the audioctl tcl extension
 *   via ahw_set, we wonna save this version.
 *
 * Revision 1.14  1995/06/22  19:52:12  czahl
 * *** empty log message ***
 *
 * Revision 1.13  1995/06/19  12:23:27  czahl
 * - now after we got some very helpfull hints from Paul Wadehra, we
 *   implemented some new sets of commands for setting various audio
 *   parameters. This was done for testing purposes only because it was
 *   easier to do this rather than changing ahw_set.
 * - Because we now we know how to deal with the audio device, we will
 *   switch back to use ahw_set for changing the audio parameters.
 *   Just for commenting, we save this nice version...
 *
 * Revision 1.12  1995/06/12  10:42:44  czahl
 * - after a long period of testing and understanding the behavior of
 *   the UMS audio adapter, we decided to support just a relatively
 *   poor interface. Before doing so, we want to save this bloody
 *   version first...
 *
 * Revision 1.11  1995/05/10  15:13:10  czahl
 * - some minor changes
 *
 * Revision 1.10  1995/04/21  12:48:01  czahl
 * - minor changes
 *
 * Revision 1.9  1995/04/20  17:43:45  czahl
 * - the logging facillity has been extracted to log.c. Currently this
 *   will be included but will be a stand-alone module in the near
 *   future
 * - A stripping mechanism has been installed for the case that the
 *   write buffer becommes to big which results in an increasing delay.
 *   This is done by a two stage mechanism. First every 16th sample
 *   will be discarded, which resulst in a faster pitch and decreased the
 *   buffer. Second the whole frame will be thrown away above a special
 *   threashold.
 *
 * Revision 1.8  1995/04/20  10:39:46  czahl
```

```
 * - all the UMS related stuff have been removed
 * - audio I/O will now be done with read/write/ioctl
 * - some kind of logging debugging informations has been installed.
 *   It will be used for tuning the audio part but can also be
 *   used for other purposes
 * - direct select support for the audio device has been used.
 *   Unfortunately this seems NOT to work with the old M-ACPA, just
 *   with the Ultimedia Audio Adapter (6302)
 *
 * Revision 1.7  1995/04/18  17:14:40  czahl
 * - this is the first working release by using the UMS audio
 *   stuff.
 *
 * Revision 1.6  1995/04/17  11:46:44  czahl
 * - first real tries with record and play
 *
 * Revision 1.5  1995/03/16  22:09:10  czahl
 * - minor changes
 *
 * Revision 1.4  1995/03/16  22:02:37  czahl
 * - bug in ahw_close; missing flag
 * - better ahw_read simulation for continous data stream
 *
 * Revision 1.3  1995/03/16  19:15:23  czahl
 * - using MU_LAW instead of PCM for testing
 *
 *update history: <obsolete>
 * date      who        why
 * 06-Mar-95 Chris      kommentieren der Funktionen, um UMS einbauen zu
 *                      können...
 * 10-Mar-95 Chris      weiter gehts...
 *                      nun ein Dummy zu debuggen
 * 16-Mar-95 Chris      problem mit "boolean", wird in UMS und include/types.h
 *                      definiert; zum Glueck gleich...
 *                      + RCS
 */
static  char rcsid[] = "$Id: ibm_audio.c,v 1.1 1995/08/17 13:58:26 hgs Exp $";
/*---------------------------------------------------------------------------*/

/*** standard includefiles ***/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/audio.h>          /* this is the new one from Paul!!! */
#include <sys/acpa.h>
#ifndef AUDIO_MODIFY_LIMITS
#  include "./ibm_audio.h"
#endif

/*** user includefiles ***/
#define NO_BOOLEAN
#include "audio_generic.h"
#include "ahw.h"
#include "util.h"
#include "config.h"
#ifdef CZAHL
#  define WITH_TRACE            0
#  include "/u/include/debug.h"
#else
#  define TRACE(xxx)
#  define FATAL_ERROR(xxx) {printf xxx; printf ("\n"); exit (1);}
#endif

/*** defines ***/

/*** type definitions ***/
typedef struct {
        int                     f;                              /* filedesc. */
        int                     Mode;                           /* O_RDONLY, O_WRONLY */
        char                    *Fn;                            /* filename for the device */
        int                     Operation;                      /* RECORD or PLAY */
        int                     Output;                         /* OUTPUT_1, ... */
        int                     Input;                          /* MIC, ... */
        int                     Rate;                           /* sample rate */
        int                     Fmt;                            /* audio format */
        int                     Channels;                       /* # of channels (1 / 2 */
        int                     BitsPerSample;                  /* 8 or 16 */
        unsigned int            Flags;                          /* for signed / unsigned */
        int                     BlockSize;                      /* for the DMA page */
        int                     Monitor;                        /* monitoring on / off */
        int                     MonitorVolume;                  /* monitoring Volume 0..255 */
        int                     Volume;                         /* playback volume 0..255 */
        int                     Balance;                        /* balance while playback -100..100 */
        int                     LeftGain;                       /* left gain level 0..255 */
        int                     RightGain;                      /* right gain level 0..255 */
        int                     IntSpeaker;                     /* on or off */
        int                     Port;                           /* currently selected ports */
        int                     HighWater;                      /* when we start to reduce the buffer */
} t_AudioDev;

/*** local functions ***/
static  int                     AhwSetRecord (audio_set_t What, int Val);
static  int                     AhwSetPlayback (audio_set_t What, int Val);
static  int                     AhwSetMonitor (audio_set_t What, int Val);
static  int                     AudioDevClose (t_AudioDev *Dev);
static  int                     AudioDevOpen (t_AudioDev *Dev);
static  void                    AudioInitDev ();
        /*** for IBM audio hardware ***/
```

```
static  int                     IbmAudioInitAudioChange (audio_change *AudioChange);
static  int                     IbmAudioInitAudioControl (audio_control *AudioControl);
static  int                     IbmAudioSelectInput (int f, int Input);
static  int                     IbmAudioSelectOutput (int f, int Output);
static  int                     IbmAudioSetBalance (int f, int Balance);
static  int                     IbmAudioSetGain (int f, int LeftGain, int RightGain);
static  int                     IbmAudioSetIntSpeaker (int f, int Enabled);
static  int                     IbmAudioSetMonitor (int f, int Monitor);
static  int                     IbmAudioSetVolume (int f, int Volume);
static  int                     IbmAudioStart (int f);
static  int                     IbmAudioStop (int f);

/*** global functions ***/

/*** local variables ***/
static  t_AudioDev              _PlayDev;                       /* the playback device */
static  t_AudioDev              _RecDev;                        /* the recording device */
static  t_AudioDev              *PlayDev = &_PlayDev;           /* for better handling */
static  t_AudioDev              *RecDev = &_RecDev;             /* for better handling */

static  char                    *RecDevFn[] = {
                                        "/dev/baud0/1",
                                        "/dev/baud1/1",
                                        "/dev/paud0/1",
                                        "/dev/paud1/1",
                                        NULL};
static  char                    *PlayDevFn[] = {
                                        "/dev/baud0/2",
                                        "/dev/baud1/2",
                                        "/dev/paud0/2",
                                        "/dev/paud1/2",
                                        NULL};

/*** global variables ***/
        int                     ahw_changed;                    /* flag: audio device has changed */
        int                     ahw_errno;                      /* error indication */


/*-----------------------------------------------------------------------*/
int ahw_open (int ndelay)
{
        TRACE (("ahw_open (ndelay = %d)\n", ndelay));
        AudioInitDev ();
/*** open and start the record part ***/
        AudioDevOpen (RecDev);
        AudioDevInit (RecDev);
        IbmAudioSelectInput (RecDev->f, RecDev->Input);
        IbmAudioSetGain (RecDev->f, RecDev->LeftGain, RecDev->RightGain);
        IbmAudioSetMonitor (RecDev->f, RecDev->Monitor);
/*      IbmAudioSetMonitorVolume (RecDev->f, RecDev->MonitorVol); */
        IbmAudioSetCaptureBuffer (RecDev->f, 2);
        IbmAudioStart (RecDev->f);
/*** open and start the playback part ***/
        AudioDevOpen (PlayDev);
        AudioDevInit (PlayDev);
        IbmAudioSelectOutput (PlayDev->f, PlayDev->Output);
        IbmAudioSetVolume (PlayDev->f, PlayDev->Volume);
        IbmAudioSetBalance (PlayDev->f, PlayDev->Balance);
        IbmAudioSetIntSpeaker (PlayDev->f, PlayDev->IntSpeaker);
        IbmAudioStart (PlayDev->f);
        return RecDev->f;                                       /* return fd for select call */
} /* ahw_open */
/*-----------------------------------------------------------------------*/
int ahw_close (void)
{
        int                     OldFd;

        TRACE (("ahw_close ()"));
/*** close the audio device ***/
        OldFd = RecDev->f;                                      /* we need it (see below) */
        IbmAudioStop (RecDev->f);                               /* stop recording */
        IbmAudioStop (PlayDev->f);                              /* and playback now */
        AudioDevClose (RecDev);                                 /* close the recording */
        AudioDevClose (PlayDev);                                /* and playback track */
/*** ***/
        return OldFd;                                           /* return the old input fd */
} /* ahw_close */
/*-----------------------------------------------------------------------*/
static int AhwSetRecord (audio_set_t What, int Val)
/*
 * Does all the settings for the recording part.
 */
{
TRACE (("AhwSetRecord (%d, %d)", What, Val));
        switch (What) {
/*** select input source ? ***/
          case AS_port:
            if (Val == AP_in_line_1)                            /* LINE-IN (external) */
              RecDev->Input = LINE_1;
            else if (Val == AP_in_line_2)                       /* AUX-1 (internal CD-ROM) */
              RecDev->Input = AUX1;
            else if (Val == AP_in_microphone_high)              /* high gain mike */
              RecDev->Input = HIGH_GAIN_MIKE;
            else if (Val == AP_in_microphone_low)               /* low gain mike */
              RecDev->Input = LOW_GAIN_MIKE;
            else                                                /* anything else? */
              return -1;                                        /* n/a */
            IbmAudioSelectInput (RecDev->f, RecDev->Input);     /* select the port */
            return 0;
```

```
/*** set the gain for left and right ***/
          case AS_gain:                                         /* input gain for left=right */
          case AS_gain_right:                                   /* input gain for right */
          case AS_gain_left:                                    /* input gain for left */
            if (Val < 0)
              Val = 0;
            else if (Val > 255)
              Val = 255;
            if ((What == AS_gain) || (What == AS_gain_left))
              RecDev->LeftGain = Val;
            if ((What == AS_gain) || (What == AS_gain_right))
              RecDev->RightGain = Val;
            IbmAudioSetGain (RecDev->f, RecDev->LeftGain, RecDev->RightGain);
            return 0;
/*** set gain step ***/
          case AS_gain_step:                                    /* ??? */
            TRACE (("AS_gain_step is not supported"));
            return 0;                                           /* n/a, ignore */
/*** set fillpoint ***/
          case AS_fillpoint:                                    /* ??? */
            TRACE (("AS_fillpoint is not supported"));
            return 0;                                           /* currently not supported */
/*** bullshit? ***/
          default:
            TRACE (("\007unknown action: ahw_set (record, %d, %d)", What, Val));
            return -1;
          } /* switch */
} /* AhwSetRecord */
/*----------------------------------------------------------------------*/
static int AhwSetPlayback (audio_set_t What, int Val)
/*
 * Does all the settings for the playback part.
 */
{
TRACE (("AhwSetPlayback (%d, %d)", What, Val));
        switch (What) {
/*** select output ? ***/
          case AS_port:
            if (Val == AP_out_line_1) {                         /* external output */
              PlayDev->Port = AP_out_line_1;
              PlayDev->Output = OUTPUT_1;
/*
 *!!! we don't HAVE TO select the output once again, because the internal speaker
 *!!! will be reset to the default (on) value
              IbmAudioSelectOutput (PlayDev->f, PlayDev->Output);
 */
              }
            else if (Val == AP_out_speaker) {                   /* internal speaker */
              PlayDev->Port = AP_out_speaker;
              }
            else                                                /* anything else ? */
              return -1;                                        /* n/a */
            return 0;
/*** set the gain for left and right ***/
          case AS_gain:                                         /* output volume for left=right */
          case AS_gain_right:                                   /* output volume for right */
          case AS_gain_left:                                    /* output volume for left */
TRACE (("AhwSetPlayback.gain (%d, %d)", What, Val));
            if (Val < 0)
              Val = 0;
            else if (Val > 255)
              Val = 255;
            if (PlayDev->Port == AP_out_speaker) {
              if (Val >= 128)
                PlayDev->IntSpeaker = 1;
              else
                PlayDev->IntSpeaker = 0;
              IbmAudioSetIntSpeaker (PlayDev->f, PlayDev->IntSpeaker);
              }
            else {
              PlayDev->Volume = Val;                            /* we don't distinguish between left and right */
              IbmAudioSetVolume (PlayDev->f, PlayDev->Volume);
              }
            return 0;
/*** set balance ***/
          case AS_balance:                                      /* output balance */
            if (Val < -255)
              Val = -255;
            else if (Val > 255)
              Val = 255;
            PlayDev->Balance = Val;
            IbmAudioSetBalance (PlayDev->f, PlayDev->Balance);
            return 0;
/*** set gain step ***/
          case AS_gain_step:                                    /* ??? */
            TRACE (("AS_gain_step is not supported"));
            return 0;                                           /* n/a, ignore */
/*** set fillpoint ***/
          case AS_fillpoint:                                    /* ??? */
            TRACE (("AS_fillpoint is not supported"));
            return 0;                                           /* currently not supported */
/*** bullshit? ***/
          default:
            TRACE (("\007unknown action: ahw_set (playback, %d, %d)", What, Val));
            return -1;
          } /* switch */
} /* AhwSetPlayback */
/*----------------------------------------------------------------------*/
static int AhwSetMonitor (audio_set_t What, int Val)
```

```
/*
 * Does all the settings for the monitoring part.
 */
{
TRACE (("AhwSetMonitor (%d, %d)", What, Val));
        switch (What) {
/*** select output ? ***/
          case AS_port:
            return -1;                                          /* n/a */
            return 0;
/*** set the gain for left and right ***/
          case AS_gain:                                         /* output volume for left=right */
          case AS_gain_right:                                   /* output volume for right */
          case AS_gain_left:                                    /* output volume for left */
            if (Val < 0)
              Val = 0;
            else if (Val > 255)
              Val = 255;
            return 0;
/*** set gain step ***/
          case AS_gain_step:                                    /* ??? */
            TRACE (("AS_gain_step is not supported"));
            return 0;                                           /* n/a, ignore */
/*** set fillpoint ***/
          case AS_fillpoint:                                    /* ??? */
            TRACE (("AS_fillpoint is not supported"));
            return 0;                                           /* currently not supported */
/*** bullshit? ***/
          default:
            TRACE (("\007unknown action: ahw_set (monitor, %d, %d)", What, Val));
            return -1;
          } /* switch */
} /* AhwSetMonitor */
/*-----------------------------------------------------------------------*/
int ahw_set (int Device, audio_set_t What, int Val)
{

        AudioInitDev ();
        switch (Device) {
          case AD_RECORD:
            return AhwSetRecord (What, Val);
          case AD_PLAYBACK:
            return AhwSetPlayback (What, Val);
          case AD_MONITOR:
            return AhwSetMonitor (What, Val);
          default:                                              /* oops! */
            TRACE (("\007unknown action: ahw_set (%d, %d, %d)", Device, What, Val));
            return -1;
          } /* switch */
} /* ahw_set */
/*-----------------------------------------------------------------------*/
int ahw_fmt(audio_descr_t *Rec, audio_descr_t *Play)
{
fprintf (stderr, "ahw_fmt (Rec=..., Play=...)\n");
fprintf (stderr, "\tRec->\n");
fprintf (stderr, "\t\tenc\t%d\n", Rec->encoding);
fprintf (stderr, "\t\trate\t%d\n", Rec->sample_rate);
fprintf (stderr, "\t\tchan\t%d\n", Rec->channels);
fprintf (stderr, "\tPlay->\n");
fprintf (stderr, "\t\tenc\t%d\n", Play->encoding);
fprintf (stderr, "\t\trate\t%d\n", Play->sample_rate);
fprintf (stderr, "\t\tchan\t%d\n", Play->channels);

        AudioInitDev ();
/*** check Rec format ***/
        RecDev->Flags = FIXED;
        switch (Rec->encoding) {
          case AE_PCMU:
TRACE (("mu-law"));
            RecDev->Fmt = MU_LAW;
            RecDev->BitsPerSample = 8;
            break;
          case AE_PCMA:
TRACE (("a-law"));
            RecDev->Fmt = A_LAW;
            RecDev->BitsPerSample = 8;
            break;
          case AE_L8:
TRACE (("pcm"));
            RecDev->Fmt = PCM;
            RecDev->BitsPerSample = 8;
            break;
          case AE_L16:
            RecDev->Fmt = PCM;
            RecDev->BitsPerSample = 16;
            RecDev->Flags = FIXED | SIGNED;
            break;
          default:
            Rec->encoding = AE_PCMU;
            RecDev->Fmt = MU_LAW;
            RecDev->BitsPerSample = 8;
            break;
          } /* switch */
/*      audio_descr (Rec);                                      /* update some paramters */
        RecDev->Rate = Rec->sample_rate;
        RecDev->Channels = Rec->channels;
        RecDev->BlockSize = 256;                        /* !!! depends on rate */
/*** check play format ***/
        PlayDev->Flags = FIXED;
```

```
        switch (Play->encoding) {
          case AE_PCMU:
            PlayDev->Fmt = MU_LAW;
            PlayDev->BitsPerSample = 8;
            break;
          case AE_PCMA:
            PlayDev->Fmt = A_LAW;
            PlayDev->BitsPerSample = 8;
            break;
          case AE_L8:
            PlayDev->Fmt = PCM;
            PlayDev->BitsPerSample = 8;
            break;
          case AE_L16:
            PlayDev->Fmt = PCM;
            PlayDev->BitsPerSample = 16;
            PlayDev->Flags = FIXED | SIGNED;
            break;
          default:
            Play->encoding = AE_PCMU;
            PlayDev->Fmt = MU_LAW;
            PlayDev->BitsPerSample = 8;
            break;
          } /* switch */
/*      audio_descr (Play);                                     /* update some parameters */
        PlayDev->Rate = Play->sample_rate;
        PlayDev->Channels = Play->channels;
        PlayDev->BlockSize = 128*3;
#define ms *8
        PlayDev->HighWater = 130 ms;
        return 0;                                               /* ok */
} /* ahw_fmt */
/*-------------------------------------------------------------------------*/
int ahw_get (audio_state_t *AudioState)
{
  TRACE (("ahw_get"));
  AudioInitDev ();
  return 0;
} /* ahw_get */
/*-------------------------------------------------------------------------*/
int ahw_waiting (void)
{
        TRACE (("ahw_waiting ()"));
        return 0;
} /* ahw_waiting */
/*-------------------------------------------------------------------------*/
void ahw_track (signal_func_t Handler)
/* Not supported or undocumented feature for the IBM RS/6000 family. */
{
        TRACE (("ahw_track: IS NOT SUPPORTED"));
} /* ahw_track */
/*-------------------------------------------------------------------------*/
int ahw_write_queue (int *Underrun)
/*
 * Returns the number of bytes in the write queue. If an underrun
 * condition has been occured, we return a negative value and set
 * Underrun to TRUE, otherwise to FLASE.
 * NOTE: Because we handle audio underruns in a special manner, we
 * will allways lie and no not indicate any underun!
 */
{
        audio_buffer            AudioBuffer;

        if (ioctl (PlayDev->f, AUDIO_BUFFER, &AudioBuffer) == -1) {
          TRACE (("ioctl"));
          return -1;
          }
        *Underrun = 0;
        return AudioBuffer.write_buf_size;
} /* ahw_write_queue */
/*-------------------------------------------------------------------------*/
int ahw_read_queue(int *errors)
{
        TRACE (("ahw_read_queue (errors=%d)", *errors));
        return 0;
} /* ahw_read_queue */
/*-------------------------------------------------------------------------*/
int ahw_write (char *Buf, int Bytes)
{
#define SKIP_INTERVAL           (32)
static  int                     SkipFlag = FALSE;
static  int                     HighWaterReached = 0;
        audio_buffer            BufStat;
        int                     n;
        int                     x;
        int                     i;

        if (PlayDev->f == -1) {                                 /* device not open? */
          TRACE (("ahw_write: dev is not open"));
          return Bytes;                                         /* we need to lie */
          } /* if */
/*** get the buffer status ***/
        if (ioctl (PlayDev->f, AUDIO_BUFFER, &BufStat) == -1) { /* get buffer state */
          TRACE (("ahw_write: unable to get buffer status"));
          return 0;
          }
/* printf ("%d ", BufStat.write_buf_size); fflush (stdout); */
/*** check if we need buffer adjustment ***/
        if (BufStat.write_buf_size >= PlayDev->HighWater)       /* buffer too full, should shrink? */
```

```
          SkipFlag = TRUE;                                      /* set flag */
        if (SkipFlag) {
          HighWaterReached++;
          if (HighWaterReached < 10)
            SkipFlag = 0;
          else
            HighWaterReached = 0;
          }
        else
          HighWaterReached = 0;
/*** eventualy reduce the buffer by omitting some samples ***/
        if (SkipFlag) {                                         /* reduce the buffer? */
printf ("[%3d ms]", BufStat.write_buf_size /8);
fflush (stdout);
          for (i=0, x=0; i<Bytes; i++)
            if (! (i % SKIP_INTERVAL))
              continue;
            Buf[x++] = Buf[i];
          }
        else                                                    /* no reducing? */
          x = Bytes;
/*** write the data to the device ***/
        n = write (PlayDev->f, Buf, x);                         /* write to playback track */
        if (n == -1) {                                          /* error ? */
printf ("[W error %d]", errno); fflush (stdout);
          return -1;
          }
        return Bytes;                                           /* always lie; all has been written */
} /* ahw_write */
/*-----------------------------------------------------------------------*/
int ahw_read (char *Buf, int Bytes)
{
        int                     n;
        audio_buffer            AudioBuffer;

        if (RecDev->f == -1) {                                  /* track is not open? */
          TRACE (("ahw_read: dev not open"));
          return Bytes;                                         /* lie anyway */
          } /* if */
ioctl (RecDev->f, AUDIO_BUFFER, &AudioBuffer);
if (AudioBuffer.flags != 0) {
  printf ("[R-%d]", AudioBuffer.flags);
  fflush (stdout);
}
        n = read (RecDev->f, Buf, Bytes);                       /* read the Data */
        if (n == -1)                                            /* error occured? */
          if (errno == EINTR)                                   /* interrupted syscall? */
            return 0;                                           /* ok, try later again */
          else {                                                /* real error ? */
            TRACE (("read"));
            return -1;
            } /* else */
        return n;                                               /* return # bytes captured */
} /* ahw_read */
/*-----------------------------------------------------------------------*/
static AudioDevOpen (t_AudioDev *Dev)
{
        TRACE (("AudioDevOpen (%p)", Dev));
        AudioInitDev ();                                                /* initialize internal control structures */
/*** check if we allready have it done ***/
        if (Dev->f != -1)                                       /* device is open ? */
          return 0;                                             /* that's all folks */
/*** open the device ***/
        Dev->f = open (Dev->Fn, Dev->Mode);                     /* open the right device part */
        if (Dev->f == -1) {                                     /* error ? */
          TRACE (("unable to open"));
          FATAL_ERROR (("open"));
          return -1;                                            /* report error */
          }
        return 0;
} /* AudioDevOpen */
/*-----------------------------------------------------------------------*/
static AudioDevInit (t_AudioDev *Dev)
/*
 * Initializes on of the audio tracks. The track has to be opened and
 * it has to stoped. Otherwise onpredicted results occurs.
 */
{
        audio_init              AudioInit;

        TRACE (("AudioDevInit (%p)", Dev));
        AudioInitDev ();                                                /* initialize internal control structures */
        AudioInit.srate =               Dev->Rate;              /* sample rate */
        AudioInit.bits_per_sample =     Dev->BitsPerSample;     /* bits / sample */
        AudioInit.bsize =               Dev->BlockSize;         /* DMA block size */
        AudioInit.mode =                Dev->Fmt;               /* encoding format */
        AudioInit.channels =            Dev->Channels;          /* 1 or 2 */
        AudioInit.position_resolution = AUDIO_IGNORE;           /* no interest now */
        AudioInit.flags =               Dev->Flags;             /* needed Flags */
        AudioInit.operation =           Dev->Operation;         /* record or playback */
        AudioInit.reserved =            NULL;                   /* we are nice today... */
        if (ioctl (Dev->f, AUDIO_INIT, &AudioInit) == -1) {
          TRACE (("AudioDevInit: ioctl"));
          FATAL_ERROR (("AudioDevInit: ioctl"));
          return -1;
          }
printf ("AudioInit.bsize = %d\n", Dev->BlockSize);
} /* AudioDevInit */
/*-----------------------------------------------------------------------*/
```

```
static AudioDevClose (t_AudioDev *Dev)
{
        TRACE (("AudioDevClose (%p)", Dev));
        if (Dev->f == -1)                                       /* already closed? */
          return 0;                                             /* ok, job done */
        close (Dev->f);                                         /* close it */
        Dev->f = -1;                                            /* and mark it as closed */
        return 0;                                               /* return success */
} /* AudioDevClose */
/*---------------------------------------------------------------------*/
static void AudioInitDev ()
/*
 * Initializes the characteristics of the audio device, for the recording
 * and playback part.
 */
{
static  int                     Init = 0;
        int                     i;
        int                     f;

        if (Init)
          return;
        Init = 1;
/*** search audio device ***/
        for (i=0; RecDevFn[i] != NULL; i++)
          if ((f = open (RecDevFn[i], O_RDONLY)) != -1)
            break;
          else
            if (errno != ENOENT)
              break;
        if (f != -1)
          close (f);
/*** record ***/
        RecDev->f =             -1;
        RecDev->Fn =            RecDevFn[i];
        RecDev->Mode =          O_RDONLY | O_NDELAY;
        RecDev->Operation =     RECORD;
        RecDev->Input =         AUDIO_IGNORE;
        RecDev->Rate =          8000;
        RecDev->Fmt =           PCM;
        RecDev->Channels =      1;
        RecDev->BitsPerSample = 8;
        RecDev->Flags =         FIXED;
        RecDev->BlockSize =     AUDIO_IGNORE;
        RecDev->Monitor =       MONITOR_OFF;
        RecDev->LeftGain =      0;
        RecDev->RightGain =     0;
        RecDev->Volume =        AUDIO_IGNORE;
        RecDev->Balance =       AUDIO_IGNORE;
        RecDev->Output =        AUDIO_IGNORE;
        RecDev->IntSpeaker =    AUDIO_IGNORE;
/*** playback ***/
        PlayDev->f =            -1;
        PlayDev->Fn =           PlayDevFn[i];
        PlayDev->Mode =         O_WRONLY | O_NDELAY;
        PlayDev->Operation =    PLAY;
        PlayDev->Output =       OUTPUT_1;
        PlayDev->Rate =         8000;
        PlayDev->Fmt =          PCM;
        PlayDev->Channels =     1;
        PlayDev->BitsPerSample = 8;
        PlayDev->Flags =        FIXED;
        PlayDev->BlockSize =    AUDIO_IGNORE;
        PlayDev->Volume =       0;
        PlayDev->Balance =      0;
        PlayDev->IntSpeaker =   0;
        PlayDev->Input =        AUDIO_IGNORE;
        PlayDev->Monitor =      AUDIO_IGNORE;
        PlayDev->LeftGain =     AUDIO_IGNORE;
        PlayDev->RightGain =    AUDIO_IGNORE;
} /* AudioInitDev */
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
/*
 * PRIVAT ROUTINES FOR SETTING THE AUDIO PARAMTERS
 * THESE ROUTINES CAN BE USED IN GENERAL BECAUSE THEY DON'T DEPEND
 * ON NEVOT.
 */

/*---------------------------------------------------------------------*/
static IbmAudioInitAudioChange (audio_change *AudioChange)
{
        AudioChange->dev_info =         NULL;
        AudioChange->input =            AUDIO_IGNORE;
        AudioChange->output =           AUDIO_IGNORE;
        AudioChange->monitor =          AUDIO_IGNORE;
        AudioChange->volume =           AUDIO_IGNORE;
        AudioChange->volume_delay =     AUDIO_IGNORE;
        AudioChange->balance =          AUDIO_IGNORE;
        AudioChange->balance_delay =    AUDIO_IGNORE;
        AudioChange->treble =           AUDIO_IGNORE;           /* not supported */
        AudioChange->bass =             AUDIO_IGNORE;           /* not supported */
        AudioChange->pitch =            AUDIO_IGNORE;           /* not supported */
} /* IbmAudioInitAudioChange */
/*---------------------------------------------------------------------*/
static IbmAudioInitAudioControl (audio_control *AudioControl)
{
```

```
        AudioControl->ioctl_request =   AUDIO_IGNORE;
        AudioControl->request_info =    NULL;
        AudioControl->position =        0;
        AudioControl->return_code =     0xDeadBeaf;
} /* IbmAudioInitAudioControl */
/*--------------------------------------------------------------------*/
static IbmAudioStart (int f)
/*
 * Starts the specified device.
 */
{
        audio_control           AudioControl;

TRACE (("IbmAudioStart (%d)", f));
        if (f == -1)
          return 0;
        IbmAudioInitAudioControl (&AudioControl);
        AudioControl.ioctl_request = AUDIO_START;               /* start cmd */
        if (ioctl (f, AUDIO_CONTROL, &AudioControl) != 0) {
          TRACE (("unable to issue AUDIO_START"));
          perror ("AudioDevStart:ioctl");
          return -1;                                            /* return failiure */
          }
        return 0;                                               /* return success */
} /* IbmAudioStart */
/*--------------------------------------------------------------------*/
static IbmAudioStop (int f)
{
        audio_control           AudioControl;

TRACE (("IbmAudioStop (%d)", f));
        if (f == -1)
          return 0;
        IbmAudioInitAudioControl (&AudioControl);
        AudioControl.ioctl_request = AUDIO_STOP;                /* stop cmd */
        if (ioctl (f, AUDIO_CONTROL, &AudioControl) != 0) {
          TRACE (("unable to issue AUDIO_STOP"));
          perror ("AudioDevStop:ioctl");
          return -1;                                            /* return failiure */
          } /* if */
        return 0;                                               /* return success */
} /* IbmAudioStop */
/*--------------------------------------------------------------------*/
static IbmAudioSetVolume (int f, int Volume)
/*
 * Volume:      0..255
 */
{
        audio_change            AudioChange;
        audio_control           AudioControl;

TRACE (("IbmAudioSet VOlume (%d, %d)", f, Volume));
        if (f == -1)
          return 0;
/*** range check ***/
        if (Volume < 0)
          Volume = 0;
        else if (Volume > 255)
          Volume = 255;
/*** set it ***/
        IbmAudioInitAudioChange (&AudioChange);
        IbmAudioInitAudioControl (&AudioControl);
        AudioChange.volume = 0x7fff0000 / 255 * Volume;
        AudioChange.volume_delay = 0;
        AudioControl.request_info = &AudioChange;
        AudioControl.ioctl_request = AUDIO_CHANGE;
        if (ioctl (f, AUDIO_CONTROL, &AudioControl) == -1) {
          TRACE (("AudioDevSetVolume"));
          }
        return 0;
} /* IbmAudioSetVolume */
/*--------------------------------------------------------------------*/
static int IbmAudioSetBalance (int f, int Balance)
/*
 * Blance:      -255..0..255
 */
{
        audio_change            AudioChange;
        audio_control           AudioControl;

TRACE (("IbmAudioSetBalance (%d, %d)", f, Balance));
        if (f == -1)
          return 0;
/*** range check ***/
        if (Balance < -255)
          Balance = -255;
        else if (Balance > 255)
          Balance = 255;
        PlayDev->Balance = Balance;
/*** scaling ***/
        Balance += 255;
        Balance = 0x7fff0000 / (511) * Balance;
/*** setting ***/
        IbmAudioInitAudioChange (&AudioChange);
        IbmAudioInitAudioControl (&AudioControl);
        AudioChange.balance = Balance;
        AudioChange.balance_delay = 0;
        AudioControl.request_info = &AudioChange;
        AudioControl.ioctl_request = AUDIO_CHANGE;
```

```
        if (ioctl (f, AUDIO_CONTROL, &AudioControl) == -1) {
          TRACE (("AudioDevSetBalance"));
          }
        return 0;
} /* IbmAudioSetBalance */
/*---------------------------------------------------------------------*/
static int IbmAudioSetGain (int f, int LeftGain, int RightGain)
/*
 * LeftGain:    0..255
 * RightGain:   0..255
 */
{
        audio_set_gain          AudioSetGain;

TRACE (("IbmAudioSetGain (%d, %d, %d)", f, LeftGain, RightGain));
        if (f == -1)
          return 0;
/*** range checks ***/
        if (LeftGain < 0)
          LeftGain = 0;
        else if (LeftGain > 255)
          LeftGain = 255;
        if (RightGain < 0)
          RightGain = 0;
        else if (RightGain > 255)
          RightGain = 255;
/*** set it ***/
        AudioSetGain.left_gain = LeftGain * 100 / 255;
        AudioSetGain.right_gain = RightGain * 100 / 255;
        if (ioctl (f, AUDIO_SET_GAIN, &AudioSetGain) == -1) {
          TRACE (("AudioDevSetGain"));
          }
        return 0;
} /* IbmAudioSetGain */
/*---------------------------------------------------------------------*/
static int IbmAudioSelectInput (int f, int Input)
{
        audio_change            AudioChange;
        audio_control           AudioControl;

TRACE (("IbmAudioSelectInput (%d, %d)", f, Input));
        if (f == -1)
          return 0;
        IbmAudioInitAudioChange (&AudioChange);
        IbmAudioInitAudioControl (&AudioControl);
        AudioChange.input = Input;
        AudioControl.request_info = &AudioChange;
        AudioControl.ioctl_request = AUDIO_CHANGE;
        if (ioctl (f, AUDIO_CONTROL, &AudioControl) == -1) {
          TRACE (("ioctl"));
          }
        return 0;
} /* IbmAudioSelectInput */
/*---------------------------------------------------------------------*/
static int IbmAudioSelectOutput (int f, int Output)
{
        audio_change            AudioChange;
        audio_control           AudioControl;

TRACE (("IbmAudioSelectOutput (%d, %d)", f, Output));
        if (f == -1)
          return 0;
        IbmAudioInitAudioChange (&AudioChange);
        IbmAudioInitAudioControl (&AudioControl);
        AudioChange.output = Output;
        AudioControl.request_info = &AudioChange;
        AudioControl.ioctl_request = AUDIO_CHANGE;
        if (ioctl (f, AUDIO_CONTROL, &AudioControl) == -1) {
          TRACE (("ioctl"));
          }
        return 0;
} /* IbmAudioSelectOutput */
/*---------------------------------------------------------------------*/
IbmAudioSetIntSpeaker (int f, int Enabled)
{
        audio_change            AudioChange;
        audio_control           AudioControl;

TRACE (("IbmAudioSetIntSpeaker (%d, %d)", f, Enabled));
        if (f == -1)
          return 0;
        IbmAudioInitAudioChange (&AudioChange);
        IbmAudioInitAudioControl (&AudioControl);
        if (Enabled)
          AudioChange.output = INTERNAL_SPEAKER_ON;
        else
          AudioChange.output = INTERNAL_SPEAKER_OFF;
        AudioControl.request_info = &AudioChange;
        AudioControl.ioctl_request = AUDIO_CHANGE;
        if (ioctl (f, AUDIO_CONTROL, &AudioControl) == -1) {
          TRACE (("ioctl"));
          }
        return 0;
} /* IbmAudioSetIntSpeaker */
/*---------------------------------------------------------------------*/
static int IbmAudioSetMonitor (int f, int Monitor)
{
        audio_change            AudioChange;
        audio_control           AudioControl;
```

```
TRACE (("IbmAudioSetMonitor (%d, %d)", f, Monitor));
        if (f == -1)
          return 0;
        IbmAudioInitAudioChange (&AudioChange);
        IbmAudioInitAudioControl (&AudioControl);
        if (Monitor)
          AudioChange.monitor = MONITOR_UNCOMPRESSED;
        else
          AudioChange.monitor = MONITOR_OFF;
        AudioControl.request_info = &AudioChange;
        AudioControl.ioctl_request = AUDIO_CHANGE;
        if (ioctl (f, AUDIO_CONTROL, &AudioControl) == -1) {
          TRACE (("ioctl"));
          }
        return 0;
} /* IbmAudioSetMonitor */
/*--------------------------------------------------------------------*/
static int IbmAudioSetCaptureBuffer (int f, int Time)
{
        init_buf_req            BufReq;

        BufReq.buffer_size = 8000* Time;        /* has to be multiple of #bytes/s */
        BufReq.play_block_count = AUDIO_IGNORE;
        BufReq.play_lower_limit = AUDIO_IGNORE;
        BufReq.cap_upper_limit = AUDIO_IGNORE;
        BufReq.byte_count = AUDIO_IGNORE;
        BufReq.return_code = 0xDeadBeaf;
        if (ioctl (f, AUDIO_MODIFY_LIMITS, &BufReq) == -1) {
          TRACE (("ioctl"));
          }
        return 0;
} /* IbmAudioSetCaptureBuffer */
```

D.2 tk/ibm_audioctl.c

```
# Control panel for IBM AIX workstations using the Ultimedia Audio Adapter
#
# $Id: ibm_audioctl.tcl,v 1.1 1995/08/17 14:00:37 hgs Exp $
#
# $Log: ibm_audioctl.tcl,v $
# Revision 1.1  1995/08/17  14:00:37  hgs
# Initial revision
#
# Revision 1.4  1995/06/27  15:24:25  czahl
# - changes were needed because we also want to switch the internal
#   speaker on and off.
#
# Revision 1.3  1995/06/27  13:11:07  czahl
# - bug: aux1 radiobutton set the wrong value
#
# Revision 1.2  1995/06/27  10:38:05  czahl
# - First "official" version for setting the AIX audio parameters.
#   The style of this panel is very different from to the others,
#   keep this in mind...
#
#

proc MakeScale {w From To o Label Side} {
        frame $w
          frame $w.s
            frame $w.s.l
              label $w.s.l.t -text $To
              label $w.s.l.f -text $From
              pack $w.s.l.t -side top
              pack $w.s.l.f -side bottom
              set S [scale $w.s.s -from $From -to $To -orient $o -showvalue 0 \
                -to $From -from $To ]
            if {$Side == "left"} {
              pack $w.s.l -side left -expand true -fill both
              pack $w.s.s -side right
            } else {
              pack $w.s.l -side right -expand true -fill both
              pack $w.s.s -side left
            }
          label $w.l -text $Label
          pack $w.s -side top
          if {$Side == "left"} {
            pack $w.l -side top -anchor e
          } else {
            pack $w.l -side top -anchor w
          }
        return $S
}

proc audioctl_panel {{w .audioctl} {title "Audio Control"}} {
        if {[winfo exists $w]} {destroy $w}

        toplevel $w
        set i [frame $w.i -bd 1 -relief raised]
          frame $i.f1 -bd 1 -relief raised
            label $i.f1.l -text "Input Settings"
            pack $i.f1.l -fill x -expand true
          frame $i.f
```

```
            frame $i.f.c -bd 1 -relief raised
              radiobutton $i.f.c.hgm -text "high gain mic" -relief flat \
                -variable AudioCtl(Input) -value highgainmike \
                -command {audioctl -input $AudioCtl(Input)}
              radiobutton $i.f.c.lgm -text "low gain mic" -relief flat \
                -variable AudioCtl(Input) -value lowgainmike \
                -command {audioctl -input $AudioCtl(Input)}
              radiobutton $i.f.c.lin -text "line 1" -relief flat \
                -variable AudioCtl(Input) -value line1 \
                -command {audioctl -input $AudioCtl(Input)}
              radiobutton $i.f.c.aux -text "aux 1" -relief flat \
                -variable AudioCtl(Input) -value line2 \
                -command {audioctl -input $AudioCtl(Input)}
              pack $i.f.c.hgm $i.f.c.lgm $i.f.c.lin $i.f.c.aux \
                -side top -anchor w -expand true -padx 2m
            frame $i.f.g -bd 1 -relief raised
              label $i.f.g.l -text "Gain"
              frame $i.f.g.s
                set s [MakeScale $i.f.g.s.l 0 100 vertical "L" left]
                $s config -command "audioctl -leftgain "
                set s [MakeScale $i.f.g.s.r 0 100 vertical "R" right]
                $s config -command "audioctl -rightgain "
                pack $i.f.g.s.l $i.f.g.s.r -side left
              pack $i.f.g.l $i.f.g.s -side top -padx 2m
            pack $i.f.c $i.f.g -side left -expand true -fill y
          pack $i.f1 $i.f -side top -expand true -fill x
        set o [frame $w.o -bd 1 -relief raised]
          frame $o.f1 -bd 1 -relief raised
            label $o.f1.l -text "Output Settings"
            pack $o.f1.l -fill x -expand true
          frame $o.f2
            frame $o.f2.c -bd 0 -relief raised
              checkbutton $o.f2.c.isp -text "internal speaker" -relief raised \
                -bd 1 -variable AudioCtl(IntSpeaker) \
                -command {
                        audioctl -intspeaker $AudioCtl(IntSpeaker)
                        audioctl -output line1
                }
              frame $o.f2.c.f -bd 1 -relief raised
                label $o.f2.c.f.l -text Balance
                scale $o.f2.c.f.s -from -100 -to 100 -showvalue false \
                        -orient horizontal \
                        -command "audioctl -balance "
              frame $o.f2.c.f.f
                label $o.f2.c.f.f.l -text L -anchor w
                label $o.f2.c.f.f.m -text M
                label $o.f2.c.f.f.r -text R -anchor e
                pack $o.f2.c.f.f.l $o.f2.c.f.f.m $o.f2.c.f.f.r \
                        -side left -expand true -fill x
              pack $o.f2.c.f.l $o.f2.c.f.s $o.f2.c.f.f -side top -expand true -fill x -padx 2m
            pack $o.f2.c.isp $o.f2.c.f \
              -anchor w -expand true -fill both -padx 0m -side top
            frame $o.f2.v -bd 1 -relief raised
              label $o.f2.v.l -text "Volume"
              scale $o.f2.v.s -from 100 -to 0 -orient vertical \
                -command "audioctl -volume "
              pack $o.f2.v.l $o.f2.v.s -side top -fill both -pady 2m
            pack $o.f2.c $o.f2.v -side left -expand true -fill both
          pack $o.f1 $o.f2 -side top -fill x -expand true
        set b [frame $w.b -bd 1 -relief raised]
          frame $b.f -bd 1 -relief raised
            button $b.f.b -text "Dismiss" -command "destroy $w"
            pack $b.f.b -fill x -padx 3m -pady 3m
          pack $b.f -expand true -fill both
        pack $i $o $b -side top -expand true -fill x
}

#audioctl_panel
```

D.3 nevot/cmd_audioctl.c

```
/*
 * COPYRIGHT BY OERTEL & ZAHL SOFTWAREENTWICKLUNG GbR
 *      file:           audioctl.c
 *      written by:     Christian Zahl
 *      description:    Tcl/Tk extensions for audio control
 *
 * $Id: cmd_audioctl.c,v 1.2 1995/08/30 11:11:25 hgs Exp $
 *
 * $Log: cmd_audioctl.c,v $
 * Revision 1.2  1995/08/30  11:11:25  hgs
 * 3.32
 *
 * Revision 1.1  1995/08/17  13:54:58  hgs
 * Initial revision
 *
 * Revision 1.2  1995/06/27  13:17:07  czahl
 * - Privat routines for IBM substituted by ahw_set
 * - Added headspeaker support
 *
 * Revision 1.1  1995/06/27  12:47:44  czahl
 * Initial revision
 *
```

```
 */
static char rcsid[] = "$Id: cmd_audioctl.c,v 1.2 1995/08/30 11:11:25 hgs Exp $\n";

/*** standard includefiles ***/
#include <stdlib.h>   /* atoi() */
#include <string.h>   /* strcmp() */
#include <tcl.h>

/*** user includefiles ***/
#include "ahw.h"

/*** local functions ***/
static  int             ScanBalance (Tcl_Interp *Interp, int argc, char *argv[]);
static  int             ScanGain (Tcl_Interp *Interp, int argc, char *argv[]);
static  int             ScanInput (Tcl_Interp *Interp, int argc, char *argv[]);
static  int             ScanIntSpeaker (Tcl_Interp *Interp, int argc, char *argv[]);
static  int             ScanLeftGain (Tcl_Interp *Interp, int argc, char *argv[]);
static  int             ScanOutput (Tcl_Interp *Interp, int argc, char *argv[]);
static  int             ScanRightGain (Tcl_Interp *Interp, int argc, char *argv[]);
static  int             ScanVolume (Tcl_Interp *Interp, int argc, char *argv[]);

/*-------------------------------------------------------------------------*/
static int ScanOption (Tcl_Interp *Interp, int argc, char *argv[])
{
        if (strcmp (*argv, "-input") == 0)
          return ScanInput (Interp, argc, argv);
        if (strcmp (*argv, "-output") == 0)
          return ScanOutput (Interp, argc, argv);
        if (strcmp (*argv, "-volume") == 0)
          return ScanVolume (Interp, argc, argv);
        if (strcmp (*argv, "-balance") == 0)
          return ScanBalance (Interp, argc, argv);
        if (strcmp (*argv, "-gain") == 0)
          return ScanGain (Interp, argc, argv);
        if (strcmp (*argv, "-leftgain") == 0)
          return ScanLeftGain (Interp, argc, argv);
        if (strcmp (*argv, "-rightgain") == 0)
          return ScanRightGain (Interp, argc, argv);
        if (strcmp (*argv, "-intspeaker") == 0)
          return ScanIntSpeaker (Interp, argc, argv);
        Interp->result = "bad option, should be -input, -output, -intspeaker, ",
                "-volume, -gain, -leftgain, -rightgain";
        return -1;
} /* ScanOption */
/*-------------------------------------------------------------------------*/
static int ScanInput (Tcl_Interp *Interp, int argc, char *argv[])
{
        int                     Input = -1;

        if (argc <= 1) {
          Interp->result = "Missing option to -input";
          return -1;
          }
        if (strcmp (argv[1], "lowgainmike") == 0)
          Input = AP_in_microphone_low;
        else if (strcmp (argv[1], "highgainmike") == 0)
          Input = AP_in_microphone_high;
        else if (strcmp (argv[1], "line1") == 0)
          Input = AP_in_line_1;
        else if (strcmp (argv[1], "line2") == 0)
          Input = AP_in_line_2;
        else if (strcmp (argv[1], "line3") == 0)
          Input = AP_in_line_3;
        else {
          Interp->result = "Bad option, should be "
                "lowgainmike, highgainmike, line1, line2, or line3";
          return -1;
          }
        ahw_set (AD_RECORD, AS_port, Input);
        return 2;
} /* ScanInput */
/*-------------------------------------------------------------------------*/
static int ScanOutput (Tcl_Interp *Interp, int argc, char *argv[])
{
        int                     Output = -1;

        if (argc <= 1) {
          Interp->result = "Missing option to -output";
          return -1;
          }
        if (strcmp (argv[1], "line1") == 0)
          Output = AP_out_line_1;
        else if (strcmp (argv[1], "line2") == 0)
          Output = AP_out_line_2;
        else if (strcmp (argv[1], "line2") == 0)
          Output = AP_out_line_3;
        else {
          Interp->result = "bad option, should be line1, line2 or line3";
          return -1;
          }
        ahw_set (AD_PLAYBACK, AS_port, Output);
        return 2;
} /* ScanOutput */
/*-------------------------------------------------------------------------*/
static int ScanIntSpeaker (Tcl_Interp *Interp, int argc, char *argv[])
{
        int                     Val = -1;

        if (argc <= 1) {
```

```
          Interp->result = "Missing option to -intspeaker";
          return -1;
          }
        if (strcmp (argv[1], "on") == 0)
          Val = 1;
        else if (strcmp (argv[1], "off") == 0)
          Val = 0;
        if (strcmp (argv[1], "1") == 0)
          Val = 1;
        else if (strcmp (argv[1], "0") == 0)
          Val = 0;
        else {
          Interp->result = "bad option, should be on, 1, off or 0";
          return -1;
          }
        ahw_set (AD_PLAYBACK, AS_port, AP_out_speaker);
        if (Val)
          ahw_set (AD_PLAYBACK, AS_gain, 255);
        else
          ahw_set (AD_PLAYBACK, AS_gain, 0);
        return 2;
} /* ScanIntSpeaker */
/*-------------------------------------------------------------------------*/
static int ScanVolume (Tcl_Interp *Interp, int argc, char *argv[])
{
        int                     Val = -1;

        if (argc <= 1) {
          Interp->result = "Missing option to -volume";
          return -1;
          }
        Val = atoi (argv[1]);
        if (Val < 0)
          Val = 0;
        else if (Val > 100)
          Val = 100;
        ahw_set (AD_PLAYBACK, AS_gain, 256 * Val / 100);
        return 2;
} /* ScanVolume */
/*-------------------------------------------------------------------------*/
static int ScanBalance (Tcl_Interp *Interp, int argc, char *argv[])
{
        int                     Val = -1;

        if (argc <= 1) {
          Interp->result = "Missing option, to -balance";
          return -1;
          }
        Val = atoi (argv[1]);
        if (Val < -100)
          Val = -100;
        else if (Val > 100)
          Val = 100;
        ahw_set (AD_PLAYBACK, AS_balance, 256 * Val / 100);
        return 2;
} /* ScanBalance */
/*-------------------------------------------------------------------------*/
static int ScanGain (Tcl_Interp *Interp, int argc, char *argv[])
{
        int                     Val = -1;

        if (argc <= 1) {
          Interp->result = "Missing option to -gain";
          return -1;
          }
        Val = atoi (argv[1]);
        if (Val < 0)
          Val = 0;
        else if (Val > 100)
          Val = 100;
        ahw_set (AD_RECORD, AS_gain, 256 * Val / 100);
        return 2;
} /* ScanGain */
/*-------------------------------------------------------------------------*/
static int ScanLeftGain (Tcl_Interp *Interp, int argc, char *argv[])
{
        int                     Val = -1;

        if (argc <= 1) {
          Interp->result = "Missing option, to -leftgain";
          return -1;
          }
        Val = atoi (argv[1]);
        if (Val < 0)
          Val = 0;
        else if (Val > 100)
          Val = 100;
        ahw_set (AD_RECORD, AS_gain_left, 256 * Val / 100);
        return 2;
} /* ScanLeftGain */
/*-------------------------------------------------------------------------*/
static int ScanRightGain (Tcl_Interp *Interp, int argc, char *argv[])
{
        int                     Val = -1;

        if (argc <= 1) {
          Interp->result = "Missing option, to -rightgain";
          return -1;
          }
```

```
        Val = atoi (argv[1]);
        if (Val < 0)
          Val = 0;
        else if (Val > 100)
          Val = 100;
        ahw_set (AD_RECORD, AS_gain_right, 256 * Val / 100);
        return 2;
} /* ScanRightGain */
/*--------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------*/
char cmd_audioctl (ClientData *cd, Tcl_Interp *Interp, int argc, char *argv[])
/*
 * Implements the new tcl/tk audio command:
 * audioctl <option> <value> [...]
 *      -input          line1, line2, line3, lowgainmike, highgainmike,
 *      -gain           0..100 (sets left and right gain)
 *      -leftgain       0..100
 *      -rightgain      0..100
 *
 *      -output         line1, line2, line3
 *      -intspeaker     on, off, 0, 1
 *      -volume         0..100
 *      -balance        -100..0..100
 */
{
        int                     n;

        if (argc <= 1) {
          Interp->result = "Usage: audio ...";
          return TCL_ERROR;
          }
        argc--, argv++;
        while (argc > 0) {
          if ((n = ScanOption (Interp, argc, argv)) <= 0)
            return TCL_ERROR;
          argc -= n;
          argv += n;
          } /* while */
        return TCL_OK;
} /* cmd_audioctl */
```

Literaturverzeichnis

[ASZ95] D. Atkins, W. Stallings, and P. Zimmermann. Pgp message exchange formats. Internet-draft from ietf, Internet Engineering Task Force, November 1995.

[BD93] C. Mic Bowman and Chanda Dharap. The enterprise distributed white-page service. pages 349–359, San Diego, California, January 1993. Usenix.

[BF93] N. Borenstein and N. Freed. MIME (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies. Request for Comments (Draft Standard) RFC 1521, Internet Engineering Task Force, September 1993. (Obsoletes RFC1341); (Updated by RFC1590).

[BLFF95] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – http/1.0. In *IETF Internet-Draft for HTTP/1.0, spec 03*. IETF, September 1995.

[BLMM94] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (URL). Request for Comments (Proposed Standard) RFC 1738, Internet Engineering Task Force, December 1994.

[Bra89] R. Braden. Requirements for internet hosts - application and support. Request for Comments (Standard) STD 3, RFC 1123, Internet Engineering Task Force, October 1989.

[Cro82] D. Crocker. Standard for the format of ARPA internet text messages. Request for Comments (Standard) STD 11, RFC 822, Internet Engineering Task Force, August 1982. (Obsoletes RFC0733); (Updated by RFC1327, RFC0987).

[Dee88] S. Deering. Host extensions for IP multicasting. Request for Comments RFC 1054, Internet Engineering Task Force, May 1988. (Obsoletes RFC0988).

[Dee89a] S. Deering. Host extensions for IP multicasting. Request for Comments (Standard) STD 5, RFC 1112, Internet Engineering Task Force, August 1989. (Obsoletes RFC0988).

[Dee89b] S. Deering. Ip multicast extensions for 4.3bsd unix and related systems. Technical report, Stanford University, June 1989. ftp://gregorio.stanford.edu/vmtp-ip/ipmulticast.tar.Z.

[GV95] A. Gulbrandsen and P. Vixie. A dns rr for specifying the location of services. Internet-draft from ietf, Internet Engineering Task Force, November 1995.

[HJ95] Mark Handley and Van Jacobson. Sdp:session description protocol (draft). In *IETF Internet-Draft of SDP, spec 01*. IETF, July 1995.

[Hop94] Andy Hopper. Communication at the desktop. 10:1253–1265, July 1994.

[ITU94] ITU. One-way transmission time, transmission quality for an entire international telephone connection. Recommendation of the ITU ITU-T Recommendation G.114, International Telecommunication Union, July 1994.

[Ltd95] VocalTec Ltd. Vocaltec internet phone. Technical report, August 1995.

[Man91] Sanjay Manandhar. Acitivity server: You can run but you can't hide. pages 299–311, Nashville, TN, June 1991. Usenix.

[Mea95] R. McCool and et al. NCSA HTTP server 1.5. Technical report, National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign, October 1995. ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a-export_source.tar.Z.

[Moc87a] P. Mockapetris. Domain names - concepts and facilities. Request for Comments (Standard) STD 13, RFC 1034, Internet Engineering Task Force, November 1987. (Obsoletes RFC0973); (Updated by RFC1101).

[Moc87b] P. Mockapetris. Domain names - implementation and specification. Request for Comments (Standard) STD 13, RFC 1035, Internet Engineering Task Force, November 1987. (Obsoletes RFC0973); (Updated by RFC1348).

[Oer95] Frank Oertel. *Aufbau und Konfiguration lokaler Multimedia-Konferenz-Umgebungen*. TU-Berlin, GMD Fokus, November 1995.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, Massachusetts, 1994.

[Sch95] Henning Schulzrinne. *Guide to NeVoT*. GMD Fokus, Berlin, Germany, 3.32 edition, September 1995.

[Shi94] Dave Shield. *Resparse Library*, February 1994. ftp://hpux.csc.liv.ac.uk/hpux/Networking/Admin/resparse-1.1.tar.gz.

[SR95] B. C. Smith and L. A. Rowe. Tcl distributed programming. Technical report, Department of Computer Science, Cornell University, June 1995. ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/Tcl-DP/tcl-dp3.3b1.tar.gz.

[Sto94] René Stolp. Lokalisierungsdienst f r Mitarbeiter zur Teilnahme an AV-Konferenzen auf Login-Basis. Instructor: Tom Pfeifer, 1994.

[Vix95] Paul Vixie. *DNS bind Version 4.9.3-beta21*, June 1995. ftp://ftp.vix.com/pub/bind-4.9.3.

[WH92] Roy Want and Andy Hopper. Active badges and personal interactive computing objects. Technical Report ORL 92-2, Olivetti Research, Cambridge, England, February 1992. also in *IEEE Transactions on Consumer Electronics*, Feb. 1992.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. 10(1):91–102, January 1992. also Olivetti Research Limited Technical Report ORL 92-1.