Studienarbeit
Fachhochschule Bielefeld

Betreuender Dozent: Professor Dr.-Ing. Lutz Grünwoldt

Asterisk – ein Überblick

Felix J. Ogris (203583)
felix_juergen.ogris@fh-bielefeld.de

23. Januar 2007

Inhaltsverzeichnis

1. Einleitung

Das in dieser Studienarbeit diskutierte Softwarepaket *Asterisk* stellt eine Telefonanlage mit zwei grundlegenden Eigenschaften dar: Zum einen wird Asterisk als Open-Source vertrieben. Es unterliegt der *GNU General Public License* (GPL) und kann somit von jedermann weitestgehend frei eingesetzt und modifiziert werden. Zum anderen ist es auf herkömmlicher PC-Hardware lauffähig, vorzugsweise auf einem x86-kompatiblen System unter Linux. Für Mark Spencer, den Initiator von Asterisk, waren dies im Jahre 1999 auch die Hauptmotive, sich seine eigene Telefonanlage buchstäblich zu programmieren, da ihm sowohl die Preise als auch die beschränkten Möglichkeiten damals verfügbarer Telefonsysteme missfielen. Mit dem Aufkommen von *Voice over IP*, kurz *VoIP*, wurde auch Asterisk um die Möglichkeit erweitert, über IP-basierte Netze wie Intranets oder dem Internet zu telefonieren. Dominierend ist hierbei die Kombination aus *SIP* und *RTP*, dem *Session Initiation Protocol* bzw. *Realtime Transport Protocol*. Als Endgeräte für Voice over IP kommen entweder sogenannte *Softphones*, also clientseitige Programme, die die Soundkarte bzw. Mikrofon und Kopfhörer eines PCs verwenden, oder *Hardphones* in Frage, welche herkömmlichen Telefon ähneln, aber die Gesprächsdaten über ein IP-Netz versenden. Die Verbindung zu (leitungsvermittelnden) Telefonnetzen kann über simple ISDN-Steckkarten erfolgen, sofern sie mittels eines CAPI-Treibers oder über die ISDN4Linux-Schnittstelle vom Betriebssystem unterstützt werden. Allerdings sind pro solcher Karte maximal 2 simultane Gespräche möglich. Mehrere D-Kanäle bieten spezielle Controller der ebenfalls von Mark Spencer gegründeten Firma *Digium*, die ausserdem Interfacekarten für analoge Telefonie vertreibt. Neben der reinen Gatewayfunktion zwischen verschiedenen Audiocodecs und Signalingmethoden bietet Asterisk die Möglichkeit, Warteschlangen oder *Queues*, wie sie z.B. in Callcentern verwendet werden, oder *Voicemailboxen* einzurichten, welche als Anrufbeantworter fungieren. Der Rufnummernplan oder der *Dialplan*, welcher die Zuordnung von Telefonnummern zu Endgeräten vornimmt, ist frei konfigurierbar. Somit gleicht der Dialplan vielmehr einer Routingtabelle. Da Asterisk als gewöhnlicher Serverdienst auf einem Linuxrechner läuft, kann es über das sogenannte *Asterisk Manager Interface*, kurz *AMI*, sehr einfach angesteuert werden. Diese Schnittstelle wird in einem Teil der vorliegenden Studienarbeit verwendet, um Telefonate zwischen zwei Teilnehmern zu vermitteln. Hierzu wurde eine Software entwickelt, die Rufnummern, die Benutzern aus einem Active Directory zugeordnet sind, in eine lokale Datenbank synchronisiert, so dass ein Anwender einen anderen Teilnehmer komfortabel mittels eines Mausklicks aus einer Weboberfläche heraus anrufen kann.

2. Aufgabenstellung und Szenarien

Ziel war die Demonstration realisierbarer Szenarien und weitergehender Möglichkeiten, die sich aus dem Einsatz eines Asterisk-Servers ergeben. Hierzu standen mehrere ältere PCs (Pentium 2 400 MHz, 128 MB RAM, IDE-Fesplatte) mit vorinstalliertem SuSE Linux 10.0 und die Netzwerkinfrastruktur im Labor für Angewandte Informatik und Mathematik der FH Bielefeld zur Verfügung. Aus der im Internet verfügbaren Menge von Softphones sollten vornehmlich diejenigen verwendet werden, die kostenlos erhältlich und einsetzbar sind und die zumindest ein uneingeschränktes Telefonieren ermöglichen. Um Hardware wie ISDN-Karten oder IP-Telefone beschaffen zu können, standen Geldmittel in Höhe von bis zu 500 Euro zur Verfügung. Ferner sollte eruiert werden, ob und wie eine sinnvolle Weiterverwendung eines Asterisk-Servers im Labor und in der Veranstaltung *Netzwerke/Verteilte Anwendungen* (NW) bzw. im neu geschaffenen CCNA-Kurs möglich ist. Um die einfache Ansteuerung eines Asterisk-Servers über das Asterisk Manager Interface zu zeigen, wurde zusätzlich eine webbasierte Datenbankanwendung erstellt. Diese speichert Telefonnummern, welche den in einem Active Directory gepflegten Benutzern zugeordnet sind, in einer lokalen MySQL-Datenbank. Eine Weboberfläche greift auf diese Datenbank zu, so dass ein Anwender ohne Wählen einer Telefonnummer eine Verbindung von seinem Telefon zu dem des gewünschten Teilnehmers aufbauen kann.

Üblicherweise wird ein Asterisk-Server als Vermittler zwischen einem oder mehreren

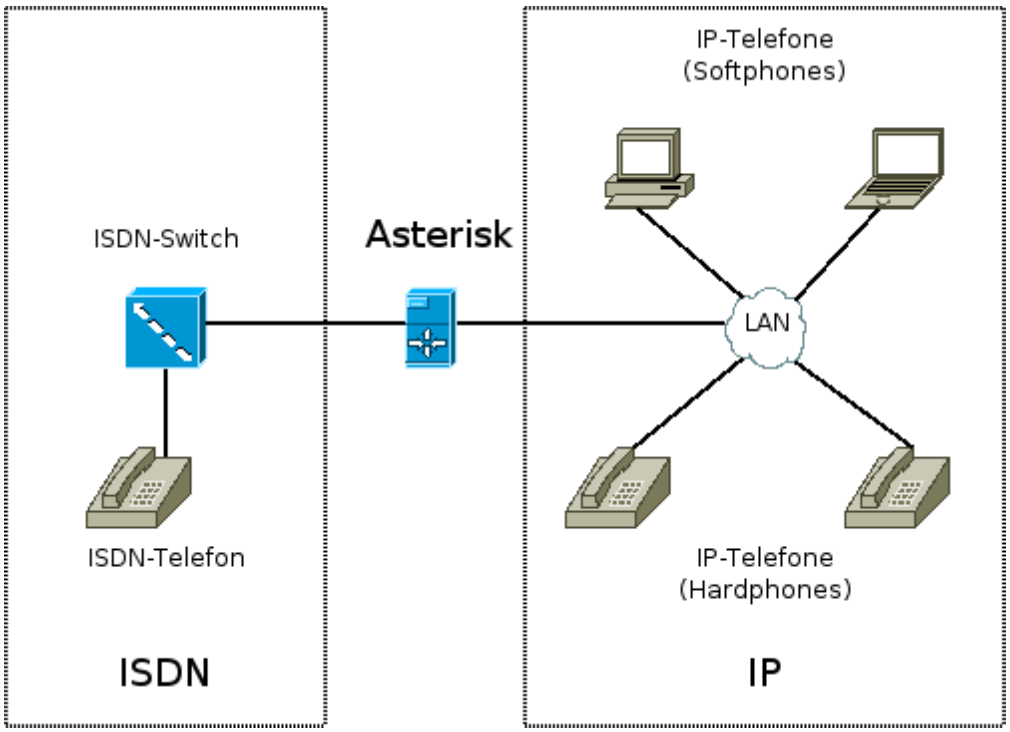

Abbildung 1: Das realisierte Szenario: Asterisk als Gateway zwischen IP- und ISDN-Telefonie

hausinternen und hausexternen Netzen eingesetzt. Prinzipiell stehen 3 Arten von Schnittstellen zur Verfügung: analog, ISDN und IP, so dass sich unter Beachtung aller Kombinationen 49 theoretische Szenarien ergeben (analog und/oder ISDN und/oder IP, sowohl intern als auch extern). Hieraus wurde das in Abbildung 1 gezeigte Szenario ausgewählt,

da es

- die Demonstration von Hard- und Softphones ermöglicht
- die modellhafte Anbindung an das ISDN-Netzes eines Telekommunikationsbetreibers zeigt
- Asterisk als Protokollumsetzer zwischen der ISDN- und IP-Welt betreibt
- mit moderatem Hardwareaufwand realisierbar ist
- die viel zitierte Konvergenz zwischen Telefon- und Computernetzen zeigt, da hausintern nur noch eine gemeinsame Netzwerkinfrastruktur (hier: IP über Ethernet) benötigt wird.

Abbildung 2: Grandstream GXP-2000

Abbildung 3: AVM FRITZ!Card PCI

Zur Simulation eines ISDN-Netzes stand im Labor ein ISDN-Switch der Firma Agfeo zur Verfügung. Dieser wurde in Zusammenarbeit mit dem Laboringenieur Herrn Manfred Fingberg so konfiguriert, dass an Port 11 und Port 12 ein ISDN-Telefon bzw. der Asterisk-Server betrieben werden konnten.
Analoge Telefonie wurde nicht weiter beachtet, da es sich um eine rückläufige Technik handelt und da sie nur mittels spezieller Interfacekarten (s. Kapitel 1) realisieren lässt. Ebenfalls wurde darauf verzichtet, einen internen S0-Bus zu betreiben, um so eine eventuell vorhandene ISDN-Installation direkt an den Asterisk-Server anzuschliessen. Hierfür wäre eine ISDN-Steckkarte nötig, deren Chipsatz im sogenannten *NT-Modus* arbeitet und sich gegenüber ISDN-Telefonen wie eine Vermittlungsstelle verhält.
Um das geforderte Szenario realisieren zu können, wurden 2 IP-Telefone vom Typ *Grandstream GXP-2000* (Abbildung 2) sowie eine ISDN-Karte *AVM FRITZ!Card PCI v2.1* (Abbildung 3) bestellt.

3. Asterisk

3.1. Kurzvorstellung

Asterisk läuft als normaler Serverdienst auf einem Linuxrechner. In einer Laborumgebung ist der Betrieb mit Superuserrechten, sprich *root-Rechten*, zu empfehlen. Für den Einsatz auf einem Produktivsystem sollte hingegen ein eigener Benutzeraccount für den Asterisk-Daemon eingerichtet werden, um so die Kompromittierung des Servers durch einen Fehler in einem Asterisk-Modul zu vermeiden. Allerdings muss bei Verwendung eines eigenen Benutzeraccounts gewährleistet sein, dass dieser auf alle Konfigurationsdateien, Spoolverzeichnisse und auf Gerätedateien im Verzeichniss `/dev` Zugriffsrechte hat. Der Server wird entweder beim Hochfahren des Systems über ein entsprechendes Skript in `/etc/init.d` bzw. `/etc/rc.d` (hier variiert jede Linux-Distribution) oder auf der Konsole durch den direkten Aufruf von `asterisk` gestartet, welches im Verzeichnis `/usr/sbin` liegt. Konfigurationsdateien werden in `/etc/asterisk` erwartet. Die eigentlichen Funktionen des Servers sind in dynamische Bibliotheken, sogenannte *shared objects* (meist mit der Dateiendung `.so`) ausgelagert. Diese sind unterhalb des Verzeichnisses `/usr/lib/asterisk` gespeichert. Somit können zum einen bestimmte Funktionen komplett ausgeblendet werden, indem man in der Datei `/etc/asterisk/modules.conf` einen Eintrag wie `noload => modul.so` hinzufügt. Zum anderen kann Asterisk so ohne erneute Kompilierung um eigene Routinen erweitert werden. Zusätzlich werden die Verzeichnishierarchien `/var/lib/asterisk` und `/var/spool/asterisk` benötigt. Unterhalb von `/var/lib/asterisk` werden u.a. Wartemelodien, Ansagetexte und eigene Skripte hinterlegt, während `/var/spool/asterisk` temporäre Dateien aufnimmt, wie z.B. noch nicht abgerufene Nachrichten einer Voicemailbox. Die Kommunikation mit ISDN-Karten erfolgt entweder über einen CAPI-Treiber bzw. einer Gerätedatei wie `/dev/ttyI0` oder, sofern es sich um eine (Primärmultiplex-)Karte der Firma Digium handelt, mittels eines Treibers aus dem *Zaptel*-Paket, welches ebenfalls unter der GPL vertrieben wird.

3.2. Installation

3.2.1. Abhängigkeiten

Die verwendete Linux-Distribution SuSE 10.0 wird ohne Asterisk-Paket geliefert. Eine Installation über das Softwareverwaltungswerkzeug *YaST* war daher nicht möglich, so dass eine Übersetzung aus den Quelltexten unumgänglich war. Zuvor wurden der Datenbankserver *PostgreSQL*, der MP3-Player *mpg123* und die *Zaptel*-Treiber ebenfalls kompiliert und installiert, da sonst der Übersetzungsvorgang von Asterisk das Fehlen jener Programme bemerkt und z.B. das Modul zur Anbindung einer PostgreSQL-Datenbank (`cdr_pgsql.so`) nicht übersetzt. Alle weiterhin benötigten Werkzeuge lassen sich über YaST installieren. Dies sind:

- der GNU C Compiler *gcc*
- Headerdateien und weitere Bibliotheken der Systemlibrary *glibc-devel*
- den Sourcecode des Linuxkernels *kernel-devel*
- die Readline-Bibliothek *readline* und ihre Headerdateien *readline-devel*
- *openssl* und *openssl-devel*
- *ncurses* und *ncurses-devel*

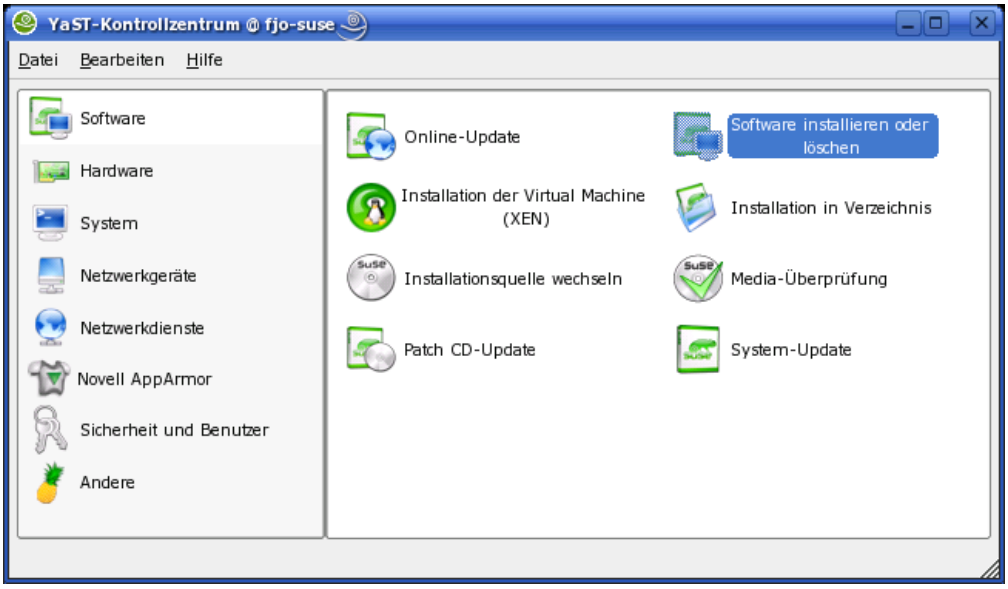

Abbildung 4: YaST

- *zlib* und *zlib-devel*
- das Programm *doxygen*, um die Dokumentation aus dem Asterisk-Quelltext erstellen zu können (vgl. JavaDoc)

Natürlich sollten alle vorgeschlagenen Abhängigkeiten ebenfalls installiert werden. Es empfiehlt sich, für die einzelnen Installationsschritte ein separates Verzeichnis namens `src` o.ä. im Homeverzeichnis anzulegen und dort alle Quellpakete abzuspeicheren.

3.2.2. PostgreSQL

Das Entpacken, Übersetzen und Installieren von PostgreSQL gelingt unter Verwendung folgender Befehle:

```
tar -xjf postgresql-8.1.5.tar.bz2
cd postgresql-8.1.5
./configure --with-openssl --enable-integer-datetimes
make
make install
```

Danach wird ein eigener Account für den Datenbankserver eingerichtet und ein Verzeichnis mit passenden Zugriffsrechten für die eigentlichen Daten angelegt:

```
useradd -c "PostgreSQL server" -s /bin/false -d /usr/local/pgsql \
  -g daemon -r pgsql
mkdir /usr/local/pgsql/data
chown pgsql /usr/local/pgsql/data
chmod 700 /usr/local/pgsql/data
```

Mit dem Programm `initdb` wird die Datenbank initialisiert. Dieses muss unter dem Benutzeraccount geschehen, mit dessen Rechten später der PostgreSQL-Server laufen soll, hier `pgsql`. Durch die Kommandozeilenoption `-W` wird man zusätzlich aufgefordert, ein Masterpasswort für den Zugriff auf die Datenbank zu vergeben:

```
sudo -u pgsql -- /usr/local/bin/initdb -U pgsql -W \
  /usr/local/pgsql/data
```

Damit PostgreSQL beim Systemstart hochfährt, kopiert man das eigens erstellte Skript `pgsql` (s. Anhang C in das Verzeichnis `/etc/init.d` und verknüpft es per Aufruf von `chkconfig -a pgsql` im Startprozess des Linuxsystems. Nach einem Reboot oder dem Aufruf von `/etc/init.d/pgsql start` sollte nun der PostgreSQL-Server laufen. Für den Einsatz mit Asterisk ist es sinnvoll, einen eigenen Datenbankuser samt eigener Datenbank anzulegen. Hierzu verbindet man sich mit dem Befehl
`psql -d template1 -U pgsql` und unter Eingabe des oben vergebenen Passwortes auf den PostgreSQL-Server und setzt nacheinander die Anweisungen

```
CREATE USER asterisk PASSWORD 'obelix';
```

und

```
CREATE DATABASE astdb OWNER asterisk;
```

ab. Per Eingabe von `\q` verlässt man den PostgreSQL-Client wieder. Aus Sicherheitsgründen sollte der Datenbankserver weitestgehend abgesichert werden. Hierzu lässt man in der Datei `/usr/local/pgsql/data/pg_hba.conf` als einzige nicht auskommentierte Zeile folgende übrig:

```
local all all md5
```

Somit ist sichergestellt, dass Verbindungen zum PostgreSQL-Server nur über einen lokalen Unix-Socket hergestellt werden dürfen und sich User per in der Datenbank hinterlegtem Passwort authentifizieren müssen.

3.2.3. mpg123

Die Installation des Kommandozeilen-MP3-Players mpg123 gestaltet sich recht einfach. Es genügen folgende Aufrufe:

```
tar -xjf mpg123-0.61.tar.bz2
cd mpg123-0.61
./configure
make
make install
```

3.2.4. zaptel

Asterisk benötigt für diverse Dienste wie z.B. Konferenzräume ein Timing-Device, wie es von Digium-basierten Interfacekarten bereitgestellt wird. Verfügt der Rechner nicht über solche Interfacekarten, emuliert das Kernelmodul *ztdummy* unter Verwendung des USB-Controllers ein derartiges Timing-Device. Die Installation des Zaptel-Treiberpaketes ist relativ einfach:

```
tar -xzf zaptel-1.2.11.tar.gz
cd zaptel-1.2.11
make
make install config
```

Nach einem Reboot oder dem Aufruf von `/etc/init.d/zaptel start` sollte ein Zaptel-Kernelmodul wie z.B. ztdummy geladen worden sein, was sich mit dem Aufruf von `lsmod` prüfen lässt.

3.2.5. Asterisk

Nach den oben ausgeführten Vorbereitungen beschränkt sich die Installation von Asterisk auf wenige Schritte. Nach dem Entpacken des heruntergeladenen Quellcodepaketes und dem Wechsel in das daraus neu erstellte Verzeichnis per Aufrufen von

```
tar -xzf asterisk-1.2.13.tar.gz
cd asterisk-1.2.13
```

genügt der folgende Befehlsdreisatz, um alle notwendigen Komponenten inklusive einer Beispielkonfiguration zu installieren:

```
make
make install
make samples
```

Die Dokumentation des Quellcodes wird über den Aufruf von `make progdocs` installiert. Da Asterisk wie PostgreSQL kein eigenes Startskript für SuSE Linux mitbringt, wird die selbst erstellte Datei `asterisk` (s. Anhang D) nach `/etc/init.d` kopiert. Den obligatorischen Aufruf von `chkconfig -a asterisk`, um Asterisk im Startprozess des Linuxsystems zu verankern, sollte man jedoch erst nach Abschluss aller Konfigurationsarbeiten (s. Kapitel 3.3) vornehmen.

3.2.6. ISDN-Karte

Die Installation der ISDN-Karte gestaltet sich sowohl hardware- als auch softwaretechnisch relativ einfach. Im Rechner wird lediglich ein freier 32Bit PCI-Steckplatz benötigt. Über Yast werden die beiden Pakete `avmfritzcapi` und `km_fritzcapi` installiert, um Treiber bzw. Kernelmodul dem System hinzuzufügen. Nach einem Reboot sollten entsprechende Einträge im Kernellog (Ausgabe von `dmesg`) zeigen, dass eine FRITZ!Card erkannt wurde.

3.3. Komponenten und deren Konfiguration

Asterisk besteht aus diversen einzeln konfigurierbaren Komponenten. Jeder Teil des Telefonieservers wird mittels einer eigenen, im Klartext lesbaren Datei im Verzeichnis `/etc/asterisk` gesteuert. Durch den Aufruf von `make samples` während der Installation werden alle notwendigen Dateien erstellt und mit einer Bespielkonfiguration versehen. Diese können als Grundlage für eigene Anpassungen dienen. In der Regel wird man jene Beispiele aber umbenennen (z.B. `mv sip.conf sip.conf.orig`) und sie beim Anlegen von neuen, eigenen Steuerdateien nur noch als kurze Befehlsreferenz verwenden.
Während der Konfigurationsphase ist es empfehlenswert, Asterisk im Debugmodus (s. Abbildung 5) direkt auf der Konsole zu starten. Dies erreicht man durch Eingabe von `asterisk -cdfvvv`. Somit wird zum einen der Startprozess des Linuxsystems nicht durch ein eventuell fehlerhaft konfiguriertes Asterisk beeinträchtigt. Zum anderen kann man so etwaige Fehler und Hinweise beim Hochfahren und Betrieb des Telefonieservers direkt, sprich ohne Umwege über eine Logdatei, erkennen und beheben. Für den Produktiveinsatz sollte der (korrekt konfigurierte) Asteriskserver auf jeden Fall vom Betriebssystem gestartet werden. Mit dem Kommando `asterisk -r` kann man sich dann auf einen schon laufenden Server verbinden und analog zu einer Debugsitzung Befehle an Asterisk absetzen.

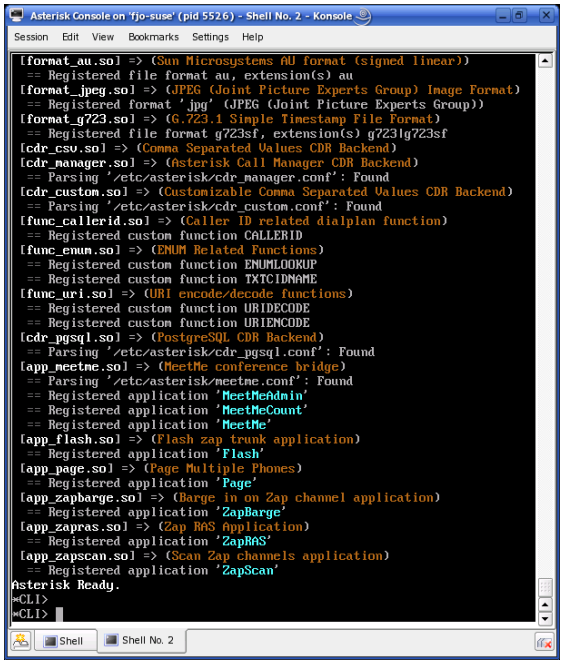

Abbildung 5: Asterisk im Debugmodus auf der Konsole

3.3.1. Dialplan

Das Herzstück einer Asteriskinstallation ist der sogenannte *Dialplan*. Mit ihm wird bestimmt, ob und wohin ein Teilnehmer beim Anwählen einer Telefonnummer weitergeleitet wird. Der Dialplan ist daher mit einer (statischen) Routingtabelle vergleichbar. Er wird in der Datei `/etc/asterisk/extensions.conf` konfiguriert und besteht hauptsächlich aus sequentiellen Zuordnungen von Telefonnummern zu bestimmten Aktionen. Formal muss eine solche Zuordnung immer wie folgt strukturiert sein:

```
exten => <Durchwahl>,<Priorität>,<Aktion>
```

Zum Beispiel ordnet folgende Zuweisung der Durchwahl `1234` die Aktion `Hangup()` zu, die - ihrem Namen entsprechend - das Telefonat beendet:

```
exten => 1234,1,Hangup()
```

Die Priorität (in obigem Beispiel ist sie 1) gibt an, in welcher Reihenfolge die einer Durchwahl zugeordneten Aktionen ausgeführt werden sollen:

```
exten => 12345,1,MP3Player(/mp3/unbekannte_nummer.mp3)
exten => 12345,2,Hangup()
```

Ruft hier ein Teilnehmer die Nummer 12345 an, so würde ihm zunächst vom Asteriskserver die MP3-Datei `/mp3/unbekannte_nummer.mp3` vorgespielt und anschliessend das Telefonat beendet werden. Um Einschübe zwischen zwei Aktionen einer Durchwahl ohne müssiges Inkrementieren aller nachfolgenden Prioritäten zu ermöglichen, kann statt expliziter Prioritäten auch der Platzhalter `n` verwendet werden. Die Abfolge der Aktionen ergibt sich somit aus der Reihenfolge, in der sie im Dialplan aufgeführt sind. Also ist nachfolgendes Beispiel zu obigem äquivalent:

```
exten => 12345,1,MP3Player(/mp3/unbekannte_nummer.mp3)
exten => 12345,n,Hangup()
```

Auf jeden Fall ist für jede Durchwahl als Startpunkt eine Aktion mit der Priorität 1 erforderlich. Telefonnummern können wie in den oben gezeigten Beispielen als feststehende Folge von Ziffern oder aber als Ausdruck mit Platzhaltern formuliert werden. Hierbei steht ein `X` für die Ziffern 0 bis 9, `Z` für 1 bis 9, `N` für 2 bis 9 und ein Punkt für ein oder mehrere beliebige Zeichen. Eine Ziffernfolge in eckigen Klammern steht stellvertretend für genau eine Ziffer aus jener Folge. Derartige Muster müssen im Dialplan mit einem anführenden Unterstrich deklariert werden. Um zum Beispiel zu verhindern, dass Telefonnummern mit vorangestellter Null angewählt werden können, wäre folgendes möglich:

```
exten => _0.,1,Hangup()
```

Das nächste Beispiel unterbindet das Wählen der in Deutschland üblichen Notrufnummern, sprich 110 und 112:

```
exten => _11[02],1,Hangup()
```

Neben der schon vorgestellten Funktion `Hangup()`, welche das Telefonat beendet, kennt Asterisk unter anderem folgende Aktionen:

MP3Player(`Datei`) spielt dem Anrufer die angegebene Datei im MP3-Format vor

Playback(`Datei`) wie `MP3Player()`, jedoch muss die Datei in einem nativ von Asterisk unterstützten Format wie μ-Law, A-Law, GSM o.ä. vorliegen

Dial(`Kanal`(&`Kanal`)(&`Kanal`) ... (,`Wartezeit [s]`)) leitet den Anrufer zu den angegebenen Kanälen, sprich Endgeräten weiter. Wird von diesen nicht eines nach der optional genannten Wartezeit abgehoben, geht Asterisk zur nächsten Aktion für diese Durchwahl weiter.

Voicemail(`Mailboxnummer`) leitet zur angegebenen Voicemailbox um, so dass der Anrufer nach einem Begrüssungstext („Bitte hinterlassen Sie ihre Nachricht nach dem Ton...“) eine Sprachnotiz aufsprechen kann. Siehe auch Kapitel 3.3.6

VoiceMailMain(`Mailboxnummer`) leitet zur angegebenen Voicemailbox, um so dass der Anrufer die aufgesprochenen Nachrichten abhören und ggf. löschen kann

MeetMe(`Konferenzraumnummer`) leitet den Anrufer zum angegebenen Konferenzraum weiter. Siehe auch Kapitel 3.3.5

Answer() veranlasst Asterisk, das Telefonat anzunehmenen und es selbst mit einer Aktion wie z.B. `Playback()` oder `MeetMe()` zu beantworten anstatt es per `Dial()` auf einen anderen Kanal weiterzuleiten

Page(`Kanal`(&`Kanal`)(&`Kanal`) ...) verbindet den Anrufer mit allen angegebenen Kanälen. Beim Anrufer wird für die Dauer des Gespräches der Hörer deaktiviert, bei den angerufenen Teilnehmern das Mikrofon

System(`Befehl`) führt den angegebenen (Linux-)Befehl aus

Ein Kanal besteht immer aus einem Protokoll und einer Rufnummer oder sonstigen Teilnehmerkennung, zum Beispiel `SIP/freund`. Verwendet man diesen Kanal in einem Dial-Befehl, würde Asterisk versuchen, per SIP einen Teilnehmer namens *freund* zu erreichen. Natürlich muss dieser Teilnehmer in der Konfigurationsdatei `sip.conf` (s. Kapitel 3.3.2) eingerichtet sein.

Im Dialplan können Aktionen nicht nur anhand der gewünschten Zielnummer, sondern auch unter Berücksichtigung der Herkunft des Anrufes ausgeführt oder gar gezielt gefiltert werden. Hierzu gibt es sogenannte *Kontexte*. Diese werden z.B. in der Datei `sip.conf` einem oder mehreren Teilnehmern zugeordnet, welche ihre eigene Sicht auf den Dialplan erhielten. Weist man z.B. einem SIP-Teilnehmer den Kontext `darfnix` zu, so könnte man ihm mit folgendem Dialplan alle abgehenden Anrufe untersagen:

```
[darfnix]
exten => _.,1,Hangup()

[default]
...
```

Ein `default`-Kontext wird von Asterisk immer gefordert. Zur Vermeidung von Redundanzen kann ein Kontext einen anderen einbinden; das entsprechende Schlüsselwort hierfür lautet `include`:

```
[technik]
include => ortsgespraeche
include => ferngespreache

[vertrieb]
include => ortsgespraeche
```

In obigem Beispiel dürften alle zum Kontext `technik` gehörenden Teilnehmer Orts- und Ferngespräche führen (sofern sich eben hinter diesen Kontexten entsprechende Aktionen verbergen), während der Vertrieb nur innerörtliche Telefonate führen darf.
Als weitere Erleichterung können im Dialplan Variablen verwendet werden. Wird z.B. der Kanal `SIP/teilnehmer1` in mehreren (Dial-)Aktionen verwendet, so ist es sinnvoll, statt dessen eine aussagekräftige Variable zu vergeben:

```
[globals]
TEILNEHMER1=SIP/teilnehmer1
```

Der Bezeichner `TEILNEHMER1` kann nun in einem Dial-Befehl verwendet werden:

```
exten => 1234,1,Dial(${TEILNEHMER1}, 20)
```

Eine besondere Bedeutung hat die vordefinierte Variable `EXTEN`. In ihr hält Asterisk die aktuelle Durchwahl fest, so dass folgende Zeilen äquivalent sind:

```
exten => 1234,1,Dial(SIP/1234, 20)
exten => 1234,1,Dial(SIP/${EXTEN}, 20)
```

Zusätzlich kann man nur einen Teil dieser Variablen auswerten lassen:

```
exten => _5000XXXX,1,VoiceMailMain(${EXTEN:4})
```

In obigem Beispiel werden die ersten 4 Ziffern der angewählten Telefonnummer (also `5000`) abgeschnitten und die nur verbleibenden würden der Funktion `VoiceMailMain` übergeben. Gibt man den Index als negative Zahl an, z.B. `${EXTEN:-2}`, werden hingegen die letzten 2 Ziffern geliefert.

3.3.2. Session Initiation Protocol

Die Kombination aus Session Initiation Protocol und Realtime Transport Protocol (RTP) hat derzeit im VoIP-Umfeld die grösste Verbreitung. Das SIP dient lediglich als Signaling-Protokoll zum Aufbau einer Verbindung zwischen den Teilnehmern, ähnlich dem D-Kanal im ISDN. RTP hingegen wird als Container für die eigentlichen Audiodaten verwendet, die per μ-Law, A-Law, GSM usw. codiert sind. Während RTP ein binäres Protokoll darstellt, ist SIP als Klartextprotokoll dem *Hypertext Transport Protocol* (HTTP), welches i.d.R. zum Abruf von Webseiten verwendet wird, sehr ähnlich. Allerdings wurde eine Unart des *File Transfer Protocol* (FTP) übernommen: SIP hinterlegt in den Nutzdaten die IP-Adresse, auf der ein Client Verbindungen (z.B. für RTP-Ströme) annehmen kann. Dies führt besonders bei Verwendung von *Network Address Translation* (NAT) auf Routern zwischen zwei SIP-Teilnehmern zu Problemen, da zwar die Adressen der IP-Pakete geändert werden, nicht aber die in der Payload. Asterisk bietet daher für die Konfiguration eines SIP-Clients die Option `nat = yes` an, mit der jegliche in SIP-Paketen angegeben IP-Adressen ignoriert werden und nur die tatsächliche Absendeadresse des Clients verwendet wird. In der Datei `/etc/asterisk/sip.conf` werden sämtliche SIP-Verbindungen definiert. Sie ist syntaktisch ähnlich zum Dialplan. Jedoch definieren Bezeichner in eckigen Klammern keine Kontexte, sondern einzelne SIP-Clients. Eine besondere Bedeutung hat der mit `[general]` eingeleitete Abschnitt, mit dem globale Einstellungen vorgenommen werden:

```
[general]
bind=0.0.0.0
port=5060
disallow=all
allow=ulaw
allow=alaw
language=de
```

Die Parameter `bind` und `port` geben an, auf welcher IP-Adresse und welchem Port der SIP-Server lauschen soll. Hierbei stehen `0.0.0.0` und `5060` für alle Netzwerkschnittstellen des Systems bzw. für den üblicherweise verwendeten SIP-Port. Die Direktiven `disallow` und `allow` verbieten bzw. erlauben die Verwendung von speziellen Audiocodecs, so dass im obigen Beispiel lediglich μ- und A-Law zugelassen sind. Zudem wird Deutsch als Sprache für Menüs wie z.B. die Ansage einer Voicemailbox festgelegt. Hierzu muss natürlich das entsprechende deutsche Sparchset installiert sein (s. Kapitel 3.3.9), sonst werden die mit Asterisk installierten englischen Texte verwendet. Die Definition eines SIP-Clients geschieht wie folgt:

```
[teilnehmer1]
context=default
type=friend
```

Der Bezeichner `teilnehmer1` kann nun als Teil eines Kanals im Dialplan verwendet werden, z.B. in der Funktion `Dial(SIP/teilnehmer1, 20)`. Der Kontext bestimmt, welchen Teil des Dialplans dieser SIP-Client sieht, wie im Kapitel 3.3.1 erläutert. Mit dem Parameter `type` wird bestimmt, wie der Teilnehmer behandelt wird. Folgende drei Optionen sind möglich:

user Teilnehmer kann nur anrufen, nicht aber angerufen werden

peer Teilnehmer kann nur angerufen werden, nicht aber anrufen

friend Kombination aus *user* und *peer*

Die Authentifizierung des Clients erfolgt per Benutzername-/Passwortkombination, wobei der Username nicht zwingend dem Bezeichner für den SIP-Client (hier: `teilnehmer1`) entsprechen muss:

```
username=teilnehmer_1
secret=streng_geheim
```

Ferner gibt es die Optionen `permit` und `deny`, mit denen die IP-Adresse oder der Netzbereich eingeschränkt werden kann, aus dem sich der SIP-Client verbinden darf:

```
deny=0.0.0.0/0
permit=192.168.1.0/24
host=dynamic
```

Somit darf sich dieser SIP-Client nur aus dem IP-Bereich 192.168.1.0 bis 192.168.1.255 am Server anmelden. Mit der Option `host=dynamic` wird erzwungen, dass sich der Client am Server registrieren muss. Viele IP-Telefone können noch nicht abgehörte Nachrichten auf der Voicemailbox mittels einer LED oder einem Hinweisfeld anzeigen. Hierzu muss allerdings dem SIP-Teilnehmer eine derartige Mailbox zugeordnet sein:

```
mailbox=1234
```

Die angegebene Voicemailbox (hier: `1234`) muss natürlich in der Datei `voicemail.conf` eingerichtet sein (s. Kapitel 3.3.6). Mit dem Parameter `callerid` können Name und Telefonnummer, die bei einem angerufenen Teilnehmer im Display erscheinen, vorgegeben werden:

```
callerid=Vorname Nachname <4711>
```

Mit den erläuterten SIP-Optionen kann ein einfaches Laborszenario eingerichtet werden. Die Vorstellung aller Parameter würde an dieser Stelle jedoch den Rahmen sprengen und wäre letztendlich nur eine Abschrift von Quellen wie Meggelen u. a. (2005).

3.3.3. IAX / IAX2

Das *InterAsterisk eXchange* oder kurz *IAX* ist ein binäres, quelloffenes Protokoll, welches Mark Spencer ursprünglich zur Verbindung von Asterisk-Servern untereinander entworfen hat. Es umgeht die Schwächen von SIP/RTP wie Probleme bei Verwendung von NAT, indem es nur einen UDP-Port verwendet und keine IP-Adressen im Datenstrom einbindet. Da die Version 2 dieses Protokolls das ursprüngliche IAX vollständig verdrängt hat, werden heutzutage IAX und IAX2 synonym verwendet. Es existieren inzwischen einige Softwareclients für IAX, eine Unterstützung durch Hardphones ist kaum gegeben.
Die Konfiguration von IAX unter Asterisk gleicht der von SIP. Zu Beginn der Datei `/etc/asterisk/iax.conf` wird ebenfalls ein Abschnitt mit generischen Einstellungen erwartet:

```
[general]
bindport=4569
bindaddr=0.0.0.0
language=de
disallow=all
allow=gsm
allow=ulaw
allow=alaw
```

Die Kommunikation mit dem IAX-Server erfolgt über UDP-Port 4569; zusätzlich wird die aus dem Mobilfunk bekannte GSM-Codecfamilie zugelassen. Die Definition einzelner IAX-Clients ist identisch zu der für SIP-Clients:

```
[teilnehmer2]
context=default
type=friend
username=teilnehmer2
secret=streng_geheim
host=dynamic
deny=0.0.0.0/0.0.0.0
permit=192.168.0.0/24
mailbox=5678
callerid=Vorname Nachname <5678>
```

Über die Optionen `deny` bzw. `permit` kann auch hier eine IP-Filterliste realisiert werden und somit sehr genau bestimmt werden, aus welchen Netzen sich `teilnehmer2` verbinden darf.

3.3.4. Call Detail Record Engine

Mit Hilfe der *Call Detail Record Engine* (CDR) können Metadaten für jedes Gespräch aufgezeichnet werden, so dass z.B. für Abrechnungszwecke festgestellt werden kann, wer wann mit wem wie lange telefoniert hat oder - falls der Angerufene nicht abgenommen hat - telefonieren wollte. Per default ist CDR aktiviert. Explizit wird es in der Datei `/etc/asterisk/cdr.conf` im Abschnitt `[general]` ein- oder ausgeschaltet:

```
[general]
enable=yes
```

Nach einer Standardinstallation loggt Asterisk Verbindungsdaten in zwei Klartextdateien: `/var/log/asterisk/cdr-csv/Master.csv` und `/var/log/asterisk/cdr-custom/Master.csv`. Diese *kommaseparierten* Listen (*comma separated values*, kurz *csv*) lassen sich mit Programmen wie *OpenOffice Calc* auslesen. Komfortabler ist der Einsatz einer SQL-Datenbank wie *PostgreSQL*. Mit dem Skript `postgres_cdr.sql` im Unterverzeichnis `contrib/scripts` des Asterisk-Quellcodes wird zunächst eine Tabelle namens `cdr` angelegt. Dieses Skript wird wie folgt dem PostgreSQL-Client `psql` übergeben:

```
psql -d astdb -U asterisk -f postgres_cdr.sql
```

Nach Eingabe des in Kapitel 3.2.2 vergebenen Passwortes für den Asterisk-User steht die Tabelle `cdr` zur Verfügung. Mit der SQL-Anweisung `SELECT * FROM cdr` im Programm psql (Aufruf wie gewohnt per psql -d astdb -U asterisk) erhält man die noch leere Tabelle samt Feldnamen. Folgene Werte werden von Asterisk aufgezeichnet:

AcctId ein für jeden Datensatz eindeutiger numerischer Schlüssel, der von der Datenbank vergeben wird

calldate Datum und Zeitpunkt, an dem der Datensatz eingetragen wurde, gemeinhin Datum und Uhrzeit des Gesprächsendes

clid CallerID (Name und Nummer) des Anrufenden

src CallerID (nur Nummer) des Anrufenden

dst die gewählte Nummer bzw. Extension

dcontext der Dialplan-Kontext

channel der (temporäre) Kanal, der dem anrufenden Gerät zugeordnet wurde

dstchannel der Kanal des angerufenen Teilnehmers

lastapp die zuletzt vom Dialplan während des Gespräches ausgeführte Funktion, z.B. `Dial()` oder `Hangup()`

lastdata die Parameter, die der unter `lastapp` aufgeführten Funktion übergeben wurden

duration Gesprächsdauer in Sekunden vom Beenden des Wählens bis zum Auflegen eines Teilnehmers

billsec Gesprächsdauer in Sekunden vom Abheben des Angerufenen bis zum Auflegen eines Teilnehmers

disposition Kurzinfo über den Exitcode des Gespräches, mögliche Werte sind `ANSWERED`, `BUSY`, `NO ANSWER` oder `FAILED`

amaflags Flag für Abrechnungszwecke (vgl. *Automatic Message Accounting*); wird entweder durch den Parameter `amaflags` in der `sip.conf` u.ä. oder im Dialplan durch die Funktion `SetAMAFlags()` gesetzt werden. Mögliche Werte sind `default`, `omit`, `billing` und `documentation`

accountcode wie *amaflags* ebenfalls für Abrechnungszwecke, jedoch handelt es sich um einen frei setzbaren numerischen Wert

uniqueid ein für jeden Datensatz eindeutiger numerischer Schlüssel, der von Asterisk vergeben wird

userfield ein beliebig verwendbares Feld, welches z.B. durch die Funktionen `AppendCDRUserField()` oder `SetCDRUserField()` belegt werden kann

Asterisk erwartet die Verbindungsinformationen für die PostgreSQL-Datenbank in der Datei `/etc/asterisk/cdr_pgsql.conf`. Hier müssen Hostname, ggf. Portnummer, Username, Passwort und Datenbank- und Tabellenname hinterlegt sein:

```
[global]
hostname=/tmp
port=5432
dbname=astdb
password=obelix
user=asterisk
table=cdr
```

PostgreSQL kann nicht nur über einen Internetsocket, sondern auch über einen lokalen Unix-Socket angesprochen werden, was einen kleinen Geschwindigkeitsvorteil bringt. Hierzu gibt man dem Parameter `hostname` nicht einen tatsächlichen Hostnamen oder eine IP-Adresse, sondern den Namen des Verzeichnisses, in dem der zugehörige Unix-Socket liegt. Bei einer Standardinstallation von PostgreSQL ist dies `/tmp`. Natürlich

müssen Anwendung und Datenbankserver hierbei auf der selben Maschine laufen. Abschliessend sollte noch das Logging in die CSV-Dateien unterbunden werden. Dieses gelingt nur, wenn man die entsprechenden Module `cdr_csv.so` und `cdr_custom.so` in der Datei `/etc/asterisk/modules.conf` sperrt:

```
[modules]
noload => cdr_csv.so
noload => cdr_custom.so
```

3.3.5. MeetMe

Mit der Anwendung *MeetMe* werden in Asterisk virtuelle Konferenzräume realisiert. In diese können sich Teilnehmer einwählen und wie in einer realen Konferenz miteinander kommunizieren. Die Einrichtung ist relativ einfach. MeetMe-Räume werden anhand einer Zahl identifiziert, i.d.R. die Telefonnummer oder Durchwahl, unter der sie zu erreichen sind. In der Konfigurationsdatei `/etc/asterisk/meetme.conf` wird im Abschnitt `[rooms]` der Konferenzraum mit der Nummer `2342` wie folgt angelegt:

```
[rooms]
conf => 2342
```

Ein solcher Raum kann zusätzlich mit einem Passwort geschützt werden:

```
conf => 2342,4711
```

Ein Anrufer müsste nun auf seinem Telefon die Ziffernfolge `4711` eingeben, um der Konferenz beitreten zu können. Im Dialplan steht die Funktion `MeetMe()` zur Verfügung, um einer Durchwahl einen Konferenzraum zuordnen zu können.

```
exten => 2342,1,Answer()
exten => 2342,n,MeetMe(2342)
exten => 2342,n,Hangup()
```

Der Funktion `MeetMe()` können unter anderem folgende, zusätzliche Flags übergeben werden:

m Monitormodus, bei dem der Anrufer nur zuhören, nicht aber selbst reden kann

t Talkmodus, bei dem der Anrufer nur reden, aber nicht zuhören kann

i der Anrufer muss seinen Namen nennen, mit dem anschliessend den anderen Teilnehmern verkündet wird, dass ein weiterer Benutzer die Konferenz betreten bzw. verlassen hat

r Recordingmodus, der ein Aufzeichnen der Konferenz als WAV-Datei im Verzeichnis `/var/lib/asterisk/sounds` ermöglicht

Um eine Konferenz aufzuzeichnen, bei der sich die Teilnehmer mit Namen identifizieren müssen, wäre also folgender Eintrag im Dialplan nötig:

```
exten => 2342,n,MeetMe(2342,ir)
```

3.3.6. Voicemail

Asterisk bietet die Möglichkeit, für jeden Benutzer einen passwortgeschützten Anrufbeantworter einzurichten. Optional können neue Nachrichten per Email verschickt werden. Voicemailboxen werden in der Datei `/etc/asterisk/voicemail.conf` konfiguriert. Dort legt man im Abschnitt `[general]` das Format fest, in dem Nachrichten aufgezeichnet und ggf. verschickt werden:

```
[general]
format=wav
```

Audiodateien im WAV-Format benötigen zwar mehr Speicherplatz als z.B. GSM-codierte Daten, sind aber auf den allermeisten Betriebssystemen problemlos abspielbar. Die Definition der einzelnen Mailboxen gestaltet sich ebenfalls relativ einfach:

```
[default]
1234 => 0815,Vorname Nachname,user@domain.tld
```

Die Voicemailbox `1234` ist hier mit dem Passwort `0815` geschützt. Der optionale Name lässt lediglich die Anrede in den Emails, die bei neuen Nachrichten an `user@domain.tld` versendet werden, persönlicher aussehen. Eine einfache Mailbox ohne Emailversand lässt sich mit der folgenden spartanischen Konfigurationszeile definieren:

```
6789 => 6969
```

Die Voicemailbox `6789` wäre somit per Passwort `6969` gesichert. Im Dialplan stehen die Funktionen `Voicemail()` und `VoiceMailMain()` zum Aufsprechen bzw. zum Abhören eines Anrufbeantworters zur Verfügung. Meist soll eine Voicemailbox besprochen werden, wenn ein angerufener Teilnehmer nach einer gewissen Zeit nicht abnimmt:

```
exten => 1234,1,Dial(SIP/teilnehmer1, 20)
exten => 1234,n,Voicemail(1234)
exten => 1234,n,Hangup()
```

Im Beispiel ist der Kanal `SIP/teilnehmer1` unter der Durchwahl `1234` erreichbar. Wird das zugehörige Endgerät nicht nach 20 Sekunden abgenommen, kann der Anrufer auf die Mailbox `1234` sprechen. Das Abhören der Nachrichten geschieht mittels der Funktion `VoiceMailMain()`:

```
exten => 30001234,1,Answer()
exten => 30001234,n,VoiceMailMain(1234)
exten => 30001234,n,Hangup()
```

Wählt man die Nummer `30001234` und gibt anschliessend über die Tastatur des Telefons das korrekte Passwort ein (hier: `0815`), so gelangt man in das Menü der Mailbox `1234`, welches dem Benutzer alle Optionen erläutert. So können mit der Taste `1` neue Nachrichten abgehört und per Druck auf Taste `7` gelöscht werden. Damit man im Dialplan nicht jede Voicemailbox verankern muss, bietet sich die Verwendung der Variablen `EXTEN` an:

```
exten => _3000XXXX,1,Answer()
exten => _3000XXXX,n,VoiceMailMain(${EXTEN:4})
exten => _3000XXXX,n,Hangup()
```

Somit wird der Funktion `VoiceMailMain()` jeweils die gewählte Durchwahl abzüglich der ersten 4 Ziffern übergeben. Bei Auswahl einer nicht existenten Mailbox wird man freundlich auf den Fehler hingewiesen. Ferner sollte man noch einen generischen Eintrag für alle Mailboxen vorsehen:

```
exten => 3000,1,Answer()
exten => 3000,n,VoiceMailMain()
exten => 3000,n,Hangup()
```

Wählt man die Nummer `3000`, so wird man vor der Passworteingabe aufgefordert, die Nummer der gewünschten Mailbox einzugeben.

3.3.7. Queues

Queues stellen Anruferwarteschlangen dar, wie man sie z.B. von Callcentern kennt. Hierbei wählt ein Anrufer eine Gruppenrufnummer und wird dann je nach Queuealgorithmus z.B. mit einem beliebigen Mitglied der Gruppe oder demjenigen, der zuerst abhebt, verbunden. Analog zu anderen Diensten werden Queues in der Datei `/etc/asterisk/queues.conf` eingerichtet:

```
[gruppe1]
member => SIP/teilnehmer1
member => SIP/teilnehmer2
```

Im Beispiel wird die Queue `gruppe1` samt ihren beiden Mitgliedern `SIP/teilnehmer1` und `SIP/teilnehmer2` konfiguriert. Diese SIP-Kanäle müssen natürlich ebenfalls in `/etc/asterisk/sip.conf` eingerichtet sein. Um einen Anruf auf eine Queue weiterleiten zu können, existiert im Dialplan die Funktion `Queue()`, die im folgenden Beispiel verwendet wird, um die Durchwahl `4000` der `gruppe1` zuzuweisen:

```
exten => 4000,1,Answer()
exten => 4000,n,Queue(gruppe1)
exten => 4000,n,Hangup()
```

Per default klingeln bei Anwahl einer Queue alle zugeordneten Endgeräte, hier also `SIP/teilnehmer1` und `SIP/teilnehmer2`. Neben diesem *ringall* genannten Algorithmus werden von Asterisk noch die folgenden unterstützt:

roundrobin leitet jeden Anrufer reihum auf den nächsten Kanal in der Queueliste weiter

leastrecent leitet einen Anrufer auf das Endgerät weiter, welches seit der längsten Zeitspanne nicht mehr von dieser Queue angesprochen wurde

fewestcalls leitet einen Anrufer auf das Endgerät weiter, welches bisher die wenigsten Anrufen angenommen hat

random leitet ein ankommendes Gespräch auf einen zufällig ausgewählten Kanal weiter

rrmemory ein verbesserter Roundrobin-Algorithmus

Einen alternativen Algorithmus weist man einer Queue mit dem Schlüsselwort `strategy` zu:

```
[gruppe1]
strategy = fewestcalls
member => SIP/teilnehmer1
member => SIP/teilnehmer2
```

3.3.8. Asterisk Manager Interface

Das *Asterisk Manager Interface* (AMI) ist ein Serverdienst, mit dem Telefongespräche zwischen Teilnehmern aufgebaut, beendet und beobachtet werden können. Es können ferner Stati von Queues, Voicemailboxen, SIP- und IAX-Clients abgefragt werden. Die gesamte Kommunikation zwischen einem AMI-Client und Asterisk verwendet ein triviales Klartextprotokoll. Per default werden serverseitig Verbindungsanfragen auf TCP-Port 5038 erwartet. Die Konfiguration samt Authentifizierungsdetails werden in der Datei `/etc/asterisk/manager.conf` vorgenommen. Hier muss zunächst im Abschnitt `[general]` der AMI-Server eingeschaltet werden:

```
[general]
enabled=yes
```

Andere Abschnittsbezeichner als `general` werden als Benutzernamen aufgefasst:

```
[amiuser]
secret=sehr_geheim
```

Hier wurde der User `amiuser` mit dem Passwort `sehr_geheim` angelegt. Wie in der `iax.conf` kann über die Parameter `deny` bzw. `permit` festgelegt werden, aus welchen IP-Netzen sich dieser Benutzer verbinden darf:

```
deny=0.0.0.0/0.0.0.0
permit=127.0.0.1/32
```

Im Beispiel wird zunächst der komplette IP-Adressraum gesperrt, um dann lediglich Verbindungen über das Loopbackinterface zuzulassen. Abschliessend werden dem Benutzer noch sämtliche Rechte zugesprochen:

```
read = system,call,log,verbose,command,agent,user
write = system,call,log,verbose,command,agent,user
```

Das Asterisk Manager Interface wird in der Webanwendung *AD2Ast* verwendet, welche im Rahmen dieser Studienarbeit erstellt wurde (s. Kapitel 6). Zudem greifen die in Kapitel 4 vorgestellten Managementtools auch auf diese Schnittstelle zurück.

3.3.9. Sprachpakete

Asterisk wird mit einem englischen Sprachset geliefert, welches u.a. die gesprochenen Menüs für Voicemailboxen oder Anruferwarteschlangen beinhaltet. Obwohl diese in sehr deutlichem und leicht zu verstehendem Englisch aufgenommen wurden, bietet sich der Einsatz lokalisierter Samples an. So hat die Stadt Pforzheim ein deutsches Sprachpaket unter der GPL veröffentlich. Das Archiv wird mit folgendem Befehlszweizeiler entpackt und installiert:

```
tar -xzf ast_prompts_de_v2_0.tar.gz
cp -av ast_prompts_de_v2_0/var/lib/asterisk/sounds/* \
  /var/lib/asterisk/sounds/
```

Ferner muss in den globalen Sektionen der Konfigurationsdateien (wie z.B. `sip.conf` oder `iax.conf`) der Parameter `language=de` hinzugefügt werden.

3.3.10. Festival

Das Programmpaket *Festival* stellt eine sogenannte *text to speech*-Anwendung dar, die Texte in Sprache umwandelt. Auf die vielfältigen Einstellungen und Optimierungsmöglichkeiten wird hier nicht näher eingegangen, so dass man mit den Standardeinstellungen ausreichende Ergebnisse erzielt, wenn kurze, englischsprachige Sätze synthetisiert werden. Festival kann entweder über das Kommandozeilenprogramm `text2wave` oder per Netzwerkverbindung angesprochen werden, sofern es als Serverdienst eingerichtet ist. Asterisk kann mittels der Dialplanfunktion `Festival()` einen solchen Server ansprechen. Hierfür muss jedoch die Konfigurationsdatei `/etc/festival.scm` um folgende Zeilen ergänzt werden (s.a. `contrib/README.festival` im Sourcecodeverzeichnis von Asterisk):

```
(define (tts_textasterisk string mode)
"(tts_textasterisk STRING MODE)
Apply tts to STRING. This function is specifically designed for
use in server mode so a single function call may synthesize the string.
This function name may be added to the server safe functions."
(let ((wholeutt (utt.synth (eval (list 'Utterance 'Text string)))))
(utt.wave.resample wholeutt 8000)
(utt.wave.rescale wholeutt 5)
(utt.send.wave.client wholeutt)))
```

Nun kann man z.B. alle ungültigen Durchwahlnummern mit einen entsprechenden Hinweis versehen, indem man am Ende des Dialplans folgendes anfügt:

```
exten => _X.,1,Answer()
exten => _X.,n,Festival(This is an invalid number.)
exten => _X.,n,Hangup()
```

Mit diesem *catch all*-Eintrag bekäme ein Anrufer den Satz „This is an invalid number.“ zu hören, wenn er eine Telefonnummer anwählt, die nicht zuvor explizit verarbeitet wird.

3.3.11. Weitere Dienste

Die im folgenden dargestellten Dienste wurden im Zuge dieser Studienarbeit nicht tiefer behandelt. Sie werden hier dennoch knapp erläutert, da sie zum Standardumfang von Asterisk gehören und spannende Themengebiete für zukünftige Arbeiten darstellen.

ENUM Mit dem *tElephone NUmber Mapping* wird im globalen *Domain Name System* (DNS) hinterlegt, wie eine Telefonnummer zu erreichen ist. Im Dialplan steht zur Abfrage die Funktion `ENUMLookup()` zur Verfügung, die bei einer erfolgreichen Suche im DNS die Variable `$ENUM` mit einer entsprechenden Kanaldefinition belegt. Diese wiederum kann dann per Aufruf von `Dial()` angewählt werden.

DUNDi Das *Distributed Universal Number Directory* ist funktional mit *ENUM* verwandt. Es handelt sich hierbei jedoch um ein Netzwerkprotokoll, welches von Mark Spencer zum Auffinden von Telefonteilnehmern über das IAX-Protokoll entwickelt wurde.

AGI Das *Asterisk Gateway Interface* ermöglicht es, eigene Skripte in Perl, PHP oder einer nahezu beliebigen Programmiersprache in den Dialplan einzubinden. Diese kommunizieren über Standardeingabe und -ausgabe sowie per Standardfehlerstream mit Asterisk. Somit ist es möglich, im Dialplan dynamische Funktionen oder gar Anbindungen an Datenbanken zu realisieren.

MoH Per *Music on Hold* können Anrufer (z.B. in einer *Queue*) über die Wartezeit hinweggetröstet werden. Asterisk bietet die Möglichkeit, eigene Musikstücke zu verwenden.

4. Managementtools

4.1. gastman

Der *GTK Asterisk Manager* (gastman) ist ein von Mark Spencer geschriebenes Programm, mit dem sich Zustände und Ereignisse von Kanälen beobachten und Telefonate steuern lassen. Als Grafikbibliothek wird das *GIMP Toolkit* (GTK) verwendet, so dass primär Linux und Unix-Derivate als Plattform in Frage kommen. Gastman greift auf das *Asterisk Management Interface* zurück, über das zwar Stati und Events, aber keine Auflistung der Endgeräte abgefragt werden können. Daher müssen alle Telefone, Queues, Voicemailboxen usw. zunächst manuell eingerichtet werden. In Abbildung 6 sind dies die Symbole in der mittleren Spalte. Über die den jeweiligen Symbolen zugeordneten Kontextmenüs können Gespräche zwischen zwei Teilnehmern initiiert oder Telefonate beendet werden. Zusätzlich verfügt gastman über einen integrierten Asteriskclient.

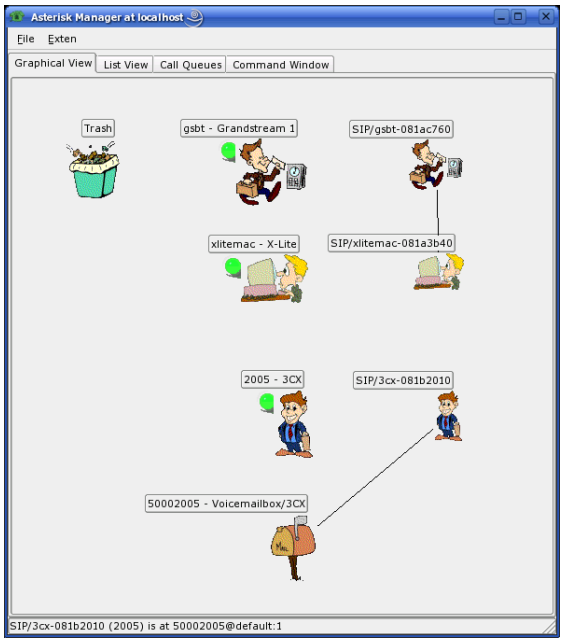

Abbildung 6: Gastman

4.2. Flash Operator Panel

Das *Flash Operator Panel* ist funktional ähnlich zu gastman. Es erlaubt ebenfalls, Endgeräte und Gespräche zu überwachen und zu steuern. Jedoch handelt es sich um eine webbasierte Client-/Serveranwendung. Im Webbrowser wird zu diesem Zweck eine Flash-Datei geladen, die sich mit einem in Perl geschriebenen Server auf dem Asteriskrechner verbindet. Dieser Server wiederum kontaktiert das Asterisk Manager Interface. Das Flash Operator Panel erlaubt eine flexible Konfiguration: Buttons können einzelnen Endgeräten zugeordnet und mit einem Link zu einer weiterführenden Webseite versehen werden. Dank der Flashoberfläche ist prinzipiell möglich, zwei Endgeräte per Drag & Drop miteinander zu verbinden. Allerdings stürzte der auf dem Asteriskrechner laufende Server des Flash Operator Panels reproduzierbar beim Versuch ab, ein Gespräch zwischen einem Hard- und einem Softphone zu initiieren. Ebenso war es nicht möglich, ein Telefonat zu einem MeetMe-Raum herzustellen. Daher wurden weitere Tests mit dem

Flash Operator Panel nicht unternommen und statt dessen empfiehlt der Autor dieser Studienarbeit, spätere Versionen der Software zu evaluieren.

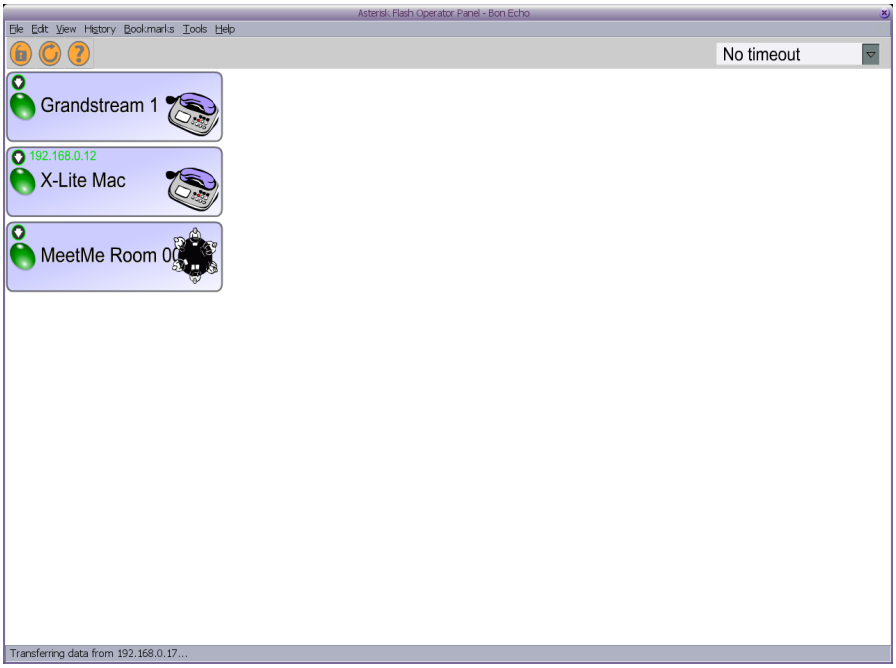

Abbildung 7: Flash Operator Panel

5. Endgeräte

5.1. Softphones

Abbildung 8: X-Lite unter Microsoft Windows

Softphones sind clientseitige Programme, die unter Verwendung einer im PC installierten Soundkarte samt Headsets ein Telefon nachbilden. Die meisten Softphones verwenden SIP/RTP, um sich mit einem Asteriskserver zu verbinden. Nur wenige Clients unterstützen IAX. Einen Zwitter, der beide Welten kombiniert, sucht man vergeblich. Ebenso fehlen zumindest bei kostenlos erhältlichen Programmen wünschenswerte Funktionen wie die Anbindung an LDAP- oder andere Verzeichnisdienste, die als Telefonbuch fungieren können. Generell hinterließen die getesteten Softphones beim Autor den Eindruck, es handle sich um Chat- oder Instant-Messaging-Clients mit Telefonfunktion. Ein sehr schlechtes Bild lieferten die als Opensource veröffentlichten SIP-Phones *kphone* und *linphone*. Ersteres reagierte nicht mehr, sobald ein Gespräch aufgebaut war, während letzteres nicht einmal fehlerfrei zu compilieren war. Ferner preisen einige Anbieter ihre Software als frei nutzbaren, standardkonformen SIP-Client an, setzen jedoch ein wenn auch kostenloses Kundenkonto voraus und sind nur per Tricks wie lokale DNS-Änderungen oder Eingriff in die Windows-Registry zur Zusammenarbeit mit einem eigenen Asteriskserver zu überreden. Daher blieben Programme wie *WengoPhone* oder *Nero Sipps* ebenfalls aussen vor. Sehr erfreulich verliefen Tests mit IAX-Clients. Hier ermöglichten Opensource- und Free-/Sharewareprogramme unter Linux, Windows und selbst unter Apples Mac OS X (auf dem Notebook des Autors) problemlos Telefongespräche unter Einsatz des Asteriskservers.

5.1.1. X-Lite

Der SIP-Client *X-Lite* wird von der Firma *CounterPath* für die Betriebssysteme Windows (Abbildung 8), Linux und Mac OS X (Abbildung 9) kostenlos angeboten. Es handelt sich dabei um die Freewareversion von *eyeBeam*, einem integrierten SIP- & Video-over-IP-Client des gleichen Herstellers. Als Besonderheit ermöglicht X-Lite das clientseitige Aufzeichnen von Telefonaten sowie die Konfiguration mehrerer Asterisk- bzw. SIP-Accounts, um so schneller zwischen einzelnen Anbietern wechseln zu können.

Abbildung 9: X-Lite unter Mac OS X

5.1.2. Snom

Abbildung 10: Snom Softphone

Der Firma *Snom* ist es gelungen, ihr durchdachtes Hardphone *snom 360* als reine Softwarevariante für Windows (Abbildung 10) zu veröffentlichen, die für private Zwecke frei nutzbar ist. Zwar wurde die Bedienung des Hardphones 1:1 auf die des Softphones übertragen, dennoch gelingt die Konfiguration problemlos dank des eingebauten Webservers (s. Abbildung 11). Die Software bietet vielfältige Möglichkeiten:

- Unterstützung von bis 12 einzelnen SIP-Accounts
- 12 frei belegbare Sondertasten
- 32 frei programmierbare Kurzwahlnummern
- Aufrufen einer URL bei Ereignissen wie *Hörer abnehmen* oder *Anruf beendet*, so dass z.B. auf einer firmeninternen Webseite angezeigt werden kann, ob ein Benutzer gerade telefoniert oder nicht gestört werden will
- Remotelogging auf einen Syslogserver
- Management per SNMP

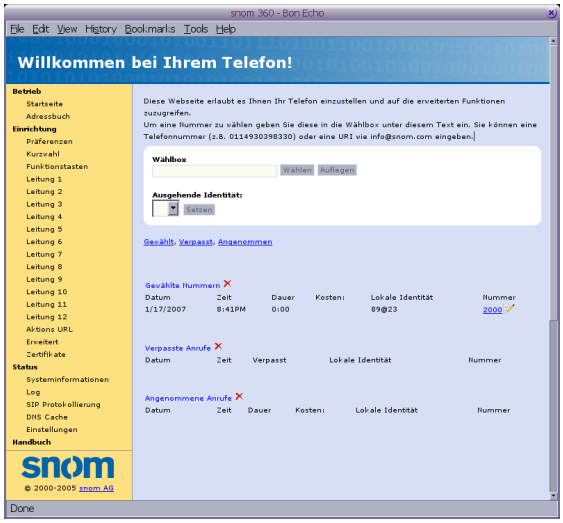

Abbildung 11: Snom Softphone – Weboberfläche

- Import des Adressbuches aus einer CSV-Datei

5.1.3. 3CX Phone

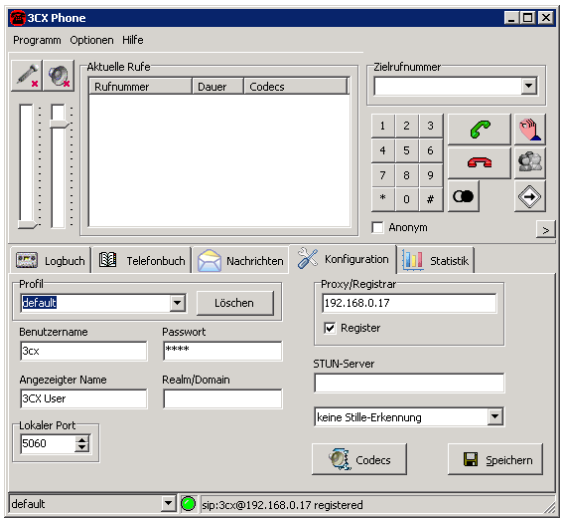

Abbildung 12: 3CX Phone

Das Kernprodukt der Firma *3CX* ist ein Telefonieserver für Windows. Das spartanische, aber voll funktionale SIP-Phone *3CX Phone* (Abbildung 12) ist separat und frei erhältlich. Es unterstützt ebenfalls mehrere SIP-Accounts und pflegt ein lokales Adressbuch. Als Besonderheit kann es per SIP Textnachrichten an andere Endgeräte versenden, gleichwohl kaum eine andere Software dies verarbeiten kann.

5.1.4. JackenIAX

Der IAX-Client *JackenIAX* (s. Abbildung 13) ist (leider) ein reinrassiges Mac OS X-Programm. Es ist ähnlich einfach wie der SIP-Client 3CX Phone, ermöglicht aber die

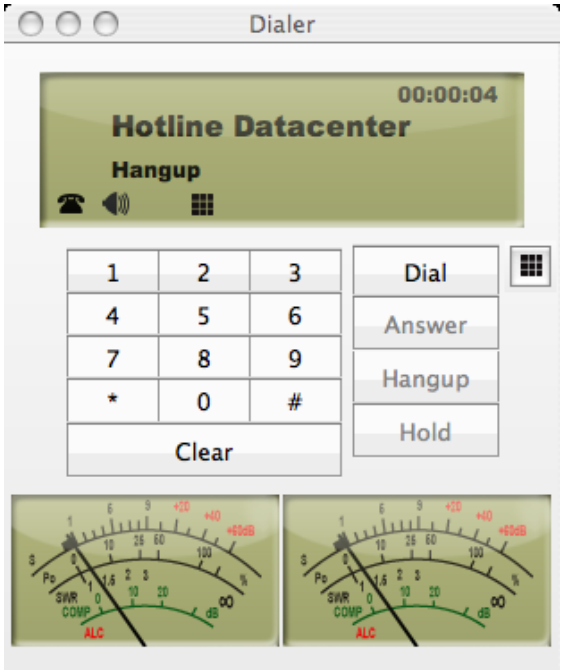

Abbildung 13: JackenIAX

Einbindung des herkömmlichen Adressbuches, wie es auf jedem System von Apple installiert ist. Statt eine eigene Adressliste für JackenIAX zu pflegen, können hier einfach per Doppelklick aus dem schon bestehenden Adressbuch Kontakte angerufen werden.

5.1.5. Idefisk

Abbildung 14: Idefisk unter Mac OS X

Idefisk ist ein frei für Windows, Linux und Mac OS X (Abbildung 14) erhältlicher IAX-Client. Als Besonderheit unterstützt er lediglich mehrere IAX-Accounts.

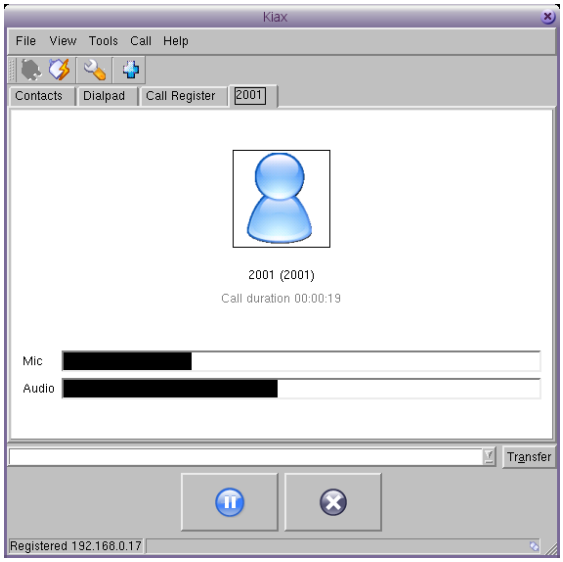

Abbildung 15: Kiax unter Linux

5.1.6. Kiax

Das funktionale Opensource-Pendant zu *Idefisk* findet sich in *Kiax* (s. Abbildung 15), welches ebenfalls mehrere IAX-Accounts unterstützt und lediglich ein separat gepflegtes Adressbuch führt.

5.2. Hardphones

5.2.1. Grandstream

Abbildung 16: Grandstream GXP-2000

Die IP-Telefone der Firma *Grandstream Networks* sind preislich attraktive Hardphones. Aktuell gibt es drei Produktlinien: die einfachen Modelle der *BudgeTone*-Serie (50 -

Abbildung 17: Grandstream BudgeTone 200

70 €), die Enterprise-Telefone der Reihe *GXP* (ca. 100 €) sowie die videofähigen Geräte der *GXV*-Reihe (ca. 250 €, Stand jeweils Januar 2007). Für den weiteren Einsatz im Labor wurden zwei Telefone des Typs *GXP 2000* (Abbildung 16) angeschafft. Ferner standen aus dem privaten Fundus des Autors zwei *BudgeTone 101* für Testzwecke zur Verfügung. All diese Geräte sprechen ausschliesslich SIP und unterstützen die gängigen Audiocodecs μ-Law, A-Law sowie die der GSM-Familie. Die Erstkonfiguration erfolgt über menügeführte Dialoge an den Telefonen. Das BudgeTone verfügt hierzu über ein LC-Display, das GXP 2000 über eine besser lesbare Dot-Matrix-Anzeige. Neben Zuweisung einer festen IP-Adresse samt Netzmaske und Defaultgateway können auch Adressen per DHCP geholt werden. Die weitere Konfiguration kann bequem über den eingebauten Webserver erfolgen (Abbildung 18). Zunächst sollten natürlich die obligatorischen

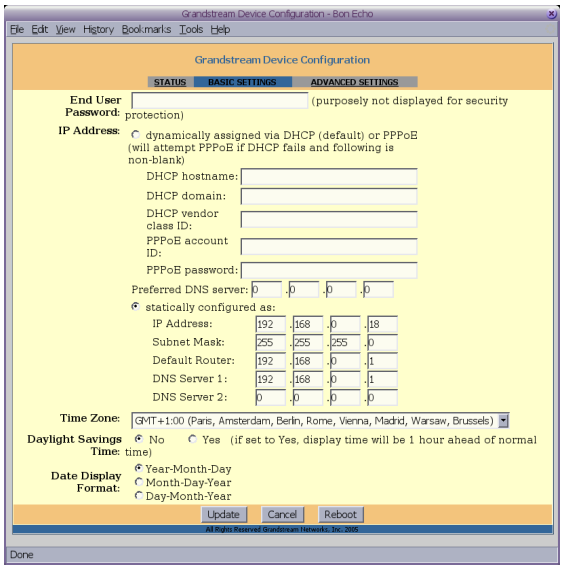

Abbildung 18: Weboberfläche zur Konfiguration eines Grandstream BudgeTone 100

SIP-Einstellungen vorgenommen werden und die Adresse des Asteriskservers samt Username und Passwort konfiguriert werden. Das GXP 2000 unterstützt hierbei bis zu 4 unabhängige SIP-Accounts. Zudem können sich die Grandstreamgeräte als PPPoE-Client selbstständig bei einem (DSL-)Provider einwählen. Die Geräte aus der GXP-Serie können sogar als NAT-Router fungieren. Datum und Uhrzeit können per *Network Time Protocol*

(NTP) von entsprechenden Server synchronisiert werden. Ebenso ist ein automatisches Firmwareupdate per HTTP oder TFTP möglich. Ferner können Gespräche weitergeleitet bzw. übergeben und Telefonkonferenzen (ohne Einsatz eines MeetMe-Raumes) eingeleitet werden. Dank des eingebauten Lautsprechers samt Mikrofon kann der Anwender frei und ohne den Hörer in die Hand zu nehmen telefonieren. Das BudgeTone verfügt über keine Möglichkeit, ein Telefonbuch zu pflegen. Beim GXP 2000 kann eine solche Liste nicht nur lokal angelegt, sondern auch als XML-Datei von einem Webserver importiert werden.

5.2.2. Snom

Abbildung 19: snom 360

Die deutsche Firma *snom* bietet neben ihrem für private Verwendung frei zu nutzendem Softphone (s. Kapitel 5.1.2) die drei Hardphones *snom 300*, *snom 320* und das *snom 360* an. Wie auch beim Softphone werden bis zu 12 SIP-Accounts unterstützt. Zudem können Telefonbücher per LDAP importiert und die Geräte per SNMP abgefragt werden.

6. AD2Ast

Um die Fähigkeiten des Asterisk Manager Interfaces zu demonstrieren, wurde die Webanwendung *AD2Ast* entwickelt. Sie synchronisiert Name, Emailadresse und Telefonnummer von Benutzerdaten aus einem Active Directory in eine MySQL-Datenbank. Eine Weboberfläche, sprich ein CGI-Skript greift auf diese Datenbank zu und ermöglicht dem Anwender, komfortabel über einen Browser ein Gespräch zwischen seinem Telefon und dem ausgewählten Kontakt herzustellen. Zuvor muss sich der Anwender jedoch an der Weboberfläche mit Username und Passwort anmelden. Der Username ist identisch mit der Nummer der Voicemailbox des jeweiligen Anwenders. Als Passwort wird daher auch nur dasjenige der entsprechenden Mailbox akzeptiert. Weitere Voraussetzung ist, dass die Nummer der Mailbox gleich der Durchwahl des Telefons ist, welches dem Anwender zugeordnet ist, da das CGI-Skript den Dialplan parsen muss, um den Kanal jenes Telefons zu ermitteln. Dieses Vorgehen ist durch das Asterisk Manager Interface bedingt. Es kann zwar das Zieltelefon bzw. die Zielnummer als Durchwahl verarbeiten, das Quelltelefon muss jedoch in der üblichen Schreibweise wie z.B. `SIP/teilnehmer1` angegeben werden. Über die Weboberfläche kann jeder Benutzer zusätzliche, nicht im Active Directory hinterlegte Telefonnummern angeben, unter denen er zu erreichen ist. Diese werden in einer separaten Tabelle in der MySQL-Datenbank gespeichert. Ferner werden über ein weiteres CGI-Skript alle Daten per XML ausgegeben. Dieses dient zum Import in das Telefonbuch der Grandstream Hardphones. Es ergibt sich der in Abbildung 20 gezeigte Aufbau. Als Programmier- bzw. Skriptsprache wurde Perl verwendet, da hierfür entsprechende DNS- und LDAP-Module zur Abfrage eines Active Directorys verfügbar sind. Zudem ist das Verarbeiten von Zeichenketten in Perl relativ einfach. Der Verlust in der Ausführungsgeschwindigkeit gegenüber nativ compilierenden Sprachen wie C wurde in Kauf genommen. Andere Skriptsprachen wie z.B. PHP verfügen nicht über die geforderten DNS- bzw. LDAP-Funktionen. Der Einsatz von in Java geschriebenen Servlets hätte an dieser Stelle einen zu hohen Aufwand bedeutet, da neben dem Webserver noch ein spezieller Servletcontainer wie Tomcat erforderlich gewesen wäre. Folgende Perl-Module werden von AD2Ast verwendet und müssen ggf. nachträglich installiert werden:

DBI bietet abstrakten Datenbankzugriff

DBD-mysql Datenbanktreiber für MySQL

Digest::MD5 stellt Routinen zur Berechnung von MD5-Hashes bereit

IO::Socket ermöglicht den Zugriff auf das Socketinterface zur Netzwerkprogrammierung unter Unix

MIME::Base64 stellt Funktionen zur Stringkonvertierung ins Base64-Format bereit

Net::DNS ermöglicht spezielle Anfragen an Nameserver

Net::LDAP stellt Funktionen zum Zugriff auf LDAP-Server bereit

Die auf dem Webserver laufenden Skripte `ad2ast_dial.pl`, `ad2ast_sync.pl` und `ad2ast_xml.pl` binden jeweils die Datei `ad2ast_subs.pl` ein. Diese enthält die Parameter zur Datenbankverbindung, Informationen über das Active Directory, Daten zur Konnektierung des Asterisk Manager Interfaces sowie Routinen zum Anbinden und Abfragen der Datenbank.

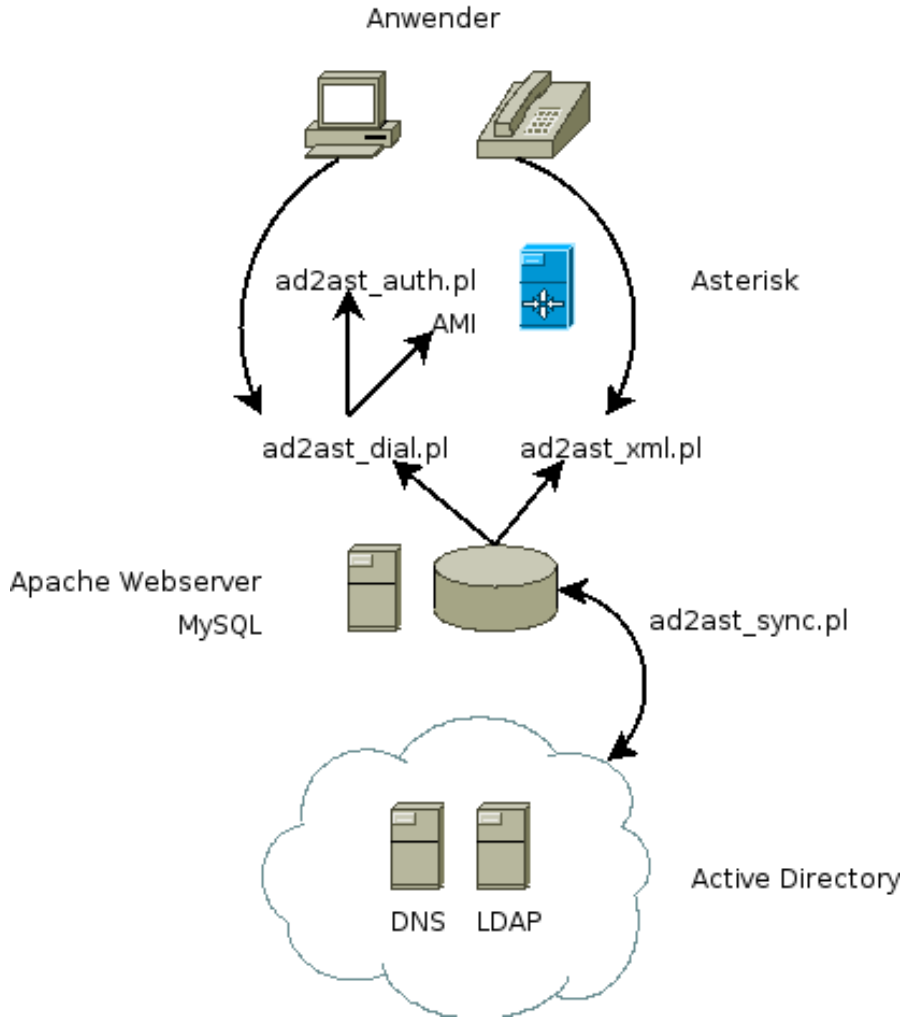

Abbildung 20: Zusammenspiel der Komponenten in der AD2Ast-Umgebung

6.1. ad2ast_sync.pl

Das Skript `ad2ast_sync.pl` synchronisiert Telefonnumer, Emailadresse und Name von Benutzern in einem Active Directory in eine MySQL-Datenbank. Die dafür vorgesehenen Benutzerkonten müssen sich in einer zusätzlichen Gruppe befinden, die durch die Variable `$search_dn` angegeben wird. Da ein Active Directory unter anderem aus mindestens einer DNS-Zone besteht, können die zuständigen LDAP-Server entweder direkt angegeben werden (Variable `@ldap_servers`) oder aber über sogenannte *SRV-Records* aus dem DNS bestimmt werden. Hierzu werden die in der Variablen `@nameservers` vorgegebenen Nameserver nach DNS-Einträgen der Form `_ldap._tcp.`*`Domäne`* gefragt. Die Domäne wird hierbei über die Variable `$domain` bestimmt. Die LDAP-Server werden mit Hilfe den in `$bind_dn` und `$bind_pw` angegebenen Usernamen und Passwort konnektiert. Anschliessend wird nach Objekten gesucht, die über das Attribut `ipPhone` verfügen und sich zudem in der o.g. Gruppe befinden. Als Rückgabe dieser LDAP-Suche werden die Attribute `displayName`, `mail` und `ipPhone` ausgewählt. Die so gelieferten Daten werden mit denen in derjenigen SQL-Tabelle verglichen, die durch die Variable `$ad_table` definiert ist. Hier werden in einer Transaktion zunächst alle Bestandsdaten gelöscht, die nicht mehr im Active Directoy vorhanden sind. Im letzten Arbeitsschritt werden noch

nicht vorhandene Benutzerinformationen in der Datenbank gespeichert.
Das Skript `ad2ast_sync.pl` ist als Cronjob gedacht und sollte einmal täglich ausgeführt werden, z.B. um 1 Uhr nachts per folgendem Eintrag in der Crontab:

```
0 1 * * * cd /srv/www/ad2ast/cgi-bin && ./ad2ast_sync.pl
```

6.2. ad2ast_dial.pl

Das CGI-Skript `ad2ast_dial.pl` ist die Schnittstelle zum Benutzer und somit die Kernkomponente. Die Authentifizierung der Benutzer erfolgt über eine eigens erstellte Client-/Serverschnittstelle, welche die in der `voicemail.conf` hinterlegten Mailboxnummern als Usernamen und das jeweilige Mailboxkennwort als Passwort verwendet. Nach dem erfolgreichen Login wird dem Anwender zunächst eine Willkommensseite präsentiert. Von hier hat er die folgenden Möglichkeiten:

- einen in der Datenbank hinterlegten Kontakt anzurufen
- eine weitere Telefonnummer, die nicht im Active Directory gepflegt wird, unter der er aber zu erreichen ist, hinzuzufügen oder zu löschen
- Teilnehmer anhand von Telefonnummer, Emailadresse oder Name zu suchen
- sich abzumelden.

Als CGI-Skript bekommt `ad2ast_dial.pl` den Namen der Methode, mit der es aufgerufen wurde, in der Umgebungsvariablen `REQUEST_METHOD` übergeben. Unterstützt werden die Methoden `POST`, bei der alle dem Skript übergebenen Daten von der Standardeingabe gelesen werden müssen, und `GET`, bei dem diese Daten als Teil der aufgerufenen URL in der Umgebungsvariablen `QUERY_STRING` gespeichert sind. Die per POST oder GET gelieferten Werte liegen in der Form `key1=value1&key2=value2&...&keyN=valueN` vor. Sonder- und Metazeichen wie z.B. das Gleichheitszeichen oder Umlaute werden als hexadezimale Zahl ihres ASCII-Wertes mit vorangestelltem Prozentzeichen dargestellt. So würde der Name *Bärbel* als `name=B%C3%A4rbel` übergeben werden. Die Aufteilung der Schlüssel-/Wertepaare übernimmt die Funktion `http_vars()`. Sie teilt zunächst die übergebene Zeichenkette bei jedem Vorkommen eines kaufmännischen Und-Zeichens auf. Die so gewonnenen Teilstrings stellen in Perl ein *Array* dar, dessen Elemente mit dem Operator `foreach` durchwandert werden können. Jedes dieser Elemente wird nun an Gleichheitszeichen geteilt, so dass jeder n-te Schlüssel und jeder n-te Wert in den Variablen `$key` und `$value` vorliegen. Nach dem Ersetzen von Meta- und Sonderzeichen mittels *regulärer Ausdrücke* wird der Schlüssel als Index und der Wert als Datenfeld in einem *Hash*, einem assoziierten Array, verwendet. Somit liegen die dem CGI-Skript übergebenen Werte nun in der Perl-Variablen `%http_post` vor. Die vom Anwender gewünschte Aktion wird generell im Datenfeld `$http_post{action}` übergeben. Folgende Aktionen werden verarbeitet:

login Der Anwender hat Username und Passwort in die entsprechenden Formularfelder eingetragen und möchte sich einloggen. Diese Authentifizierungswerte erhält das Skript in den Variablen `$http_post{username}` bzw. `$http_post{password}`. Sie werden anschliessend verwendet, um sich gegen die in der `voicemail.conf` hinterlegten Daten zu authentifizieren. Stimmen Username und Passwort überein, wird dem Webbrowser ein sogenanntes *Cookie* geliefert, welches bei jedem weiteren Seitenaufruf sicherstellt, dass der Anwender sich zuvor korrekt angemeldet hat

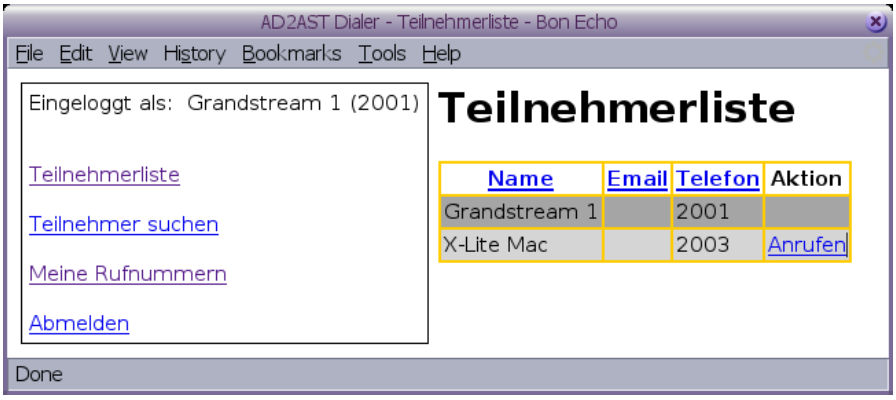

Abbildung 21: Die Teilnehmerliste in der Weboberfläche von AD2Ast

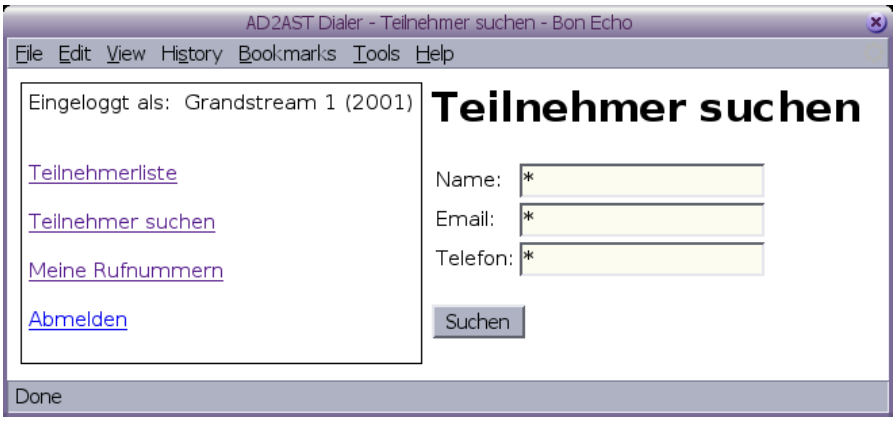

Abbildung 22: Maske zur Benutzersuche

diallist Dem Benutzer werden alle in der MySQL-Datenbank hinterlegten Teilnehmer samt Telefonnummern und Emailadresse dargestellt (s. Abbildung 21). Er hat hier die Möglichkeiten, die Liste nach Name, Telefon oder Email zu sortieren, oder per Klick auf den mit *Anrufen* bezeichneten Link die entsprechende Person anzurufen. Die eigentliche Darstellung in HTML-Code übernimmt die Funktion `diallist()`, welche auch bei der Ergebnispräsentation einer Suche (s.u.) zum Einsatz kommt

search Dem Anwender wird die Eingabemaske für die Suche im Datenbestand angezeigt (Abbildung 22). Er hat die Möglichkeit, nach Name, Email und Telefonnummer zu suchen, wobei diese 3 Suchparameter logisch UND-verknüpft sind. Als Wildcard steht das allgemein verwendete Sternchen (`*`) zur Verfügung

dosearch Mit dieser Aktion wird die eigentliche Suche im Datenbestand durchgeführt. Da kein normalisiertes Datenbankmodell realisiert wurde (dies ist beim Datenimport aus einem LDAP-Verzeichnis wie einem AD auch nur mit grossem Aufwand möglich), wird die Suche in Perl mittels regulärer Ausdrücke durchgeführt. Zuvor müssen jedoch die 3 Suchparameter in gültige reguläre Ausdrücke überführt werden. So muss z.B. der vom Anwender eingegebene Suchstring `*an*üller*` in den äquivalenten regulären Ausdruck `^.*an.*üller.*$` überführt werden. Dieser würde dann auf Namen wie *Hans Müller*, *Johann Schnüller*, *Janine Füller* usw. zutreffen. Treffen alle drei Suchparameter (Name, Telefon, Email) zu, und sei es

nur, weil der Anwender überall das Sternchen verwendet hat, so werden die Ergebnisse wiederum der Funktion `diallist()` übergeben, welche die Darstellung per HTML vornimmt

dial Diese Aktion wird aufgerufen, wenn der Anwender aus der Kontaktliste oder aus einem Suchergebnis heraus auf *Anrufen* geklickt hat. Hier wird jedoch nur ein Hinweistext („Ihr Telefon klingelt nun.“) präsentiert und der Webbrowser automatisch per Redirect dazu aufgefordert, die gleiche Seite, aber mit dem Aktionsparamter *dodial* aufzurufen

dodial Dies ist das Kernstück der gesamten Weboberfläche, da diese Aktion den Anwender mit dem von ihm gewünschten Teilnehmer verbindet. Nach einigen Plausibilitätstests, mit denen unterbunden wird, dass ein nicht in der Datenbank hinterlegter Kontakt angerufen wird und dass sich der Anwender nicht selbst anrufen will (obgleich dies nur duch Manipulation in der URL möglich ist), wird zunächst der Dialplan über das Asterisk Manager Interface befragt, wie das Telefon des aktuell angemeldeten Anwenders zu erreichen ist. Hierzu wird der AMI-Befehl `show dialplan` mit der entsprechenden Nummer abgesetzt. Geliefert werden all diejenigen Einträge der `extensions.conf`, die als Durchwahl eben die gefragte Nummer aufweisen. Diese Einträge werden nun per regulärem Ausdruck nach einem Aufruf der Dialplanfunktion `Dial()` durchsucht, die ja in diesem Falle den der gesuchten Durchwahl zugeordneten Kanal als Parameter aufweisen muss. Ist dieser gefunden, kann über das Asterisk Manager Interface das Kommando `Originate` mit Quellkanal und Zielnummer abgesetzt werden. Jedoch blockiert dieses Kommando solange, bis der Anwender den Hörer seines Telefons (welches über den Quellkanal angesprochen wird) abgenommen hat. Erst danach wird der Programmfluss fortgesetzt und per HTML der Hinweistext ausgegeben, dass nun das Telefon des gewünschten Gegenübers klingelt

extra Hier wird dem Anwender eine Übersicht seiner zusätzlich eingetragenen Telefonnummern dargestellt. Er hat wie in Abbildung 23 dargestellt die Möglichkeit, entweder eine davon zu löschen oder eine neue einzutragen

extra-add Diese Aktion wird aufgerufen, wenn der Anwender eine neue Telefonnummer hinzufügen möchte. Er erhält lediglich die Information, dass die Nummer in die Datenbank eingetragen wurde und einen Link, der ihn zur Übersicht über seine Telefonnummern zurückleitet

extra-delete Analog zu *extra-add* löscht diese Aktion eine Telefonnummer aus der Datenbank

logout Der Anwender möchte sich abmelden. Hierzu wird das Cookie mit leerem Inhalt überschrieben.

Enthält die Variable `$http_post{action}` keine der aufgeführten Werte oder wurde sie gar nicht übergeben, so wird dem Anwender die Loginmaske präsentiert, die zur Eingabe von Benutzernamen und Kennwort auffordert.
Da der Asteriskserver und der Web-/Datenbankserver nicht auf der selben Maschine laufen, musste sichergestellt werden, dass sich die Anwender gegen die in der `voicemail.conf` hinterlegten Anmeldedaten authentifizieren können. Hierzu läuft auf dem Asteriskrechner der Dienst *ad2ast_auth.pl* (s. Kapitel 6.3). Er akzeptiert Verbindungen per TCP auf Port 6666. Der Authentifizierungsvorgang basiert auf einem Challenge-/Responseverfahren, bei dem das Passwort niemals im Klartext über das Netzwerk verschickt wird.

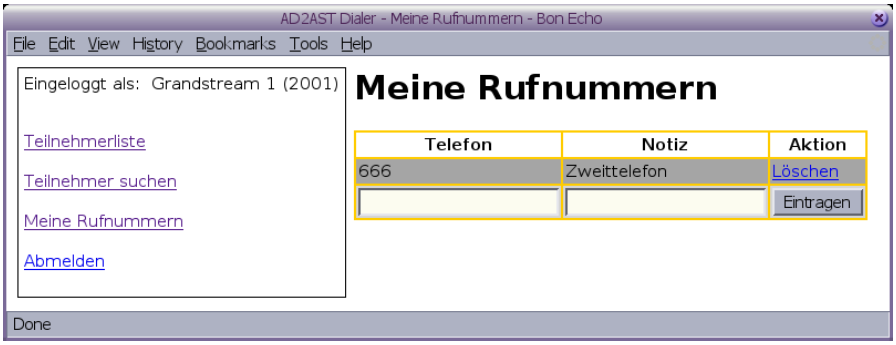

Abbildung 23: Maske zum Bearbeiten eigener Rufnummern

Die Funktion `check_credentials()` kapselt alle notwendigen Schritte. Nach Aufbau der Verbindung liefert der auf dem Asteriskserver laufende Dienst `ad2ast_auth.pl` zunächst einen zufälligen Einmalwert, eine sogenannte *Nonce*. Diese wird als Zeichenkette zusammen mit dem Usernamen und dem Passwort der Einweghashfunktion `md5()` übergeben. Dieser Hashwert sowie der Username werden dem Dienst übermittelt, der seinerseits die Hashfunktion in gleicher Art und Weise aufruft. Stimmen der von ihm ermittelte und der vom CGI-Skript gelieferte Hash überein, wird dem Login des Anwenders zugestimmt.
Die Kommunikation mit dem Asterisk Manager Interface erfolgt ebenfalls über eine TCP-Verbindung auf Port 5038. Es handelt sich dabei um eine einfache Klartextkommunikation. Jede Anweisung beginnt mit dem Schlüsselwort `Action`, gefolgt von einem Doppelpunkt samt Leerzeichen und dem eigentlichen Befehl. In den folgenden Anweisungszeilen können weitere Parameter folgen, ebenfalls in der Syntax `Key: Value`. Das Ende einer Anweisung wird durch eine Leerzeile markiert. Antworten vom AMI-Server erfolgen in der gleichen Syntax, sprich ebenfalls durch derartige Schlüssel-/Wertepaare. Lediglich bei mehrzeiligen Rückgabewerten wie z.B. einem Auszug aus dem Dialplan stellt die erste Leerzeile nicht das Ende der Antwort dar, sondern trennt den Statusteil von den eigentlichen Daten. Erst der String `--END COMMAND--` weist auf das Ende aller gelieferten Daten hin. Um sich am Asterisk Manager Interface anzumelden, setzt der Client folgende Answeisungen ab:

```
Action: Login
Username: amiuser
Secret: geheim
Events: off
```

Die Option `Events: off` weist den AMI-Server an, keine Ereignisse wie z.B. das Zustandekommen einer Verbidnung zwischen zwei Telefonen an den Client zu senden. Der Dialplan oder der für eine gewünschte Durchwahl zuständige Teil des Dialplans wird mittels der nachfolgenden Befehle abgefragt:

```
Action: Command
Command: show dialplan <Durchwahl>@default
```

Möchte man den gesamten Dialplan erhalten, so ist die Durchwahl samt Kontext wegzulassen. Der zentrale Befehl, um den sich diese Anwendung dreht, lautet `Originate`. Er verbindet einen Kanal, sprich das Telefon, welches über ihn erreicht wird, mit einer Durchwahl:

```
Action: Originate
```

```
Channel: SIP/teilnehmer1
Exten: 4711
CallerID: 2001 <teilnehmer1>
Context: default
Priority: 1
```

Somit würde das SIP-Gerät `teilnehmer1` mit der Durchwahl `4711` verbunden werden. Der angerufene Teilnehmer sähe dann als CLIP-Information auf seinem Telefon, dass eben `teilnehmer1` mit der Rufnummer `2001` anruft.

6.3. ad2ast_auth.pl

Das Skript `ad2ast_auth.pl` stellt die Authentifizierung der Benutzer im Webfrontend sicher. Es läuft unter der Obhut eines Servicedämons wie *inetd*, *xinetd* oder *tcpserver*. Diese müssen so konfiguriert sein, dass bei einer Verbindungsanfrage auf Port 6666/TCP das Skript gestartet wird. Dieses kann dann mit einem Client kommunizieren, indem es auf die Standardausgabe schreibt bzw. von der Standardeingabe liest. Somit konnte die Programmierung einer eigenen Serverkomponente in `ad2ast_auth.pl` vermieden werden. Um der geforderten, als hinreichend sicher bewerteten Authentifizierung per Challenge-/Responseverfahren mittels MD5-Verschlüsselung nachzukommen, liest das Skript zunächst aus der Devicedatei des Systemzufallgenerators `/dev/urandom` 60 Bytes ein. Da es sich dabei um binärte Werte handelt, werden sie in eine Base64-encodierte Nonce umgewandelt, die nur noch Buchstaben, Ziffern, das Pluszeichen oder den Schrägstrich enthält. Diese Zeichenkette wird an den Client geschickt, welcher daraufhin mit dem Benutzernamen und der per `md5()` verschlüsselten Zeichenkette aus Benutzernamen, Passwort und Nonce antwortet. In der Datei `/etc/asterisk/voicemail.conf` wird nun nach diesem Benutzer, sprich dieser Durchwahl bzw. Voicemailboxnummer geparst. Mit dem dort hinterlegten Passwort wird ebenfalls die gleiche MD5-Verschlüssung unter Verwendung der Nonce durchgeführt. Stimmt das Ergebnis mit dem vom Client gelieferten Wert überein, wird diesem per Ausgabe von `ok` signalisiert, dass sich der Anwender korrekt authentifiziert hat. Andernfalls und generell bei Fehlern antwortet `ad2ast_auth.pl` mit einem knappen `no`, was dem Webfrontend anzeigt, den Anwender nicht hereinzulassen.

6.4. ad2ast_xml.pl

Die IP-Telefone GXP 2000 von Grandstream können ihr Adressbuch als XML-Datei von einem Webserver laden. Das erwartete Format ist relativ einfach:

```
<?xml version="1.0"?>
<AddressBook>
  <Contact>
    <LastName>...</LastName>
    <FirstName>...</FirstName>
    <Phone>
      <phonenumber>...</phonenumber>
      <accountindex>0</accountindex>
    </Phone>
  </Contact>
  ...
</AddressBook>
```

Nach der obligatorischen XML-Deklarierung wird der Root-Container `AddressBook` erwartet. Dieser kann mehrere Elemente vom Typ `Contact` enthalten. Ein Kontakt besteht schliesslich aus einem Vor- und einem Nachnamen sowie einem weiteren Container, der die eigentliche Telefonnummer enthält. Das Skript `ad2ast_xml.pl` erzeugt diese XML-Datei auf Abruf und liefert sie direkt an den Client, sprich das Telefon aus. Dabei werden alle Telefonnummern aus der MySQL-Datenbank exportiert.

7. Integration in das Labornetz

Der Asteriskserver soll zukünftig im Laboralltag im Rahmen von Praktika zu Vorlesungen und im CCNA-Kurs eingesetzt werden. Hierzu wurde aus zwei älteren Computern eine hinreichend leistungsfähige Maschine mit 512 MB Hauptspeicher, einer 15 GB grossen Festplatte, einem DVD-ROM und einer Pentium 3 CPU mit 450 MHz geschaffen. Zusätzlich erhielt dieser Rechner die ISDN-Karte. Als Betriebssystem wurde abweichend zu den vorangegangenen Tests *openSuSE Linux 10.2* installiert, wobei im Hinblick auf den reinen Servereinsatz auf eine grafische Oberfläche wie *KDE* oder *Gnome* verzichtet wurde. Die Bedienung erfolgt daher ausschliesslich über die Textkonsole. In dieser Konfiguration sollte der Rechner 5 gleichzeitige Telefonate ohne Verzögerungen oder Paketverlust verarbeiten können (vgl. Meggelen u. a. (2005)). Die Festplatte bekam eine individuelle Partitionierung: 1024 MB zu Beginn der Festplatte als Swapspeicher, der Rest von rund 14 GB als Root-Verzeichnis mit dem Dateisystem *ext3* und der Option *data=journal*. Folgende Pakete kamen zum Einsatz:

- openSuSE-Basissystem
- Konsolenwerkzeuge
- YaST-Systemverwaltung
- Grundlegende Entwicklung
- C/C++-Entwicklung
- Linux-Kernel-Entwicklung
- Voice over IP-Server (Asterisk)
- Asterisk-PgSQL
- PostgreSQL-Server
- Perl DBD MySQL
- Festival

Der Rechner erhielt den Hostnamen `lnx-servp` in der Domain `infma-labor.local`. Als interne IP-Adresse wurde die `192.168.1.4` vergeben. Für eine Gruppe von Diplomanden sollte der direkte Zugriff von aussen möglich sein. Die öffentliche Adresse für das zweite, externe Interface wird zu einem späteren Zeitpunkt erfolgen. Da diese neuste Version von SuSE Linux ein komplettes Asterisk-Paket samt Zaptel-Treiber und PostgreSQL-Modul mitbringt, konnte darauf verzichtet werden, den Telefonieserver selbst zu übersetzen. Ebenso wurde der PostgreSQL-Server als Binärpaket von der SuSE-DVD installiert. Dennoch musste die Datei `/var/lib/pgsql/data/pg_hba.conf` wie in Kapitel 3.2.2 beschrieben angepasst werden. Um die CDR-Tabelle anlegen zu können, wurde die Datei `postgres_cdr.sql` vom vormals genutzen Entwicklungssystem auf den neuen kopiert. Als einziges Programm bedurfte lediglich mpg123 einer Installation aus dem Sourcecode. Ferner wurde das deutsche Sprachset für Asterisk manuell in das System kopiert. Die Installation des Kernelmoduls für die ISDN-Karte war im Gegensatz zu SuSE Linux 10.0 nicht ohne weiteres möglich. openSuSE verzichtet konsequent auf die Auslieferung von Programmen, die nicht komplett als Sourcecode unter einer zur GPL gleichwertigen Lizenz vorliegen. Daher musste das Kernelmodul für die AVM-Karte von der Webseite `http://opensuse.fltronic.de/SUSE10_2.htm` heruntergeladen werden und manuell per folgendem Befehl installiert werden:

```
rpm -i fcpci-kmp-default-0.1_2.6.18.2_34-0.i586.rpm
```

Damit das Kernelmodul `ztdummy` aus dem Zaptel-Paket automatisch beim Systemstart geladen wird, war die Datei `/etc/sysconfig/zaptel` mit folgender Zeile zu ergänzen:

```
ZAPTEL_MODULES="ztdummy"
```

Vorerst nicht benötigte Dienste innerhalb des Asteriskservers wurden aus Sicherheitsgründen deaktiviert. Dies geschah mit besonderem Hinblick auf die direkte, d.h. nicht durch eine Firewall geschützte Anbindung des Rechners an das Internet. Hierzu wurden die Unterstützung für DUNDi sowie für das von der Firma Cisco entwickelte Protokoll *Skinny* durch folgende Einträge in der `/etc/asterisk/modules.conf` unterbunden:

```
noload => chan_mgcp.so
noload => chan_skinny.so
noload => pbx_dundi.so
```

Da im Labornetz schon ein Webserver samt MySQL-Datenbank existiert, wurden dort alle Skripte aus der Anwendung AD2Ast ausser `ad2ast_auth.pl` installiert. Dieses Skript musste auf dem Asteriskserver verbleiben, da es die lokale Datei `/etc/asterisk/voicemail.conf` ausliest. Ferner mussten auf dem Webserver die Perl-Module `Net::DNS` und `Net::LDAP` nachinstalliert werden. Der Apache2-Webserverdienst wurde um einen eigenen *Virtual Host* erweitert, der nun unter der IP-Adresse `192.168.1.18` ansprechbar ist. Das laborinterne Active Directory wurde um die Gruppe `Domänen-Asterisk` erweitert, die bei den Benutzern `Lutz Grünwoldt` und `Felix Ogris` als zusätzliche Gruppe eingetragen wurde. Diesen Benutzern wurde zudem jeweils eine IP-Telefonnummer zugewiesen.

8. Ausblick

8.1. Todo

In dieser Studienarbeit konnten (leider) nicht alle Aspekte zu Voice over IP und Asterisk betrachtet werden. Die nachfolgend aufgeführten Themen und Ideen könnten daher für spätere Arbeiten als Grundlage dienen.

8.1.1. ENUM & DUNDi

ENUM und DUNDi dienen zum automatischen Auffinden von Telefonieteilnehmern. Dieses ergibt sich unter anderem aus der Problematik, dass jeder Internetnutzer seine eigene kleine Vermittlungsstelle in Form eines Asteriskservers betreiben kann. Während ENUM als weitestgehend standarisiertes Verfahren das Domain Name System nutzt, stammt DUNDi aus der Feder von Mark Spencer und bedarf vor dem Einsatz einer manuellen Vermaschung der einzelnen Asteriskserver. Neben einer Beschreibung der reinen Funktionsweise und der Konfiguration in Asterisk sollten hierbei vor allem Sicherheitsaspekte und eventuelle Routingprobleme o.ä. erläutert werden.

8.1.2. AGI Skripte

Das *Asterisk Gateway Interface* (s. Kapitel 3.3.11) ermöglicht es, eigene Programme aus dem Dialplan heraus aufzurufen und somit dynamische und benutzergesteuerte Funktionen zu realisieren. Neben der Schnittstellenbeschreibung wären hierbei vor allem interessante Ideen gefragt, die sich realisieren lassen.

8.1.3. Protokolluntersuchung

SIP/RTP und IAX wurden im Rahmen dieser Studienarbeit nur als *Blackbox* angesehen. Wie sie implementiert sind, wie ihr Verhalten in Weitverkehrsnetzen in Bezug auf Latenz und Güte sind und welche etwaigen Designfehler bei diesen Protokollen zu Sicherheitsproblemen führen könnten, stellt ein weiteres, spannendes Thema dar.

8.1.4. Asterisk-Module

Alle Funktionen, Protokolle und Codecs sind als Module in Form von *shared objects* (Dateiendung *.so) realisiert. Asterisk kann somit nahezu beliebig erweitert werden. Neben der notwendigen Interfacebeschreibung wären hier Prototypen und idealerweise ausprogrammierte, noch dringend benötigte Module erforderlich.

8.2. Version 1.4

Während der Niederschrift dieser Studienarbeit erschien Asterisk in der Version 1.4. Die neuen oder verbesserten Funktionen konnten jedoch nicht mehr mit einfließen. Für eine detaillierte Aufzählung aller Änderungen zur Vorgängerversion ist die Datei `CHANGES` aus dem Quellpaket von Asterisk 1.4 heranzuziehen. Besonders hervorzuheben sind folgende Neuerungen:

- AEL, die *Asterisk Extension Language*, wird nicht mehr als experimentell betrachtet und steht somit offiziell als Ersatz für die herkömmliche Syntax der `extension.conf` zur Verfügung
- IMAP-Server (*Internet Mail Access Protocol*) können zur Speicherung von Voicemailnachrichten verwendet werden

- Zustandsabfragen des Asteriskservers per *Simple Network Management Protocol* (SNMP) sind nativ möglich
- Das Asterisk Manager Interface ist nun über einen integrierten HTTP-Server ansprechbar.

A. Literatur

[Blank u. Dieterle 2004] Blank, Petra ; Dieterle, Stefan: *ENUM-Domains bei der DENIC eG*. Version: March 2004. http://www.denic.de/media/pdf/enum/veranstaltungen/pre-reader_20040316.pdf, Abruf: 2006-11-05

[Diverse a] Diverse: *InterAsterisk eXchange*. http://de.wikipedia.org/wiki/InterAsterisk_eXchange, Abruf: 2007-01-12

[Diverse b] Diverse: *Session Initiation Protocol*. http://de.wikipedia.org/wiki/Session_Initiation_Protocol, Abruf: 2007-01-17

[Diverse c] Diverse: *SNOM*. http://de.wikipedia.org/wiki/SNOM, Abruf: 2007-01-21

[Diverse d] Diverse: *Telephone Number Mapping*. http://de.wikipedia.org/wiki/ENUM, Abruf: 2007-01-17

[Diverse e] Diverse: *voip-info.org*. http://http://www.voip-info.org/, Abruf: 2007-01-21. – Anmerkung des Autors: **das** Voice over IP-Wiki

[Gurow 2005] Gurow, Lars: *Snom bringt kostenloses VoIP-Softphone*. Version: March 2005. http://www.netzwelt.de/news/70233-snom-bringt-kostenloses-voipsoftphone.html, Abruf: 2007-01-17

[Meggelen u. a. 2005] Meggelen, Jim V. ; Smith, Jared ; Madsen, Leif: *Asterisk - The Future of Telephony*. Version: September 2005. http://www.nufone.net/downloads/asteriskdocs/AsteriskTFOT.zip, Abruf: 2007-01-12. ISBN 0–596–00962–3

[Schildt 2004] Schildt, Holger: *VoIP mit IAX*. Version: April 2004. http://archiv.tu-chemnitz.de/pub/2004/0051/, Abruf: 2007-01-12

B. Software

B.1. Asteriskserver

SuSE Linux 10.0 ftp://ftp-stud.fht-esslingen.de/Mirrors/ftp.suse.com/pub/suse/i386/10.0/iso/

Asterisk 1.2.13 http://www.asterisk.org/

Zaptel 1.2.11 http://www.asterisk.org/

Deutsches Sprachset für Asterisk Version 2.0 http://www.stadt-pforzheim.de/asterisk/

PostgreSQL 8.1.5 http://www.postgresql.org

gastman 1.0-RC1 http://ftp.digium.com/pub/gastman/

Flash Operator Panel 0.26 http://www.asternic.org/

mpg123 0.61 http://www.mpg123.de/

FRITZ!Card Kernelmodul für openSuSE 10.2 http://opensuse.fltronic.de/SUSE10_2.htm

B.2. Softphones

X-Lite for Windows 3.0 / X-Lite for Mac OS X 2.0 http://www.counterpath.com

Snom 360 Softphone 5.3 http://www.snom.de

3CX Phone http://www.3cx.com/VOIP/voip-phone.html

Idefisk for Windows 1.37Idefisk for Mac OS X 1.35 http://www.asteriskguru.com/idefisk/

JackenIAX 1.0beta http://www.jackenhack.com/jackeniax/

Kiax 0.8.5 http://www.kiax.org/

Wengo 2.0 http://www.openwengo.org

Sipps 2.1.6 http://www.nero.com/sippstar/deu/SIPPS_Light.html

C. PostgreSQL Startskript

```
1  #!/bin/sh
2
3  # chkconfig: 2345 90 10
4
5  cd / || exit 1
6
7  case "$1" in
8    start)
9      sudo -u pgsql /usr/local/pgsql/bin/pg_ctl start -w \
10                   -D /usr/local/pgsql/data -l /var/tmp/pg_ctl.log
11     ;;
12   stop)
13     sudo -u pgsql /usr/local/pgsql/bin/pg_ctl stop \
14                   -D /usr/local/pgsql/data
15     ;;
16   *)
17     echo "usage: $0 <start|stop>"
18     exit 1
19     ;;
20 esac
```

D. Asterisk Startskript

```
1  #!/bin/sh
2
3  # chkconfig: 2345 90 10
4
5  cd / || exit 1
6
7  case "$1" in
8    start)
9      /usr/sbin/asterisk
10     ;;
11   stop)
12     /usr/sbin/asterisk -r -x "stop gracefully"
13     ;;
14   *)
15     echo "usage: $0 <start|stop>"
16     exit 1
17     ;;
18 esac
```

E. AD2Ast

E.1. ad2ast_auth.pl

```
1  #!/usr/bin/perl
2
3  # Fehlerausgabe schliessen
4  close(stderr);
5
6  # Module einbinden
7  use MIME::Base64;
8  use Digest::MD5 (md5);
9
10 # Deskriptoren auf Auto-Flush
11 $| = 1;
12
13 # zufaellige Bytes einlesen
14 open(FH, "/dev/urandom") or &no();
15 &no() unless (read(FH, $nonce, 60) == 60);
16 close(FH);
17
18 # Nonce an den Client senden
19 print encode_base64($nonce, "") . "\r\n";
20
21 # Benutzernamen und Hash vom Client lesen
22 $user = <>;
23 $imposed_hash = <>;
24 close(stdin);
25
26 # Newline strippen und dekodieren
27 foreach ($user, $imposed_hash) {
28   &no() unless s/\r\n$//o;
29   $_ = decode_base64($_);
30 }
31
32 # voicemail.conf einlesen
33 open(FH, "/etc/asterisk/voicemail.conf") or &no();
34 @config = map { s/\r|\n//sgio; $_ } <FH>;
35 close(FH);
36
37 # voicemail.conf parsen
38 foreach (@config) {
39   if (/^\[([^\]]+)\]$/) {
40     # Kontext Deklaration
41     if ($1 eq "default") { $in_context = 1; }
42     else { $in_context = 0; }
43     next;
44   }
45   # Voicemail-Kontext gefunden?
46   next unless $in_context;
47   # gueltigen Eintrag gefunden?
48   next unless /^(\d+)\s*\=\>\s*([^\,]+),/;
49   # Benutzernummer gefunden?
50   next unless ($1 eq $user);
51   # Hashsummen vergleichen
52   &yes() if ($imposed_hash eq md5($user, $2, $nonce));
53   last;
54 }
55
56 # default=verbieten & Ende
57 &no();
58
59
60 ##############################################################################
61 # lokale Subroutinen                                                         #
62 ##############################################################################
63
64 sub no () { print "no\r\n"; exit 1; }
65 sub yes () { print "ok\r\n"; exit 0; }
```

E.2. ad2ast_dial.pl

```
1  #!/usr/bin/perl
2
3  # Module einbinden
4  use MIME::Base64;
5  use IO::Socket;
6  use DBI;
7  use Digest::MD5 (md5);
8
9  # Subroutinen und Konfiguration einlesen
10 require "ad2ast_subs.pl";
11
12 # Deskriptoren auf Auto-Flush
13 $| = 1;
14
15 # CGI-Parameter einlesen
16 if ($ENV{REQUEST_METHOD} =~ /^post$/i) {
17   read(STDIN, $post_string, $ENV{CONTENT_LENGTH});
18 }
19 else {
20   $post_string = $ENV{QUERY_STRING};
21 }
22
23 # CGI-Parameter in einen Perl-Hash wandeln
24 &http_vars($post_string, \%http_post);
25
26 # Username + Passwort bestimmen...
27 if ($http_post{action} eq "login") {
28   # ...beim Login aus den Eingabefeldern
29   $user = $http_post{username};
30   $pass = $http_post{password};
31 }
32 elsif ($http_post{action}) {
33   # ...sonst aus dem Cookie
34   ($user, $pass) = split(/:/, decode_base64(
35                                        (split(/=/, $ENV{HTTP_COOKIE}, 2))[1]));
36 }
37
38 # Username + Passwort testen, falls nicht die Startseite aufgerufen wurde
39 $err = &check_credentials($user, $pass) if $http_post{action};
40 $http_post{action} = "" if $err;
41
42 # Verbindung zum Asterisk Manager und zur Datenbank aufbauen und
43 # ...Telefonnummern aus der DB laden, falls nicht die Startseite oder
44 # ...die Logoutseite gewaehlt wurden
45 if (($http_post{action} ne "") && ($http_post{action} ne "logout")) {
46   $err = &manager_connect(\$mgr_sock);
47   $err = &db_connect(\$dbh) unless $err;
48   $err = &load_records(\%records, $dbh) unless $err;
49   &fatal_error($err) if $err;
50 }
51
52 ##############################################################################
53 # Aktionsauswertung                                                          #
54 ##############################################################################
55
56 if ($http_post{action} eq "login") {
57   # Cookie setzen und Startseite anzeigen
58   &set_cookie(encode_base64("$user:$pass", ""));
59   &page_start("Willkommen", 1);
60   print "<p>Hallo, $records{$user}->{name}.</p>\n";
61   &page_end(1);
62 }
63 elsif ($http_post{action} eq "diallist") {
64   # alle Telefonnummern ausgeben
65   &diallist(\%records, "Teilnehmerliste", "&action=diallist");
66 }
67 elsif ($http_post{action} eq "search") {
68   # Suchmaske
69   &page_start("Teilnehmer suchen", 1);
```

```
70    print "<form␣method=\"POST\">\n" .
71          "<input␣type=\"hidden\"␣name=\"action\"␣value=\"dosearch\">\n" .
72          "<table><tr><td>Name:</td>" .
73          "<td><input␣type=\"text\"␣name=\"name\"␣value=\"*\"></td></tr>\n" .
74          "<tr><td>Email:</td>" .
75          "<td><input␣type=\"text\"␣name=\"mail\"␣value=\"*\"></td></tr>\n" .
76          "<tr><td>Telefon:</td>" .
77          "<td><input␣type=\"text\"␣name=\"phone\"␣value=\"*\"></td></tr>\n" .
78          "</table><br>\n<input␣type=\"submit\"␣value=\"Suchen\">\n</form>\n";
79    &page_end(1);
80  }
81  elsif ($http_post{action} eq "dosearch") {
82    # Suche durchfuehren
83    %search_result = ();
84    $urlext = "&action=dosearch";
85    foreach (phone, name, mail) {
86      $urlext .= "&$_=$http_post{$_}";
87      # Suche per Regexp
88      $search{$_} = "^" . $http_post{$_} . "\$";
89      $search{$_} =~ s/\*/\.\*/sgio;
90    }
91    foreach $phone(sort keys %records) {
92      $found_phone = 0;
93      # alle Telefonnummern vergleichen
94      foreach ($phone, keys %{$records{$phone}->{extra_phones}}) {
95        next unless /$search{phone}/;
96        $found_phone = 1;
97        last;
98      }
99      next unless $found_phone;
100     next unless ($records{$phone}->{name} =~ /$search{name}/);
101     next unless ($records{$phone}->{mail} =~ /$search{mail}/);
102     $search_result{$phone}->{mail} = $records{$phone}->{mail};
103     $search_result{$phone}->{name} = $records{$phone}->{name};
104     $search_result{$phone}->{extra_phones} = $records{$phone}->{extra_phones};
105   }
106   # Ergebnis der Suche ausgeben
107   &diallist(\%search_result, "Suchergebnis", $urlext);
108 }
109 elsif ($http_post{action} eq "dial") {
110   # Anrufen: Hilfetext ausgeben und weiterleiten
111   &page_start("Anrufen", 1, "?action=dodial&exten=$http_post{exten}" .
112                             "&dialexten=$http_post{dialexten}");
113   print "<p>Ihr␣Telefon␣(Durchwahl␣$user)␣klingelt␣nun.<br>Sobald␣Sie␣das␣" .
114         "Gespräch␣angenommen␣haben,␣wird␣versucht,␣" .
115         $records{$http_post{exten}}->{name} . "␣(Durchwahl␣" .
116         $http_post{dialexten} . ")␣anzurufen.</p>\n";
117   &page_end(1);
118 }
119 elsif ($http_post{action} eq "dodial") {
120   # Missbrauch abfangen
121   &fatal_error("Sie␣dürfen␣sich␣nicht␣selbst␣anrufen.")
122     if ($user eq $http_post{dialexten});
123   $found_number = 0;
124   foreach (keys %records) {
125     $found_number = 1 if ($_ eq $http_post{dialexten});
126     foreach (keys %{$records{$_}->{extra_phones}}) {
127       $found_number = 1 if ($_ eq $http_post{dialexten});
128     }
129   }
130   &fatal_error("Unbekannter␣Teilnehmer.") unless $found_number;
131
132   # Anrufen: Waehlen (blockiert bis Anrufer abnimmt)
133   $err = &manager_action($mgr_sock, \$result, "Command",
134                          Command => "show␣dialplan␣$user\@$voicemail_context");
135   &fatal_error($err) if $err;
136   $channel = $1 if ($result =~ /Dial\(([^\|\)]+)/);
137   &fatal_error("Ihr␣Telefon␣wird␣vom␣Dialplan␣nicht␣erreicht!") unless $channel;
138   $err = &manager_action($mgr_sock, \$result, "Originate",
139                          Channel => $channel,
140                          Exten => $http_post{dialexten},
```

```
141                             Priority => 1,
142                             Context => $voicemail_context,
143                             CallerID => "$user <$records{$user}->{name}>");
144     &page_start("Anrufen", 1);
145     if ($err) {
146       # keine Standardfehlerseite hier!
147       print "<p class=\"error\">Der Anruf konnte nicht platziert werden:</p>\n" .
148             "<pre style=\"margin-left:10px\">$err\n$result</pre>\n" .
149             "<p>Eventuell haben Sie nicht abgenommen..? ;-)</p>\n";
150     }
151     else {
152       print "<p>Bleiben Sie am Telefon, bis " .
153             $records{$http_post{exten}}->{name} . " (Durchwahl " .
154             $http_post{dialexten} . ") abgenommen hat.</p>\n";
155     }
156     print "<p><a href=\"javascript:back()\">Weiter</a></p>\n";
157     &page_end(1);
158   }
159   elsif ($http_post{action} eq "extra") {
160     # Meine (Extra-)Rufnummern anzeigen
161     &page_start("Meine Rufnummern", 1);
162     print "<form method=\"POST\">\n" .
163           "<input type=\"hidden\" name=\"action\" value=\"extra-add\">\n" .
164           "<table class=\"table1\">\n<tr><th>Telefon</th>\n<th>Notiz</th>\n" .
165           "<th>Aktion</th></tr>\n";
166     $color = 1;
167     foreach (sort keys %{$records{$user}->{extra_phones}}) {
168       print "<tr>\n<td class=\"td$color\">$_</td>\n" .
169             "<td class=\"td$color\">$records{$user}->{extra_phones}->{$_}</td>\n".
170             "<td class=\"td$color\"><a href=\"?action=extra-delete&phone=$_\">" .
171             "Löschen</a></td>\n</tr>\n";
172       $color = 3 - $color;
173     }
174     # Eingabefelder fuer neue Rufnummer
175     print "<tr>\n" .
176           "<td class=\"td$color\"><input type=\"text\" name=\"phone\"></td>\n" .
177           "<td class=\"td$color\"><input type=\"text\" name=\"comment\"></td>\n" .
178           "<td class=\"td$color\"><input type=\"submit\" value=\"Eintragen\">" .
179           "</td>\n</tr>\n</table>\n";
180     &page_end(1);
181   }
182   elsif ($http_post{action} eq "extra-add") {
183     # neue Rufnummer eintragen
184     $err = &db_prepare($dbh, \$sth, "INSERT INTO $user_table " .
185                                     "(phone, extra_phone, comment) " .
186                                     "VALUES (?, ?, ?)");
187     &fatal_error($err) if $err;
188     $err = &db_execute($sth, $user, $http_post{phone}, $http_post{comment});
189     &fatal_error($err) if $err;
190     &db_finish($sth);
191     &page_start("Meine Rufnummern", 1);
192     print "<p>Ihre Telefonnummer wurde eingetragen.</p>\n" .
193           "<p><a href=\"?action=extra\">Zurück zur Übersicht</a></p>\n";
194     &page_end(1);
195   }
196   elsif ($http_post{action} eq "extra-delete") {
197     # Rufnummer loeschen
198     $err = &db_prepare($dbh, \$sth, "DELETE FROM $user_table " .
199                                     "WHERE phone=? AND extra_phone=?");
200     &fatal_error($err) if $err;
201     $err = &db_execute($sth, $user, $http_post{phone});
202     &fatal_error($err) if $err;
203     &db_finish($sth);
204     &page_start("Meine Rufnummern", 1);
205     print "<p>Ihre Telefonnummer wurde gelöscht.</p>\n" .
206           "<p><a href=\"?action=extra\">Zurück zur Übersicht</a></p>\n";
207     &page_end(1);
208   }
209   elsif ($http_post{action} eq "logout") {
210     # Cookie loeschen und Logoutseite anzeigen
211     &destroy_cookie();
```

```
212   &page_start("Abmeldung");
213   &center_page_start("Abmeldung");
214   print "<p>Sie haben sich vom System abgemeldet.</p>\n" .
215         "<p><a href=\"ad2ast_dial.pl\">Hier können Sie sich neu anmelden." .
216         "</a></p>\n";
217   &center_page_end();
218   &page_end();
219 }
220 else {
221   # Default: Loginseite anzeigen, eventuell mit Fehlertext
222   &login_page($err);
223 }
224
225 # Verbindungen abbauen
226 if (defined $dbh) {
227   &db_disconnect($dbh);
228   &manager_disconnect($mgr_sock);
229 }
230
231 # Programmende
232 exit(0);
233
234
235 ##############################################################################
236 # lokale Subroutinen                                                         #
237 ##############################################################################
238
239 # Verbindung zum Asteriskmanager aufbauen
240 sub manager_connect ()
241 {
242   my $sock = shift;
243   my $result = "";
244
245   $$sock = new IO::Socket::INET(PeerAddr => $manager_host,
246                                 PeerPort => $manager_port,
247                                 Proto    => "tcp")
248     or return "Asterisk manager: $!";
249   my $msg = <$$sock>;
250   return &manager_action($$sock, \$result, "Login",
251                          Username => $manager_user,
252                          Secret => $manager_pass,
253                          Events => "off");
254 }
255
256 # Verbindung zum Asteriskmanager beenden
257 sub manager_disconnect ()
258 {
259   my $sock = shift;
260   close($sock);
261 }
262
263 # Befehl zum Asteriskmanager absetzen
264 sub manager_action ()
265 {
266   my $sock = shift;
267   my $result = shift;
268   my $action = shift;
269   my %kvhash = @_;
270
271   # Aktion + etwaige Schluessel-/Wertepaare senden
272   print $sock "Action: $action\r\n";
273   while (my ($key, $value) = each %kvhash) {
274     print $sock "$key: $value\r\n";
275   }
276   print $sock "\r\n";
277
278   my $status = "";
279   $$result = "";
280
281   while (<$sock>) {
282     $$result .= $_;
```

```
283       if (/^([a-zA-Z]+): ([^\s]+)/) {
284         # Status in Form von "Key: Value"
285         my $key = $1;
286         my $value = $2;
287         if ($key eq "Response") { $status = $value; }
288       }
289       elsif (/^\s*$/) {
290         # Leerzeile markiert Ende, sofern wir einen Status haben
291         next if !$status;
292         last unless ($status eq "Follows");
293       }
294       elsif (/^--END COMMAND--/) {
295         # laengere Ausgaben werden durch "--END COMMAND--" beendet
296         last if ($status eq "Follows");
297       }
298     }
299     return "" if (($status eq "Success") || ($status eq "Follows"));
300     return "Asterisk manager: $status";
301   }
302
303   # zum eigenen Auth-Server verbinden und Username/Passwort verifizieren
304   sub check_credentials ()
305   {
306     my $user = shift;
307     my $pass = shift;
308     my $sock = new IO::Socket::INET(PeerAddr => $auth_host,
309                                     PeerPort => 6666,
310                                     Proto    => "tcp")
311       or return $!;
312
313     # Nonce vom Server lesen
314     my $nonce = <$sock>;
315     $nonce =~ s/\r\n$//o or return "Protokollfehler";
316     $nonce = decode_base64($nonce);
317
318     # gewuenschten User & salted Hash schreiben
319     print $sock encode_base64($user, "") . "\r\n" .
320                 encode_base64(md5($user, $pass, $nonce), "") . "\r\n";
321     my $result = <$sock>;
322     close($sock);
323
324     # Ergebnis auswerten
325     $result =~ s/\r\n$//o or return "Protokollfehler";
326     return "" if ($result eq "ok");
327     return "Falscher Benutzername oder Passwort" if ($result eq "no");
328     return "Protokollfehler";
329   }
330
331   # alle Telefonnummer aus der Datenbank laden
332   sub load_records ()
333   {
334     my $records = shift;
335     my $dbh = shift;
336     my $sth;
337
338     # Telefonnummern aus dem AD
339     $err = &db_prepare($dbh, \$sth, "SELECT * FROM $ad_table");
340     return $err if $err;
341     $err = &db_execute($sth);
342     return $err if $err;
343     while (my $row = $sth->fetchrow_hashref()) {
344       $records->{$row->{phone}}->{name} = $row->{name};
345       $records->{$row->{phone}}->{mail} = $row->{mail};
346     }
347     &db_finish($sth);
348
349     # von den Benutzern gefuehrte Telefonlisten
350     $err = &db_prepare($dbh, \$sth, "SELECT * FROM $user_table");
351     return $err if $err;
352     $err = &db_execute($sth);
353     return $err if $err;
```

```
354     while (my $row = $sth->fetchrow_hashref()) {
355       $records->{$row->{phone}}->{extra_phones}->{$row->{extra_phone}} =
356                                                                 $row->{comment};
357     }
358     &db_finish($sth);
359     return "";
360   }
361
362   # dem CGI-Skript uebergebene Variablen in einen Perlhash schreiben
363   sub http_vars ()
364   {
365     my $vars_string = shift;
366     my $ret_ref = shift;
367
368     foreach (split(/\&/, $vars_string)) {
369       my ($key, $value) = split(/\=/, $_);
370       $key =~ tr/+/ /;
371       $key =~ s/%([0-9a-fA-F]{2})/pack("C", hex($1))/ge;
372       $value =~ tr/+/ /;
373       $value =~ s/%([0-9a-fA-F]{2})/pack("C", hex($1))/ge;
374       $ret_ref->{"$key"} = "$value";
375     }
376   }
377
378   # HTML-Tabelle mit Telefonnummern ausgeben
379   sub diallist ()
380   {
381     my $values = shift;
382     my $title = shift;
383     my $url_ext = shift;
384
385     &page_start($title, 1);
386     $dir = ($http_post{dir} eq "desc" ? "asc" : "desc");
387     print "<table class=\"table1\">\n<tr>" .
388           "<th><a href=\"?sort=name&dir=$dir$url_ext\">Name</a></th>\n" .
389           "<th><a href=\"?sort=mail&dir=$dir$url_ext\">Email</a></th>\n" .
390           "<th><a href=\"?sort=phone&dir=$dir$url_ext\">Telefon</a></th>\n" .
391           "<th>Aktion</th></tr>\n";
392     $color = 2;
393     $sort = $http_post{sort};
394     $sort = "name" if (($sort ne "mail") && ($sort ne "phone"));
395     $dir = ($http_post{dir} eq "desc" ? -1 : 1);
396     foreach (sort {
397       ($values->{$a}->{$sort} cmp $values->{$b}->{$sort}) * $dir
398     } keys %$values) {
399       $skip = scalar keys %{$values->{$_}->{extra_phones}};
400       $skip++;
401       $color = 3 - $color;
402       print "<tr>\n".
403             "<td rowspan=\"$skip\" class=\"td$color\">$values->{$_}->{name}</td>".
404             "<td rowspan=\"$skip\" class=\"td$color\">$values->{$_}->{mail}</td>".
405             "<td class=\"td$color\">$_</td>\n<td class=\"td$color\">";
406       if ($user ne $_) {
407         # Benutzer soll sich nicht selbst anrufen koennen
408         print "<a href=\"?action=dial&exten=$_&dialexten=$_\">Anrufen</a>";
409       }
410       print "</td>\n</tr>\n";
411       foreach $phone(sort keys %{$values->{$_}->{extra_phones}}) {
412         print "<tr>\n<td class=\"td$color\">$phone</td>\n<td class=\"td$color\">";
413         if ($user ne $phone) {
414           print "<a href=\"?action=dial&exten=$_&dialexten=$phone\">Anrufen</a>";
415         }
416         print "</td>\n</tr>\n";
417       }
418     }
419     print "</table>";
420     &page_end(1);
421   }
422
423   # Grundgeruest einer HTML-Seite ausgeben
424   sub page_start ()
```

```
425 {
426   my $title = shift;
427   my $menu = shift;
428   my $redirect = shift;
429   my $app_title = "AD2AST Dialer";
430   $app_title .= " - $title" if $title;
431   $redirect = "\n<meta http-equiv=\"Refresh\" content=\"0; url=$redirect\">"
432     if $redirect;
433
434   print <<EOF;
435 Content-Type: text/html
436
437 <html>
438 <head>
439 <title>$app_title</title>
440 <style type="text/css">
441 <!--
442 body {
443   color: black;
444   background: white;
445   font-family: sans-serif;
446   font-size: 12px;
447   margin: 0px;
448   padding: 0px;
449 }
450 th {
451   background-color:white;
452   padding:2px;
453 }
454 .table1 {
455   background-color:#fc0;
456 }
457 .td1 {
458   background-color:#a5a5a5;
459   padding:2px;
460 }
461 .td2 {
462   background-color:#d5d5d5;
463   padding:2px;
464 }
465 .error {
466   color:red;
467 }
468 .headImg {
469   float: left;
470   margin-top: 0;
471   margin-left: 0;
472   height: 86px;
473   width: 314px;
474   padding: 3px 3px 3px 18px;
475   background-color: #999999;
476   border: 0px solid black;
477 }
478 .headTxt {
479   float: left right;
480   margin: 0;
481   height: 86px;
482   padding: 3px;
483   background-color: #999999;
484   text-align: center;
485   font-weight: bold;
486   font-size: 124%;
487 }
488 //-->
489 </style>$redirect
490 </head>
491 <body>
492
493 <div class="headImg">
494 <img src="/fh_logo.gif" alt="FH-Logo" border="0">
495 </div>
```

```
496
497 <div class="headTxt">
498 <br>Labor f&#252;r Angewandte Informatik und Mathematik
499 </div>
500
501 EOF
502    if ($menu) {
503       print <<EOF;
504 <table width="99%" height="79%">
505 <tr><td valign="top" height="99%" style="border:1px solid black; padding:5px;">
506 <p style="white-space: nowrap">Eingeloggt als: 
507 $records{$user}->{name} ($user)</p><br>
508 <a href="?action=diallist">Teilnehmerliste</a><br><br>
509 <a href="?action=search">Teilnehmer suchen</a><br><br>
510 <a href="?action=extra">Meine Rufnummern</a><br><br>
511 <a href="?action=logout">Abmelden</a></td>
512 <td width="99%" height="99%" valign="top" style="padding-left:5px">
513 <h1>$title</h1>
514 EOF
515    }
516 }
517
518 # HTML-Seite beenden
519 sub page_end ()
520 {
521   my $menu = shift;
522
523    print "</td></tr></table>\n" if $menu;
524    print "</body></html>\n";
525 }
526
527 # Fehlerseite
528 sub fatal_error ()
529 {
530   &page_start("Fehler");
531   &center_page_start("Fehler");
532    print "<p class=\"error\">$_[0]</p>\n";
533   &center_page_end();
534   &page_end();
535    exit(1);
536 }
537
538 # in HTML mittig & zentriert ausgeben
539 sub center_page_start ()
540 {
541   my $title = shift;
542    print <<EOF;
543 <table width="99%" height="79%">
544 <tr>
545 <td width="99%" height="99%" align="center" valign="middle">
546 EOF
547    print "<h1>$title</h1>\n" if $title;
548 }
549
550 # mittig & zentriert beenden
551 sub center_page_end ()
552 {
553    print "</td></tr></table>\n";
554 }
555
556 # Loginmaske ausgeben
557 sub login_page ()
558 {
559   my $error = shift;
560
561   &page_start("Anmeldung");
562   &center_page_start("Anmeldung");
563    print "<p class=\"error\">$error</p>\n" if $error;
564    print <<EOF;
565 <p>Willkommen bei <i>Active Directory to Asterisk</i> (AD2Ast).<br>
566 Dies ist ein Demo-System des Labors für Angewandte Informatik und Mathematik
```

```
567 der FH Bielefeld.<br><br>
568 Mit diesem System können Sie eine Voice over IP-Verbindung zu einem
569 Gesprächspartner aufbauen,<br>
570 sofern dessen Kontaktdaten im laborinternen Active Directory eingetragen
571 sind.<br></p>
572 <form method="POST" name="loginform">
573 <input type="hidden" name="action" value="login">
574 <table>
575 <tr><td>Benutzer:</td><td><input type="text" name="username"></td></tr>
576 <tr><td>Passwort:</td><td><input type="password" name="password"></td></tr>
577 <tr><td colspan="2" align="center">
578 <input type="submit" value="Einloggen"></td></tr>
579 </table>
580 </form>
581 <script type="text/javascript">
582 <!--
583 if (document.loginform.username) document.loginform.username.focus();
584 //-->
585 </script>
586 EOF
587   &center_page_end();
588   &page_end();
589 }
590
591 # Cookie setzen
592 sub set_cookie ()
593 {
594   print "Set-Cookie: $cookie_name=$_[0]; path=$cookie_path\n";
595 }
596
597 # Cookie loeschen (=leeres Cookie schreiben)
598 sub destroy_cookie ()
599 {
600   &set_cookie();
601 }
```

E.3. ad2ast_subs.pl

```
1  ##############################################################################
2  # Konfigurationswerte                                                        #
3  ##############################################################################
4
5  ### Sync-Skript + Webfrontend
6  $db_type                          = "mysql";
7  $db_host                          = "localhost";
8  # $db_port                        = undef;
9  $db_name                          = "astdb";
10 $db_user                          = "asterisk";
11 $db_pass                          = "test";
12 $ad_table                         = "adtbl";
13 $user_table                       = "usrtbl";
14
15 ### Sync-Skript
16 $domain                           = "infma-labor.local";
17 # @nameservers                    = ( "192.168.1.6", "192.168.1.7" );
18 @ldap_servers                     = ( "192.168.1.6" );
19 # $base_dn                        = undef;
20 $bind_dn                          = "cn=Felix Ogris,ou=Stuff,ou=User";
21 $append_base_dn_to_bind_dn        = 1;
22 $bind_pw                          = "geheim";
23 $search_dn                        = "cn=Domänen-Asterisk,ou=Global=ou=Groups";
24 $append_base_dn_to_search_dn      = 1;
25
26 ### Webfrontend
27 $manager_host                     = "192.168.1.4";
28 $manager_port                     = 5038;
29 $manager_user                     = "ad2ast";
30 $manager_pass                     = "test";
31 $cookie_name                      = "ad2ast_id";
32 $cookie_path                      = "/";
33 $auth_host                        = "192.168.1.4";
34
35 ##############################################################################
36 # globale Subroutinen                                                        #
37 ##############################################################################
38 sub db_connect ()
39 {
40   my $dbh = shift;
41
42   my $dsn = "DBI:$db_type:dbname=$db_name";
43   $dsn .= ";host=$db_host" if $db_host;
44   $dsn .= ";port=$db_port" if $db_port;
45
46   $$dbh = DBI->connect($dsn, $db_user, $db_pass,
47                        { RaiseError => 0, PrintError => 0 });
48   return "Database error: " . DBI::errstr if DBI::err;
49   return "";
50 }
51
52 sub db_disconnect ()
53 {
54   my $dbh = shift;
55   $dbh->disconnect();
56 }
57
58 sub db_prepare ()
59 {
60   my $dbh = shift;
61   my $sth = shift;
62
63   $$sth = $dbh->prepare(@_);
64   return "Database error: " . $dbh->errstr if $dbh->err;
65   return "";
66 }
67
68 sub db_finish ()
69 {
```

```
70    my $sth = shift;
71    $sth->finish();
72    $sth = undef;
73  }
74
75  sub db_execute ()
76  {
77    my $sth = shift;
78
79    $sth->execute(@_);
80    return "Database error: " . $sth->errstr if $sth->err;
81    return "";
82  }
83
84  sub db_begin_transaction ()
85  {
86    my $dbh = shift;
87    $dbh->begin_work();
88    return "Database error: " . $dbh->errstr if $dbh->err;
89    return "";
90  }
91
92  sub db_end_transaction ()
93  {
94    my $dbh = shift;
95    $dbh->commit();
96    return "Database error: " . $dbh->errstr if $dbh->err;
97    return "";
98  }
```

E.4. ad2ast_sync.pl

```
#!/usr/bin/perl

# Module einbinden
use Net::DNS;
use Net::LDAP;
use DBI;

# Subroutinen und Konfiguration einlesen
require "ad2ast_subs.pl";

# Base DN + Bind DN bestimmen
$base_dn = join(",", map { "dc=$_" } split(/\./, $domain))
  unless defined $base_dn;
$bind_dn .= ",$base_dn" if $append_base_dn_to_bind_dn;

# Search DN bestimmen
$search_dn .= ",$base_dn" if $append_base_dn_to_search_dn;

# LDAP Server ermitteln
unless (defined @ldap_servers) {
  $err = &get_ldap_servers($domain, \@ldap_servers, @nameservers);
  die $err if $err;
}

# LDAP Server konnektieren
$ldap = Net::LDAP->new(\@ldap_servers) or die "$!";

# LDAP Server binden
$result = $ldap->bind($bind_dn, password => $bind_pw);
die $result->error_text if $result->is_error;

# Personen suchen
$search = $ldap->search(base   => $base_dn,
                        scope  => "sub",
                        attrs  => [ "displayName", "mail",
                                    "ipPhone", #"otherIpPhone",
                                    #"telephoneNumber", "otherTelephone",
                                    #"homePhone", "otherHomePhone",
                                    #"mobile", "otherMobile"
                                  ],
                        filter => "(&(ipPhone=*)(memberOf=$search_dn))");

# Ergebnis holen
$entries = $search->as_struct();

# Ergebnis normalisieren
%ad_records = ();
foreach $dn(sort keys %$entries) {
  my %values = ();
  foreach $attr(sort map { lc($_) } keys %{$entries->{$dn}}) {
    $value = ${$entries->{$dn}->{$attr}}[0];
    if ($attr eq "ipphone") {
      warn "phone_number_$value_is_used_multiple_time"
        if exists $ad_records{$value};
      $ad_records{$value} = \%values;
    }
    else {
      warn "object_$dn_has_multiple_$attr_attributes" if exists $values{$attr};
      $values{$attr} = $value;
    }
  }
}

# LDAP Verbindung beenden
$ldap->unbind();
$ldap->disconnect();

# Datenbank konnektieren
$err = &db_connect(\$dbh);
```

```
70  die $err if $err;
71
72  # Transaktionsblock einleiten
73  $err = &db_begin_transaction($dbh);
74  die $err if $err;
75
76  # Bestandsdaten aus der AD Tabelle auslesen
77  $err = &db_prepare($dbh, \$sth, "SELECT * FROM $ad_table FOR UPDATE");
78  die $err if $err;
79  $err = &db_execute($sth);
80  die $err if $err;
81  %old_ad_records = %{$sth->fetchall_hashref("phone")};
82  &db_finish($sth);
83
84  # Statements zum Einfuegen, Aktualisieren und Loeschen vorbereiten
85  $i = 0;
86  foreach ("INSERT INTO $ad_table (phone, mail, name, last_mod) " .
87                                          "VALUES (?, ?, ?, now())",
88           "UPDATE $ad_table SET mail=?, name=?, last_mod=now() WHERE phone=?",
89           "DELETE FROM $ad_table WHERE phone=?") {
90    $err = &db_prepare($dbh, \$sth[$i], $_);
91    die $err if $err;
92    $i++;
93  }
94
95  # 1. Schritt: alles loeschen, was nicht mehr im AD vorhanden ist
96  foreach $phone(keys %old_ad_records) {
97    next if exists $ad_records{$phone};
98    $err = &db_execute($sth[2], $phone);
99    die $err if $err;
100 }
101
102 # 2. Schritt: neue Eintraege aus dem AD einfuegen und bestehende aktualisieren
103 foreach $phone(keys %ad_records) {
104   $mail = $ad_records{$phone}->{mail};
105   $name = $ad_records{$phone}->{displayname};
106   if (!exists $old_ad_records{$phone}) {
107     # neuer Eintrag
108     $err = &db_execute($sth[0], $phone, $mail, $name);
109     die $err if $err;
110     next;
111   }
112   # bestehender Eintrag -> Werte vergleichen
113   $old_mail = $old_ad_records{$phone}->{mail};
114   $old_name = $old_ad_records{$phone}->{name};
115   if (($mail ne $old_mail) || ($name ne $old_name)) {
116     $err = &db_execute($sth[1], $mail, $name, $phone);
117     die $err if $err;
118   }
119 }
120
121 # Datenbankverbindung beenden
122 $err = &db_end_transaction($dbh);
123 die $err if $err;
124 foreach (@sth) {
125   &db_finish($_);
126 }
127 &db_disconnect($dbh);
128
129 # Programmende
130 exit 0;
131
132
133 ##############################################################################
134 # lokale Subroutinen                                                         #
135 ##############################################################################
136
137 # IP-Adressen und Portnummern zu LDAP-Servern in einem AD herausfinden
138 sub get_ldap_servers ()
139 {
140   my $domain = shift;
```

```
141   my $return_array = shift;
142   my %ldap_servers = ();
143
144   # 1. Schritt: SRV-Records holen
145   my $dns_res = Net::DNS::Resolver->new();
146   $dns_res->nameservers(@_) unless ($#_ < 0);
147   my $dns_pkt = $dns_res->query("_ldap._tcp." . $domain, "SRV");
148   return $dns_res->errorstring unless defined $dns_pkt;
149   foreach my $dns_rr($dns_pkt->answer()) {
150     next unless ($dns_rr->type() eq "SRV");
151     my $prio = $dns_rr->priority();
152     my $weight = $dns_rr->weight();
153     my $target = $dns_rr->target();
154     $ldap_servers{$prio}->{$weight}->{$target} = $dns_rr->port();
155   }
156
157   # 2. Schritt: SRV-Records sortieren und IP-Adressen bestimmen
158   foreach my $prio(sort { $b <=> $a } keys %ldap_servers) {
159     my $this_prio = $ldap_servers{$prio};
160     foreach my $weight(sort { $b <=> $a } keys %$this_prio) {
161       my $this_weight = $this_prio->{$weight};
162       foreach my $target(sort keys %$this_weight) {
163         my $dns_pkt = $dns_res->query($target);
164         next unless defined $dns_pkt;
165         foreach my $dns_rr($dns_pkt->answer()) {
166           next unless ($dns_rr->type() eq "A");
167           # IP:Port gefunden!
168           push @$return_array, $dns_rr->address() . ":$this_weight->{$target}";
169         }
170       }
171     }
172   }
173
174   return "no_LDAP_servers_available" if ($#$return_array < 0);
175   return "";
176 }
```

E.5. ad2ast_xml.pl

```
1  #!/usr/bin/perl
2
3  # Module einbinden
4  use DBI;
5
6  # Subroutinen und Konfiguration einlesen
7  require "ad2ast_subs.pl";
8
9  # Datenbank konnektieren
10 $err = &db_connect(\$dbh);
11
12 # Daten aus dem AD
13 $err = &db_prepare($dbh, \$sth, "SELECT * FROM $ad_table");
14 return $err if $err;
15 $err = &db_execute($sth);
16 return $err if $err;
17 while (my $row = $sth->fetchrow_hashref()) {
18   $records{$row->{phone}}->{name} = $row->{name};
19   $records{$row->{phone}}->{mail} = $row->{mail};
20 }
21 &db_finish($sth);
22
23 # von den Benutzern geführte Telefonlisten
24 $err = &db_prepare($dbh, \$sth, "SELECT * FROM $user_table");
25 return $err if $err;
26 $err = &db_execute($sth);
27 return $err if $err;
28 while (my $row = $sth->fetchrow_hashref()) {
29   $records{$row->{phone}}->{extra_phones}->{$row->{extra_phone}} =
30                                                          $row->{comment};
31 }
32 &db_finish($sth);
33 &db_disconnect($dbh);
34
35 # XML ausgeben
36 print "Content-Type: text/xml\n\n<?xml version=\"1.0\"?>\n<AddressBook>\n";
37 foreach (sort keys %records) {
38   my ($fname, $lname) = split(/ /, $records{$_}->{name});
39   &print_contact($fname, $lname, $_);
40   foreach $phone(sort keys %{$records{$_}->{extra_phones}}) {
41     &print_contact($fname, $lname, $phone);
42   }
43 }
44 print "</AddressBook>\n";
45
46 # Programmende
47 exit(0);
48
49
50 ##############################################################################
51 # lokale Subroutinen                                                         #
52 ##############################################################################
53
54 # einen Telefonbucheintrag ausgeben
55 sub print_contact ()
56 {
57   my $fname = shift;
58   my $lname = shift;
59   my $phone = shift;
60
61   print "<Contact>\n<LastName>$lname</LastName>\n" .
62         "<FirstName>$fname</FirstName>\n<Phone>\n" .
63         "<phonenumber>$phone</phonenumber>\n" .
64         "<accountindex>0</accountindex>\n</Phone>\n</Contact>\n";
65 }
```

E.6. ad2ast.sql

```
1  -- MySQL Datenbankschema fuer AD2Ast
2
3  CREATE TABLE adtbl (
4     phone      TEXT,
5     mail       TEXT,
6     name       TEXT,
7     last_mod DATETIME
8  );
9
10 CREATE TABLE usrtbl (
11    phone         TEXT,
12    extra_phone TEXT,
13    comment       TEXT
14 );
```

E.7. ad2ast_auth

```
1  # /etc/xinetd.d/ad2ast_auth
2  # description: Authentifizierungsdienst fuer AD2Ast
3
4  service ad2ast_auth
5  {
6    socket_type    = stream
7    protocol       = tcp
8    port           = 6666
9    wait           = no
10   server         = /usr/local/lib/ad2ast/ad2ast_auth.pl
11   type           = UNLISTED
12   user           = asterisk
13   disable        = no
14 }
```

E.8. ad2ast.conf

```
1  # /etc/apache2/vhosts.d/ad2ast.conf
2  # Apache2 Konfigurationsdatei fuer den AD2Ast Virtual Host
3
4  <VirtualHost 192.168.1.18:80>
5      ServerAdmin webmaster@host.invalid
6      ServerName infma-lnxp.fh-bielefeld.de
7
8      DocumentRoot /srv/www/ad2ast/htdocs
9
10     ErrorLog /var/log/apache2/ad2ast.error_log
11     CustomLog /var/log/apache2/ad2ast.access_log combined
12
13     HostnameLookups Off
14     UseCanonicalName Off
15     ServerSignature On
16
17     ScriptAlias /cgi-bin/ "/srv/www/ad2ast/cgi-bin/"
18
19     <Directory "/srv/www/ad2ast/cgi-bin">
20         AllowOverride None
21         Options +ExecCGI -Includes
22         Order allow,deny
23         Allow from all
24     </Directory>
25
26     <IfModule mod_userdir.c>
27         UserDir public_html
28         Include /etc/apache2/mod_userdir.conf
29     </IfModule>
30
31     RedirectMatch 301 ^\/$ /cgi-bin/ad2ast_dial.pl
32
33     <Directory "/srv/www/ad2ast/htdocs">
34         Options Indexes FollowSymLinks
35         AllowOverride All
36         Order deny,allow
37     </Directory>
38
39 </VirtualHost>
```

F. Konfigurationsdateien

Nachfolgend sind alle Konfigurationsdateien des Asteriskservers aufgeführt, die gegenüber einer Standardinstallation verändert wurden.

F.1. cdr_pgsql.conf

```
1 [global]
2 hostname=/tmp
3 port=5432
4 dbname=astdb
5 password=test
6 user=asterisk
7 table=cdr
```

F.2. extensions.conf

```
1  [default]
2  exten => 2000,1,Dial(SIP/gs0)
3  exten => 2000,n,Voicemail(2000)
4  exten => 2000,n,Hangup()
5
6  exten => 2001,1,Dial(SIP/gs1)
7  exten => 2001,n,Voicemail(2001)
8  exten => 2001,n,Hangup()
9
10 exten => 5000,1,Answer()
11 exten => 5000,n,VoiceMailMain()
12 exten => 5000,n,Hangup()
13
14 exten => _5000XXXX,1,Answer()
15 exten => _5000XXXX,n,VoiceMailMain(${EXTEN:4})
16 exten => _5000XXXX,n,Hangup()
17
18 exten => _6000X,1,Answer()
19 exten => _6000X,n,MeetMe(${EXTEN:4},i)
20 exten => _6000X,n,Hangup()
21
22 exten => _0.,1,Dial(CAPI/g1,${EXTEN:1})
23 exten => _0.,n,Hangup()
24
25 exten => _X.,1,Answer()
26 exten => _X.,n,Zapateller()
27 exten => _X.,n,Festival(This number is not assigned.)
28 exten => _X.,n,Hangup()
```

F.3. manager.conf

```
1 [general]
2 enabled = yes
3
4 [ad2ast]
5 secret = test
6 deny=0.0.0.0/0.0.0.0
7 permit=127.0.0.1
8 permit=192.168.1.0/24
```

F.4. meetme.conf

```
1 [rooms]
2 conf => 0
3 conf => 1,0815
```

F.5. modules.conf

```
;
; Asterisk configuration file
;
; Module Loader configuration file
;

[modules]
autoload=yes
;
; Any modules that need to be loaded before the Asterisk core has been
; initialized (just after the logger has been initialized) can be loaded
; using 'preload'. This will frequently be needed if you wish to map all
; module configuration files into Realtime storage, since the Realtime
; driver will need to be loaded before the modules using those configuration
; files are initialized.
;
; An example of loading ODBC support would be:
;preload => res_odbc.so
;preload => res_config_odbc.so
;
; If you want, load the GTK console right away.
; Don't load the KDE console since
; it's not as sophisticated right now.
;
noload => pbx_gtkconsole.so
;load => pbx_gtkconsole.so
noload => pbx_kdeconsole.so
;
; Intercom application is obsoleted by
; chan_oss.  Don't load it.
;
noload => app_intercom.so
;
; The 'modem' channel driver and its subdrivers are
; obsolete, don't load them.
;
noload => chan_modem.so
noload => chan_modem_aopen.so
noload => chan_modem_bestdata.so
noload => chan_modem_i4l.so
;
load => res_musiconhold.so
;
; Load either OSS or ALSA, not both
; By default, load OSS only (automatically) and do not load ALSA
;
noload => chan_alsa.so
noload => chan_oss.so
;
; Module names listed in "global" section will have symbols globally
; exported to modules loaded after them.
;
noload => cdr_csv.so
noload => cdr_custom.so
noload => chan_mgcp.so
noload => chan_skinny.so
noload => pbx_dundi.so
[global]
chan_capi.so=yes
```

F.6. sip.conf

```
1  [general]
2  bind=0.0.0.0
3  port=5060
4  disallow=all
5  allow=ulaw
6  allow=alaw
7  allow=gsm
8  language=de
9
10 [gs0]
11 type=friend
12 context=default
13 deny=0.0.0.0/0
14 permit=192.168.1.230
15 username=gs0
16 secret=0002
17 mailbox=2000
18 callerid=Grandstream 0 <2000>
19 canreinvite=yes
20 host=dynamic
21
22 [gs1]
23 type=friend
24 context=default
25 deny=0.0.0.0/0
26 permit=192.168.1.231
27 username=gs1
28 secret=1002
29 mailbox=2001
30 callerid=Grandstream 1 <2001>
31 canreinvite=yes
32 host=dynamic
```

F.7. voicemail.conf

```
1 [general]
2 format=wav
3
4 [default]
5 2000 => 0002,Grandstream 0,root@localhost
6 2001 => 1002,Grandstream 1,root@localhost
```