UNIVERSITY OF APPLIED SCIENCES

FACHHOCHSCHULE GIESSEN FRIEDBERG

DIPLOMARBEIT

Fachgebiet der Diplomarbeit:
Softwareentwicklung (Software Engineering)

Thema der Diplomarbeit:
Erweiterung einer VoiceXML-Plattform und
Entwicklung einer Beispielanwendung

Diplomand:	Stefan Gräf
Referent:	Prof. Dr. Stephan Euler
Korreferent:	Prof. Dr. rer. nat. Rudolf Jaeger

Fachhochschule Gießen-Friedberg, Bereich Friedberg
Fachbereich Informationstechnik – Elektrotechnik – Mechatronik
Fachrichtung Medieninformatik, Sommersemester 2008

Hiermit versichere ich, dass ich die Diplomarbeit selbständig angefertigt und nur die in der Arbeit angegebenen Hilfsmittel und Literaturstellen verwendet habe.

Elsenfeld, den 1. August 2008

_____

Stefan Gräf

Inhaltsverzeichnis

1 Aufgabenstellung

In der heutigen Zeit kommen automatische sprachgesteuerte Informationsdienste immer häufiger zum Einsatz. Diese Dienste sind in der Regel per Telefon erreichbar und bieten dem Nutzer rund um die Uhr Auskünfte und andere Dienstleistungen an. Bekannte Beispiele für solche Systeme sind die Fahrplanauskunft der Bahn AG, das Online-Banking diverser Kreditinstitute und die Vertragsbetreuung bei Mobilfunk-Anbietern. Auch werden solche Systeme gerne bei Service-Hotlines größerer Firmen eingesetzt. Der Hauptgrund für den Einsatz solcher Systeme ist wohl die enorme Kosteneinsparung, welche durch die Reduzierung des Call Center-Personals erreicht wird.

Die Anschaffung eines sprachgesteuerten Informationsdienstes ist in der Regel mit einem gewissen finanziellen Aufwand verbunden und eignet sich daher eher für den Einsatz in größeren Firmen. Es gibt aber auch Lösungen zum Erstellen und Ausprobieren eigener Sprach-Dialoge, die sich an Privatpersonen, kleine Unternehmen und nicht-kommerzielle Organisationen richten. Ein Beispiel hierfür ist das VOICE Testcenter der Voice Community[1]. Hier können eigene Dialoge im VoiceXML-Format hochgeladen werden, die dann über spezielle Rufnummern erreichbar sind.

Jedoch haben auch diese kostengünstigeren Lösungen einige Einschränkungen, die den Einsatz als Experimentierumgebung erschweren. Der erste Nachteil ist die Inflexibilität. Man kann zwar beliebige Dialoge erstellen, aber es können keine Zusatzaufgaben eingebunden werden, wie z.B. eine Datenbankabfrage oder der Aufruf anderer Programme. Des Weiteren hat man keine Einsicht in die interne Funktionsweise. Der Nutzer erhält nur das Ergebnis. So kann u. A. der Spracherkennungs-Mechanismus nicht studiert werden und auch nicht dem Bedarf angepasst werden. Als letztes ist hier noch ein Blick auf die Kosten zu werfen, die bei Diensten wie dem VOICE Testcenter hauptsächlich durch Telefongebühren anfallen. Zur Finanzierung des Dienstes und auch zum Schutz vor Missbrauch werden meistens spezielle Service-Rufnummern verwenden, deren Gebühren etwas höher liegen als der normale Festnetz-Tarif. Durch ausgiebiges Testen der eigenen Dialoge können sich beträchtliche Kosten anhäufen.

Demnach wäre es gut, ein System zur Verfügung zu haben, welches alle diese Nachteile nicht aufweist. Der Kurs „Sprachanwendungen“ hätte somit eine ideale Experimentierumgebung und es wäre auch ein eigener Sprachdienst der Fachhochschule Friedberg realisierbar.

Das System sollte also folgende Eigenschaften aufweisen:

- Geringe Kosten
- Transparenz der internen Abläufe
- Dem Bedarf anpassbar
- Erweiterbarkeit

[1] URL: http://www.voice-community.de/

Zur Verfügung steht ein bereits vorhandenes Projekt, der Friedberger Sprachserver (FBSV). Dieser beinhaltet bereits die Bereiche Spracherkennung, Dialogverarbeitung und Sprachsynthese. Allerdings funktioniert dieser bisher nur lokal, d.h. die Spracheingabe erfolgt über ein Mikrofon und die Ausgabe über die angeschlossenen Lautsprecher. Eine einfache Möglichkeit, die Erreichbarkeit zu erweitern, ist eine Netzwerkanbindung mittels Voice-over-IP (VoIP).

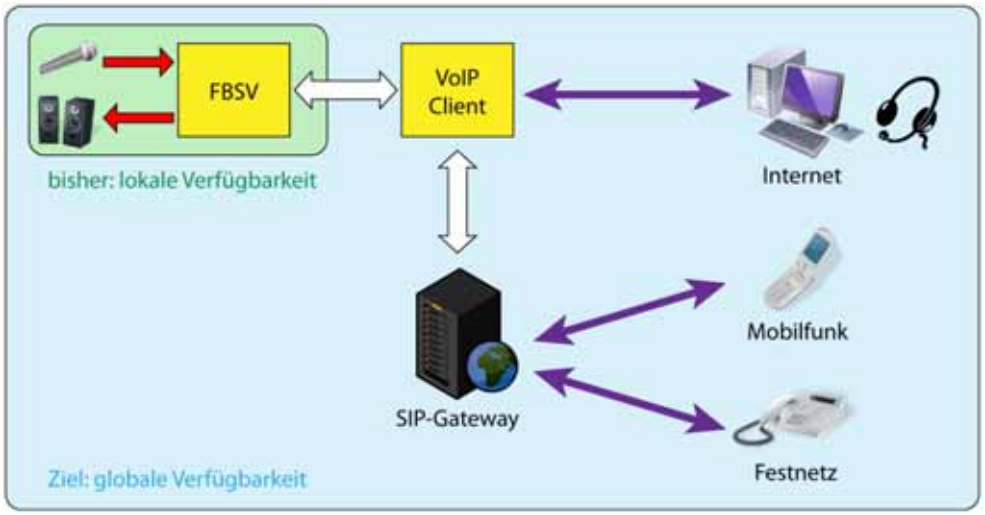


Abbildung 1: Erweiterung der Erreichbarkeit

Die Aufgabe besteht darin, den FBSV mit einer VoIP-Komponente auszustatten, so dass eine weltweite Erreichbarkeit besteht. Die Spracherkennung und die Sprachausgabe müssen an die neuen Verhältnisse, d.h. die niedrigere Qualität der Verbindung, angepasst werden.

Als zusätzliche Aufgabe ist noch ein Sprachdialog als Beispielanwendung zu erstellen: der FH Sprach-Informationsdienst. Das entstandene System soll die Machbarkeit demonstrieren und wird daher als „Demonstrator“ bezeichnet.

2 Grundlagen

2.1 Voice-over-IP

2.1.1 Einführung

Unter Voice-over-IP (VoIP) versteht man das Übertragen von Sprachsignalen in paketorientierten Netzwerken. VoIP wird auch als IP- oder Internet-Telefonie bezeichnet. IP steht für Internet Protocol und bezeichnet das Netzwerk-Protokoll, auf dem die Technologie aufsetzt.

Im Unterschied zum klassischen Telefonieren im Festnetz werden bei VoIP keine Leitungen geschaltet, sondern die Sprachdaten werden in kleine Pakete aufgeteilt, welche dann über das Netzwerk oder das Internet übertragen werden. Damit die Pakete auch beim richtigen Empfänger ankommen, wird jedes Paket für sich mit einer Adresse, der sogenannten IP-Adresse, versehen.

2.1.2 Geschichte

VoIP ist keine vollständig neue Technologie. Schon 1973 wurden erste Sprachsignale in digitaler Form über das ARPANET mittels des Network Voice Protocols übertragen [IPT08].

Durch den Boom des Internets seit 1998 und dem steigenden Leitungsdurchsatz von Internetverbindungen hielt VoIP langsam Einzug in den Massenmarkt.

Im Jahr 2004 erscheint die populäre Software Skype. Wegen der mittlerweile starken Verbreitung von DSL-Anschlüssen in Deutschland wird die Technologie auch hierzulande interessant.

Bald darauf gibt es viele Angebote großer Internet-Provider, die bereits VoIP-Flatrates enthalten. Zudem werden bei Abschlüssen von DSL-Verträgen oft spezielle Router mitgeliefert, welche die Nutzung von VoIP ohne spezielle Hardware über herkömmliche Telefone erlauben.

Abbildung 2: Fritz!Box Fon von AVM. Beispiel für ein DSL-Router mit VoIP-Unterstützung

2.1.3 Funktionsprinzip

Anhand des folgenden Diagramms soll nun ein grober Überblick zur Funktionsweise von VoIP gegeben werden [Krö081]:

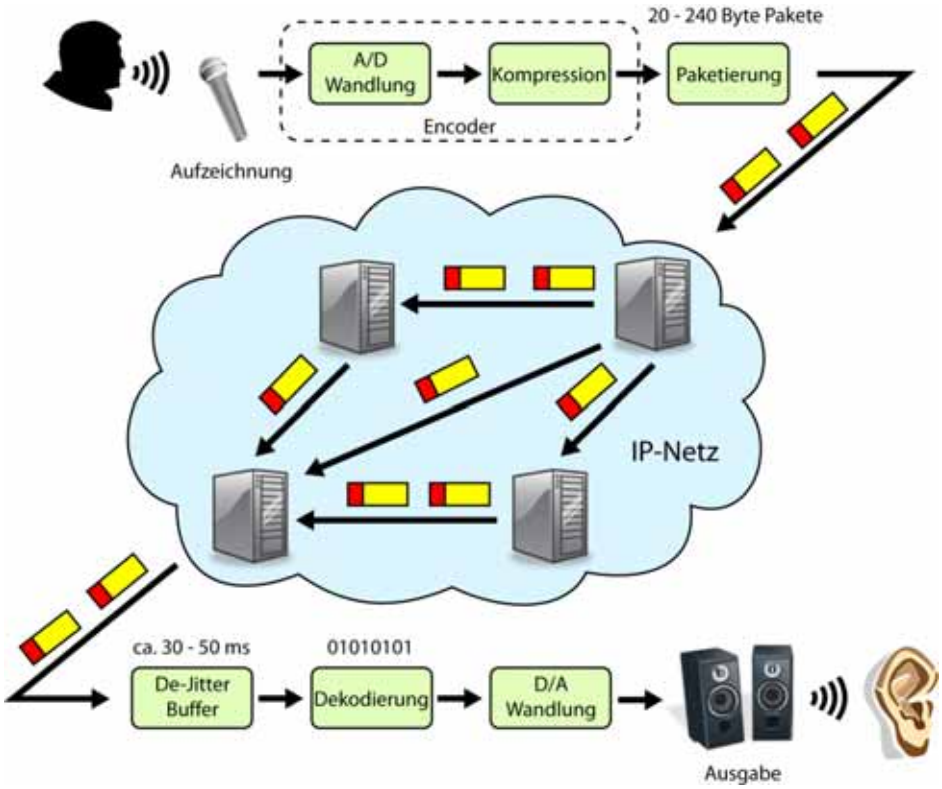


Abbildung 3: Funktionsweise von VoIP

Die Übertragung des Sprachsignals lässt sich in neun Schritte unterteilen:

1. **Aufzeichnung**
   Im ersten Schritt wird das Sprachsignal mit Hilfe eines Mikrofons in elektrische Signale umgewandelt.
2. **Analog/digital-Wandlung**
   Das elektrische Signal vom Mikrofon muss für die weitere Verarbeitung in eine digitale Form gebracht werden. Dies geschieht mit Hilfe eines Analog/digital-Wandlers.
3. **Kompression**
   Die digitalen Daten werden nun komprimiert, um den Bandbreitenbedarf zu senken. Je nach verwendetem Codec kann die Qualität dadurch mehr oder weniger stark beeinträchtigt werden. Im besten Fall ist der Qualitätsverlust subjektiv nicht wahrnehmbar.
4. **Paketierung**
   Die Sprachdaten stellen bisher einen kontinuierlichen Datenstrom dar. Dieser wird in kleinere Einheiten (Pakete) aufgeteilt. Dies ist nötig, damit die Daten über das paketorientierte Netz übertragen werden können. Vor dem Senden eines Pakets muss auf eine ausreichende Menge an Daten gewartet werden. Jedes Paket wird schließlich noch mit einer Zieladresse versehen.
5. **Übertragung**
   Die Pakete werden nun ins Netz gesendet. Dort werden sie von Routern eingesammelt, die die

Adresse in den Paket-Headern auswerten und die Pakete weiter in Richtung Ziel schicken. Üblicherweise muss ein Paket viele Router passieren bevor es den Bestimmungsort erreicht. Welchen Weg ein Paket letztendlich zurücklegt ist nicht festgelegt. Die Router können jedes einzelne Paket auf einen anderen Weg schicken, z.B. um die Netzauslastung zu verteilen.

6. **Pufferung**
   Die Pakete werden auf der Empfänger-Seite zunächst in einem Buffer zwischengespeichert. Dadurch wird das sogenannte Jitter kompensiert (eine Erklärung erfolgt weiter unten im Text). Sind genügend Pakete eingegangen, so werden die Nutzdaten weiter zur Dekompression geleitet. Es entsteht wieder ein kontinuierlicher Datenstrom.
7. **Dekompression**
   Die Komprimierung der Daten wird nun rückgängig gemacht. Die Daten liegen danach fast genau so vor, wie vor der Kompression im Sender. Allerdings entsteht ein Qualitätsverlust, da die meisten Codecs verlustbehaftet sind.
8. **Digital/Analog-Wandlung**
   Im vorletzten Schritt werden die digitalen Daten mittels Digital/Analog-Wandlers wieder in die analoge Form gebracht.
9. **Ausgabe**
   Schließlich wird das analoge elektrische Signal durch einen Lautsprecher oder Kopfhörer in Luftdruck-Schwankungen umgesetzt, was auch als Schall bekannt ist. Dieser kann nun vom Gesprächsteilnehmer wahrgenommen werden.

2.1.4 Qualität

2.1.4.1 Einleitung

Die Sprachqualität einer VoIP-Verbindung wird von verschiedenen Faktoren beeinflusst. Der wichtigste Faktor ist der verwendete Codec. „Codec“ ist ein Kunstwort und setzt sich aus den englischen Worten „coder“ und „decoder“ zusammen [Wik082]. Ein Codec ist eine Vorschrift, wie Daten kodiert und auch wieder dekodiert werden müssen. Für eine erfolgreiche Verbindung müssen der Sender und der Empfänger denselben Codec anwenden. In Tabelle 1 sind einige der bei VoIP gebräuchlichen Codecs aufgelistet [Aud08] [OzV08].

Codec	Name/Bezeichnung	Übertragungsrate	MOS	Delay	Sprachqualität
G.711	Pulse Code Modulation (PCM)	56 oder 64 kbit/s	4,4	0,25 ms	ISDN
G.726	Adaptive Differential Pulse Code Modulation (ADPCM)	16-40 kbit/s	4,2		Mobilfunk
G.728	Low Delay Code Excited Linear Prediction (LD-CELP)	16 kbit/s	4,2	1,25 ms	ungefähr ISDN
G.729/ G.729A	Conjugate Structure Algebraic Code Excited Linear Prediction (CS-ACELP)	8 kbit/s	3,98/ 3,7	25 ms	besser als G.723.1
G.723.1	Multiple Maximum Likelihood Quantization (MPMLQ)	6,3 kbit/s	3,9	67,5 ms	Gut
G.723	Algebraic Code Excited Linear Prediction (ACELP)	5,3 kbit/s	3,5		

Tabelle 1: VoIP Codecs (unvollständig)

2.1.4.2 Mean Opinion Score (MOS)

Die Sprachqualität eines Codes wird durch den Mean Opinion Score (MOS, engl. für durchschnittliche Beurteilung) ausgedrückt. Um den MOS eines Codecs zu bestimmen, hört sich eine Gruppe von ausgewählten Personen Gespräche an, die mit diesem Codec durchgeführt wurden. Die Probanden bewerten anschließend die subjektiv empfundene Qualität mit Punkten. Die Punkte reichen von 1 für „schlechteste Qualität“ bis 5 für „exzellent“. Die Mittelung aller vergebenen Punkte ergibt den MOS-Wert [Wik083].

Besitzt der Codec einen MOS größer als 4, so kann man die Qualität mit der eines Festnetz-Gesprächs vergleichen. Bei Werten kleiner als 4 ist die Qualität eher mit der eines Mobilfunk-Gesprächs zu vergleichen [Aud08]. In der Regel benötigen Codecs mit einer höheren Sprachqualität auch mehr Bandbreite. Die zur Verfügung stehende Bandbreite ist entscheidend für die Wahl des verwendeten Codecs.

2.1.4.3 Codecs im Detail

2.1.4.3.1 G. 711

Dieser Codec wird auch für die ISDN-Übertragung im Festnetz verwendet und hat gegenüber den meisten anderen Codecs die höchste Qualität. Die Auflösung der Audio-Daten wird mittels einer nicht-linearen Bewertungskurve (a-LAW oder u-LAW) von 16 Bit auf 8 Bit reduziert. Eine weitere Kompression erfolgt nicht. Weitere Vorteile neben der höheren Qualität sind die geringe Verzögerung beim Kodieren und die Möglichkeit, die Daten direkt ins Festnetz einzuspeisen, ohne dass ein Umcodieren nötig wäre.

Der große Nachteil dieses Codecs ist der hohe Bandbreitenbedarf. Durch Packet-Overhead und Signalisierungsdaten erhöht sich die Netto-Bandbreite von 64 kBit/s pro Richtung auf etwa 80 bis 100 bKit/s. Dieser Codec ist für Internet-Verbindungen mit niedriger Bandbreite (z. B. Modem, ISDN) ungeeignet [Aud08] [Wik084].

2.1.4.3.2 G.729

In den meisten Fällen wird bei VoIP der Codec G.729 eingesetzt. Dabei wird das Sprachsignal in Parameter zerlegt und übertragen. Man spricht dabei von einem sogenannten Vocoder. Zusätzlich wird noch die Differenz zwischen dem eigentlichen Sprachsignal und dem künstlichen, aus den Parametern rekonstruierten Signal, übertragen. Auf der Gegenseite wird das Sprachsignal mittels Sprachsynthese und des Differenzsignals wieder zusammengesetzt. Durch dieses Verfahren kann bei einer niedrigen Bandbreite von üblicherweise 8 kbit/s noch eine gute Gesprächsqualität erzielt werden. Der MOS beträgt bei diesem Codec 3,98.

Die Nachteile dieses Codecs sind die relativ große Kodierungsverzögerung von 25 ms und der hohe Rechenaufwand von etwa 50 MIPS[2]. Es gibt auch eine vereinfachte Variante dieses Codecs mit der Bezeichnung G.729.A. Bei dieser beträgt der Rechenaufwand nur etwa 10,5 MIPS, allerdings ist die Übertragungsqualität mit einem MOS von 3,7 etwas niedriger [Aud08] [Wik07].

[2] MIPS: million instructions per second

2.1.5 Probleme bei der Übertragung

2.1.5.1 Jitter

Unter dem Begriff Jitter versteht man die Varianz der Zeit, welche die einzelnen Pakete benötigen, um vom Sender zum Empfänger zu gelangen. Die Ursachen für den Unterschied in der Durchlaufzeit sind hauptsächlich der nicht fest definierte Weg durch das Netz und das sich stets verändernde Verkehrsaufkommen. Um den Jitter zu kompensieren, wird ein Empfangspuffer benötigt. Dieser Puffer hat jedoch den Nachteil, dass sich die Latenz weiter erhöht [IPT08] [Wik085].

2.1.5.2 Latenz

Die Verzögerung zwischen der Audioeingabe beim Sender und der Audioausgabe beim Empfänger wird als Latenz bezeichnet. Bleibt die Latenz unter 150ms, so wird diese nicht bewusst wahrgenommen. Eine Latenz von über 250ms hat bereits einen negativen Einfluss auf das Gespräch. Bei einer Latenz von bis zu 400ms gilt das Gespräch noch als akzeptabel, liegt der Wert aber darüber, so führt das zu unangenehmen Verzögerungen und die Gesprächspartner fallen sich oft gegenseitig ins Wort. Der Latenzwert setzt sich aus der Leitungsdurchlaufgeschwindigkeit, der Kodierungsverzögerung und der Verweildauer im Empfangspuffer zusammen [IPT08] [VoI08].

2.1.5.3 Packet Drops

Das zurzeit verwendete Internet Protocol IPv4 basiert auf dem Prinzip „Best Effort“ (engl. für größte Bemühung), d. h. es gibt keine Zusicherung, dass Datenpakete auch tatsächlich ihr Ziel erreichen. Geht ein Datenpaket unterwegs verloren, so spricht man von einem „Packet drop“ (to drop: engl. für fallen lassen). Diese können z.B. durch Kollisionen[3] versursacht werden oder aber Router können Pakete auch absichtlich verwerfen, falls keine weiteren Kapazitäten mehr zur Verfügung stehen. Bei nicht-zeitkritischen Anwendungen wie etwa beim Dateitransfer werden höhere Protokolle eingesetzt, die Paketverluste feststellen und verlorene Pakete nochmals übertragen. Bei Echtzeitanwendungen wie VoIP ist die Verzögerung durch Neutransmissionen nicht tragbar und deshalb wird hier eine zusätzliche Sicherung der Verbindung nicht angewendet. Je nach eingesetztem Codec verursachen Paketverluste Aussetzer, Rauschen oder Störgeräusche im Audiosignal [IPT08] [VoI08].

2.1.6 Verbesserungsmöglichkeiten

2.1.6.1 Priorisierung

An dieser Stelle werden nun zwei Möglichkeiten vorgestellt, um Problemen bei der Übertragung entgegenzuwirken. Die erste Möglichkeit ist die Priorisierung. Dabei werden die IP-Datenpakete mit speziellen Flags[4] versehen, die von Routern ausgewertet werden können. Dazu gibt es im Header des IP-Paketes das Feld Type of Service (TOS). Dort können drei verschiedene Flags gesetzt werden, die für eine geringe Verzögerung, einen hohen Durchsatz oder eine hohe Zuverlässigkeit stehen.

[3] Kollisionen entstehen, wenn zwei oder mehrere Netzteilnehmer gleichzeitig Daten auf ein Medium mit geteiltem Zugriff übertragen wollen.
[4] ein Flag ist üblicherweise eine Variable, die nur zwei Zustände kennt: 0 (nicht gesetzt) und 1 (gesetzt)

Wird z. B. der Flag für eine geringe Verzögerung gesetzt, so können diese Pakete andere Pakete in der Warteschlange des Routers überspringen (sofern der Router dies unterstützt) und somit schneller ans Ziel gelangen. Treten bei der Übertragung viele Packet Drops auf, so hilft hier evtl. das Setzen des Flags für eine hohe Zuverlässigkeit [Wik086].

2.1.6.2 Traffic Shaping

Die zweite Möglichkeit wird als Traffic Shaping[5] oder IntServ (Integrated Services) bezeichnet. Dabei wird die zur Verfügung stehende Bandbreite auf spezielle Dienste oder Endgeräte aufgeteilt. Dadurch kann z.B. eine gewisse Menge an Bandbreite für VoIP reserviert werden. Das Shaping funktioniert nur für ausgehende Daten und wird meistens vom Router oder Gateway mit der Verbindung zum Internet durchgeführt [Wik071] [Wik087].

2.1.7 Vor- und Nachteile

VoIP hat gegenüber der klassischen Festnetztelefonie einige Vorteile, dennoch gibt es aber auch Nachteile. In Tabelle 2 und 3 sind die wichtigsten Vor- und Nachteile zusammengefasst [Nöl06] [Meß07].

Vorteile	
Geringere / keine Gebühren	In den meisten Fällen sind Gespräche unter VoIP-Teilnehmern kostenlos. Nur bei Gesprächen in oder aus dem Festnetz fallen Gebühren an, diese liegen jedoch üblicherweise unter dem normalen Festnetz-Tarif. Besonders Gespräche ins Ausland sind mit VoIP erheblich günstiger.
Keine Telefonleitungen benötigt	Es wird bei VoIP nur eine Verbindung zum Internet benötigt. Das spart auf der einen Seite die Grundgebühr, die für die Telefonleitung anfallen würde, auf der anderen Seite müssen keine zusätzlichen Leitungen verlegt werden, wenn bereits ein Netzwerk vorhanden ist, wie z.B. in größeren Firmen.
Mehrere Gespräche über eine Leitung	Bei analogen Festnetzanschlüssen ist nur ein zeitgleiches Gespräch möglich, bei ISDN immerhin zwei. Die Anzahl der zeitgleichen VoIP-Gespräche hängt von der verfügbaren Bandbreite ab. Bei der heute üblichen DSL-Geschwindigkeit von 6000 kbit/s downstream und 512 kbit/s upstream ergäbe das für den Codec G.729 (8 kbit/s + Overhead + Reserve = 12 kbit/s) ca. 42 mögliche Gespräche zur selben Zeit (512 kbit/s : 12 kbit/s = 42,7).
Videoübertragung möglich	Es ist zwar auch Video-Telefonie über ISDN möglich, aber diese hat einige gewaltige Nachteile, so dass sie sich nie im Massenmarkt durchgesetzt hat, wie z.B. teure Endgeräte, die beide Teilnehmer besitzen mussten oder doppelte Gebühren durch die notwendige Kanalbündelung. In der Regel bieten heutige Internetanschlüsse genügend Bandbreite für eine problemlose Videoübertragung. Ein vorhandener Rechner muss nur um eine Webcam erweitert werden, die bereits für wenige Euro erhältlich ist. Viele Notebooks haben aber auch bereits eine Webcam integriert.

Tabelle 2: Vorteile von VoIP gegenüber klassischer Telefonie

[5] engl.: den Verkehr in Form bringen

Nachteile	
Teils noch schlechtere Sprachqualität	Durch die oben genannten Probleme, die die Übertragungsqualität beeinflussen, und die starke Kompression mancher Codecs liegt die Sprachqualität in vielen Fällen noch etwas unter der Qualität von Festnetzgesprächen.
Kompliziertere Technik	Das Einrichten und Konfigurieren eines SIP-Accounts und die Verwendung von VoIP-Software kann für Personen, die im Umgang mit Computern nicht versiert sind, erhebliche Probleme bereiten. Abhilfe schafft hier spezielle Hardware, die Gespräche über normale Festnetztelefone zulässt, so dass für den Endanwender kein zusätzlicher Aufwand entsteht. Es wird jedoch eine Person (Administrator) benötigt, die diese Hardware einrichtet und wartet.
Höhere Latenzzeiten	Auch bei einer optimalen Verbindung dauert es im Vergleich zum Festnetz etwas länger, bis das Sprachsignal beim Empfänger angekommen ist.
Keine reservierter Kanal	Es gibt bei VoIP keinen reservierten Kanal wie bei der Festnetztelefonie. Die vorhandene Bandbreite muss mit anderen Netzanwendungen geteilt werden. Werden z.B. Downloads durchgeführt oder Datei-Tauschprogramme betrieben, kann es sein, dass für VoIP nicht genügend Bandbreite mehr zur Verfügung steht und Gespräche unverständlich werden oder auch komplett zusammenbrechen.

Tabelle 3: Nachteile von VoIP gegenüber klassischer Telefonie

2.2 Session Initiation Protocol (SIP)

2.2.1 Einführung

Das Session Initiation Protocol (SIP) ist eines von vielen möglichen Signalisierungsprotokollen für VoIP. Momentan ist es das am weitesten verbreiteteste Protokoll. Seine Aufgabe ist der Aufbau, die Steuerung und der Abbau einer Kommunikationssitzung zwischen zwei oder mehreren Teilnehmern. Der Einsatz beschränkt sich nicht nur auf VoIP. „Sessions“ können beliebige Multimediaströme, Konferenzen, Computerspiele usw. sein.

SIP ist ein reines Signalisierungsprotokoll, d.h. es wird nur sichergestellt, dass eine Verbindung zum richtigen Ziel aufgebaut wird. Für den weiteren Kommunikationsverlauf ist SIP dementsprechend nicht mehr zuständig.

Das Protokoll wurde von der IETF (Internet Engineering Task Force) entwickelt und wurde 1999 erstmals in RFC 2543 vorgestellt. Im Jahr 2004 wurde mit RFC 3261 eine neuere Version veröffentlicht [Ses08].

2.2.2 SIP im Detail

2.2.2.1 Einleitung

Das SIP besitzt eine standardisierte Protokollstruktur, die in ihrem Aufbau stark an das Hypertext Transfer Protocol (HTTP) angelehnt ist, welches zur Übertragung von Webseiten verwendet wird. Es ist zu diesem jedoch nicht kompatibel.

2.2.2.2 Der Header

Die Header-Struktur des SIP ist der des HTTP sehr ähnlich und sie ist ebenfalls textbasiert. In Tabelle 4 ist der Header einer SIP Verbindungsanfrage zu sehen. Die Bedeutung der einzelnen Felder wird in Tabelle 5 erklärt [Krö08].

Status	INVITE sip:49180123456@217.0.0.1:5060 SIP/2.0
Header	Via: SIP/2.0/UDP 217.10.79.9:5060 From: "060311234" <sip:060311234@sipgate.de> To: <sip:0049180123456@sipgate.de> Contact: <sip:060311234@217.10.67.11> Call-ID: 647acd983e80f76ec6d@sipgate.de CSeq: 102 INVITE Max-Forwards: 67 Supported: replaces Content-Type: application/sdp Content-Length: 410
Leerzeile	
Body	v=0 o=root 14520 14520 IN IP4 217.10.67.11 s=session c=IN IP4 217.10.67.11 t=0 0 m=audio 11286 RTP/AVP 8 0 3 97 18 112 101 a=rtpmap:8 PCMA/8000 a=rtpmap:0 PCMU/8000 a=rtpmap:3 GSM/8000

Tabelle 4: Beispiel für einen SIP Header

Feld	Beschreibung
Status	Hier wird festgelegt, um was für eine Art von SIP-Nachricht es sich handelt. INVITE steht für eine Verbindungsanfrage (eingehender Anruf).
Via	Gibt an, wie der Sender der Nachricht erreicht werden kann. Dazu werden die Informationen Protokoll, IP-Adresse und Port-Nummer benötigt.
From	SIP-Adresse des Absenders
To	SIP-Adresse des Empfängers
Contact	Unter dieser SIP-Adresse kann eine direkte Verbindung zum Absender hergestellt werden.
Call-ID	Eine Zufallsnummer, die das Gespräch eindeutig kennzeichnet.
CSeq	Legt die Sequenz-Nummer fest. Der Empfänger muss mit derselben Nummer antworten.
Max-Forwards	Eine Zahl, die festlegt, wie oft diese Nachricht von SIP-Servern weitergeleitet werden darf. Bei jeder Weiterleitung wird die Zahl um eins verringert. Beim Erreichen des Wertes Null wird die Nachricht vernichtet.
Content-Type	Dieses Feld spezifiziert, um welche Art von Nutzdaten es sich im Body der Nachricht handelt. Hier sind es Daten vom Typ SDP (Session Description Protocol). Die Aufgabe vom SDP ist der Austausch von Informationen über die zu verwendenden Multimedia-Ströme.
Content-Length	Größe der Nutzdaten in Bytes.

Tabelle 5: Bedeutung der einzelnen Felder im SIP Header

2.2.2.3 SIP-Adressen

Die Teilnehmeradressen werden bei SIP im sogenannten URI (Uniform Resource Identifier)-Format angegeben, welches u. A. auch bei E-Mail-Adressen verwendet wird: „sip:user@domain“. Der Teil „sip:“ gibt an, dass es sich dabei um eine SIP-Adresse handelt. An der Stelle „user“ steht eine eindeutige Kennung des Teilnehmers, üblicherweise eine Nummer, unter der dieser Teilnehmer auch über das Festnetz erreichbar ist. Beispiel: sip:0049180123456@sipgate.de. Hier wäre der Teilnehmer unter der Nummer +49 (0) 180 123 456 im Festnetz erreichbar. Der letzte Teil „domain“ kennzeichnet das Netzwerk des SIP-Anbieters.

Eine Alternative zu der SIP-Adresse ist der tel-URI. Die SIP-Adresse aus obigem Beispiel lautet im tel-URI-Format: tel:+49-180-123456 [Ses08].

2.2.2.4 Verbindungsaufbau

2.2.2.4.1 Einleitung

Eine Verbindung kann bei SIP auf zwei verschiedene Arten aufgebaut werden. Die erste Methode wird als Proxy-Mode bezeichnet und die zweite als Redirect Mode. Im Folgenden werden die Unterschiede zwischen beiden Methoden genauer dargestellt [Krö08].

2.2.2.4.2 Proxy-Mode

Bei dieser Methode spielt der Proxy-Server die entscheidende Rolle beim Verbindungsaufbau. Er verarbeitet dabei die Anrufanfrage und ist auch für die anschließende Anrufvermittlung zuständig. Das folgende Diagramm zeigt die einzelnen Schritte zur Herstellung der Verbindung im Proxy-Modus:

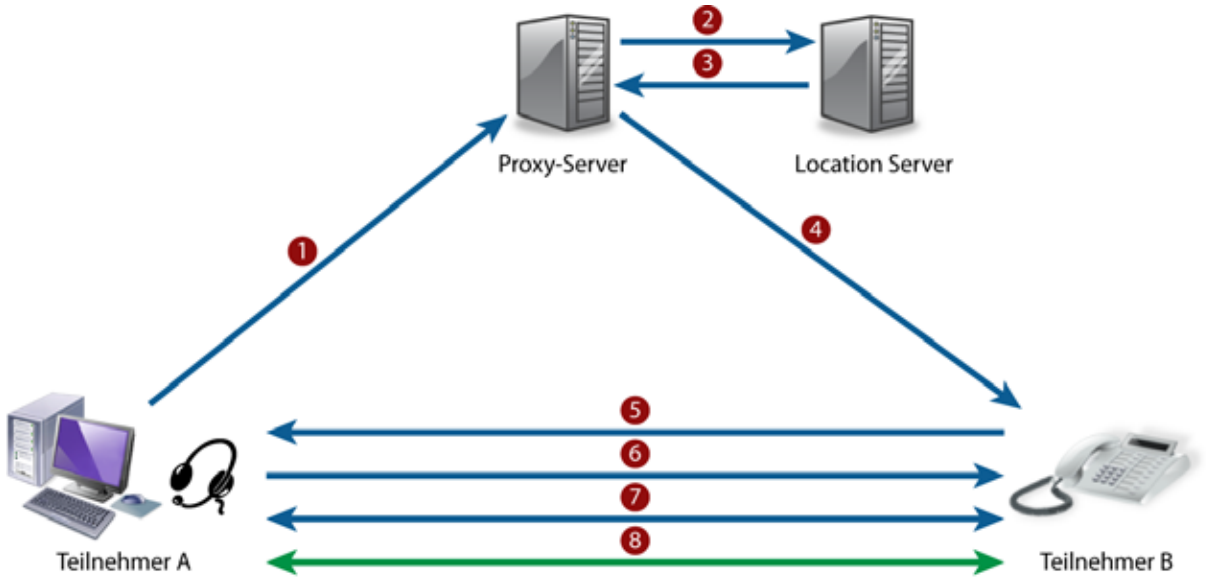


Abbildung 4: Verbindungsaufbau bei SIP im Proxy-Mode

1. Teilnehmer A sendet mittels der SIP-Rufnummer einen INVITE-Request an den Proxy-Server, der für Teilnehmer B zuständig ist.
2. Der Proxy-Server leitet die Anfrage zur Lokalisierung des Teilnehmers B an den Location-Server weiter.

3. Der Location-Server sucht in seiner Datenbank einen passenden Datensatz zu der Rufnummer des Ziels. Ist die Suche erfolgreich, so wird der gefundene Datensatz an den Proxy-Server zurückgegeben.
4. Die Verbindungsanfrage wird vom Proxy-Server an den Teilnehmer B weitergeleitet. Dieser kann nun entscheiden, ob er die Verbindung annimmt oder nicht.
5. Akzeptiert Teilnehmer B die Verbindungsanfrage, so sendet er die entsprechende Signalisierung mittels des Proxy-Servers an Teilnehmer A.
6. Teilnehmer A bestätigt die Signalisierung von Teilnehmer B.
7. Die Verbindung zwischen Teilnehmer A und B ist aufgebaut.
8. Datenaustausch über die Medienströme

2.2.2.4.3 Redirect Mode

Wenn der Verbindungsaufbau im Redirect-Modus durchgeführt wird, dann ist der Redirect-Server die zentrale Verwaltungsstelle. Er ist dafür zuständig, dass die Zieldaten an den Anrufer übermittelt werden. Im Redirect-Modus sieht der Ablauf folgendermaßen aus:

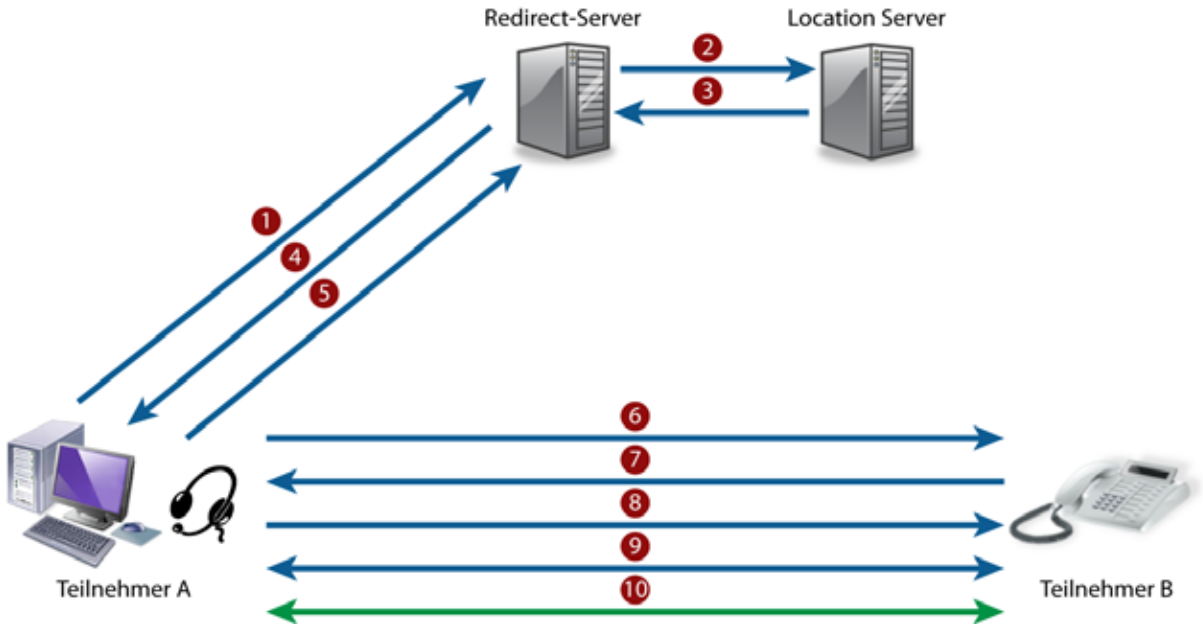


Abbildung 5: Verbindungsaufbau bei SIP im Redirect-Mode

1. Teilnehmer A sendet mittels der SIP-Rufnummer einen INVITE-Request an den Redirect-Server, der für Teilnehmer B zuständig ist.
2. Der Redirect-Server leitet die Anfrage zur Lokalisierung des Teilnehmers B an den Location-Server weiter.
3. Der Location-Server sucht in seiner Datenbank einen passenden Datensatz zu der Rufnummer des Ziels. Ist die Suche erfolgreich, so wird der gefundene Datensatz an den Redirect-Server zurückgegeben.
4. Im Gegensatz zur Proxy-Mode wird die Verbindungsanfrage nun nicht vom Redirect-Server an Teilnehmer B weitergeleitet, sondern Teilnehmer A erhält alle benötigten Informationen, um eine direkte Verbindung zu Teilnehmer B herzustellen.
5. Teilnehmer A sendet eine Bestätigung über den Erhalt der Zieldaten an den Redirect-Server. Dieser wird ab jetzt für den weiteren Verlauf der Kommunikation nicht mehr benötigt.

6. Teilnehmer A sendet eine direkte Verbindungsanfrage an Teilnehmer B. Dieser hat wieder die Möglichkeit, diese anzunehmen oder abzulehnen.
7. Akzeptiert Teilnehmer B die Verbindungsanfrage, so sendet er die entsprechende Signalisierung direkt an Teilnehmer A.
8. Teilnehmer A bestätigt die Signalisierung von Teilnehmer B.
9. Die Verbindung zwischen Teilnehmer A und B ist aufgebaut.
10. Datenaustausch über die Medienströme

2.3 Spracherkennung

2.3.1 Einleitung

Die Spracherkennung ist ein Teilgebiet der angewandten Informatik. Dabei werden Verfahren untersucht und entwickelt, mit denen Automaten (Computern) die gesprochene Sprache zur automatischen Datenerfassung zugänglich gemacht werden kann. Wichtig ist es, die Spracherkennung von Sprechererkennung zu unterscheiden. Bei letzterem handelt es sich um ein biometrisches Verfahren zur Personenidentifikation [Spr08].

2.3.2 Geschichtliche Entwicklung

Die ersten Forschungen auf dem Gebiet der Spracherkennung wurden in den 60er Jahren unternommen, blieben jedoch überwiegend erfolglos. Die Erkennung funktionierte maximal mit einigen hundert Einzelworten. Eine Ursache hierfür war das zu der Zeit noch sehr begrenzte Wissen auf diesem Gebiet, es fehlte u. A. an effektiven Algorithmen. Weiterhin waren die technischen Möglichkeiten bezüglich der Rechenleistung noch sehr begrenzt.

Erste Mitte 1980 kam die Entwicklung weiter voran und setzte sich danach rasant fort. Die Firma IBM beschäftigte sich dabei in besonderem Maße mit der Entwicklung und Vermarktung von Spracherkennungssystemen.

Die einzelnen Meilensteine der Spracherkennung sind in Tabelle 6 aufgeführt [Hen01].

Jahr	Ereignis
1984	IBM stellt ein Spracherkennungssystem vor, das mit Hilfe eines Großrechners ca. 5000 Einzelworte in einem mehrere Minuten dauerndem Vorgang erkennen konnte. Die 1982 von ehemaligen IBM-Mitarbeitern gegründete Firma Dragon Systems bringt mit dem Dragon Dictate System das erste Spracherkennungssystem für tragbare Computer auf den Markt.
1986	IBM präsentiert das System TANGORA 4 für Englisch (benannt nach dem Weltmeister im Maschineschreiben). Das System arbeitete in Echtzeit auf einem normalen Arbeitsplatzrechner. Dies war möglich durch Spezialprozessoren. Durch Kontextprüfung mittels Trigrammstatistik[6] konnten auch Homophone[7] unterschieden werden.
ab 1988	Entwicklung einer deutschen Version von TANGORA. Sie fand Einsatz in Medizin, Gerichtswesen und Geschäftskorrespondenz. Jedes Einsatzgebiet hatte ein eigenes Vokabular von etwa 20-30 tausend Worten (jede Wortform als eigenes Wort gezählt).
1991	Auf der CeBIT wird TANGORA 4 deutsch vorgestellt. Die Präsentation musste in einem abgeschirmten Raum stattfinden, um das System vor dem Messelärm zu schützen.
1992	Die TANGORA-Technik ist ein einem Produkt verfügbar: der ISSS (IBM Speech Server Series). Dabei handelte es sich um eine Client/Server Lösung mit einem IBM RS/6000-Server. Die akustische Eingabe konnte z.B. auf einem PC mit OS/2 erfolgen. Einsatz fand dieses System besonders in Krankenhäusern und Kliniken.
1993	Ankündigung eines neuen auf TANGORA-Technologie basierenden Produktes: IBM Personal Dictate System (später in IBM VoiceType umbenannt). Der Preis betrug weniger als 1000$ (vorher haben Spracherkennungssysteme meistens mehr als 10000$ gekostet). Es war als reine PC-Lösung ein Produkt für den Massenmarkt
1994	Vorstellung der englischen Version von IBM VoiceType auf der CeBIT, wenige Wochen später auch als deutsche Version. Die 1990 gegründete Philips Tochter Philips Dictation Systems stellt die erste Lösung für die Erkennung von kontinuierlicher Sprache vor.
1995	Vorstellung von IBM VoiceType mit speziellem Fachvokabular für Mediziner und Anwälte auf der CeBIT. Die Präsentation war sogar in lautester Messeatmosphäre möglich.
1996	Vorstellung von IBM VoiceType Diktiersystem 3.0 für Windows 95 auf der CeBIT. Es wurde keine IBM-Audiokarte mehr benötigt und das System lief auch auf Windows 95.
1997	Philips Dictation Systems bringt SpeechMagic auf den Markt, eine Client/Server fähige Spracherkennungslösung. Die 1987 gegründete Firma Learnout & Hauspie stellt ihr erstes Spracherkennungsprodukt vor.
1998	Mit FreeSpeech98 bringt Philips Dictation Systems ein Konsumentenprodukt für den Massenmarkt heraus. Spracherkennungsprodukte von Learnout & Hauspie sind nun auch in deutscher Sprache verfügbar.

Tabelle 6: Zeitlicher Abriss der Entwicklungen auf dem Gebiet der Spracherkennung

[6] Ein Trigramm ist eine Wortkette bestehend aus drei Wörtern. Durch Untersuchung von Texten kann eine Liste ermittelt werden, die die Wahrscheinlichkeiten für das Auftreten verschiedener Trigramme enthält.

[7] Wörter mit einer ähnlichen oder identischen Aussprache, aber unterschiedlicher Bedeutung (z.B. rein und Rhein)

2.3.3 Anwendungsgebiete

Die wichtigsten Einsatzmöglichkeiten für Spracherkennung sind in Tabelle 7 dargestellt.

Gebiet	Beschreibung	Beispiele
Command & Control	Gerätesteuerung mit Hilfe von Spracheingaben	• Handy-Sprachwahl • Bedienung von Unterhaltungselektronik • PC-Steuerung
Sprachdialoge	Automatisierte Dialoge zwischen Mensch und Maschine, häufig über eine Telefonverbindung	• Telefon-Banking • Fahrplanauskunft • Vertragsbetreuung beim Mobilfunk
Texterfassung	Gesprochene Wörter werden in geschriebenen Text umgewandelt	• Diktier-Systeme • Digitaler Dolmetscher

Tabelle 7: Anwendungsgebiete Spracherkennung

Die Anforderungen an das Spracherkennungssystem sind je nach Anwendung unterschiedlich. Grundsätzlich unterscheidet man nach folgenden Kriterien [Lin08]:

Sprecherabhängigkeit

- Sprecherabhängig
  Das System kann erst nach einer Trainingsphase von einem Benutzer verwendet werden. Jeder Benutzer muss das System separat für die eigene Stimme anlernen. Typisches Anwendungsgebiet: Command & Control-Systeme

- Sprecherunabhängig
  Hier kann ein beliebiger Benutzer das Spracherkennungssystem verwenden ohne vorher das System auf die eigene Stimme zu trainieren. Typisches Anwendungsgebiet: Sprachdialoge

Einzelworterkennung vs. kontinuierliche Sprache

- Einzelworterkennung
  Die Wörter müssen mit deutlichen Pausen dazwischen gesprochen werden. Jedes Wort wird nur für sich alleine betrachtet, der Kontext wird nicht berücksichtigt. Typisches Anwendungsgebiet: Command & Control-Systeme, Sprachdialoge

- Kontinuierliche Sprache
  Wie bei einem Gespräch mit einer anderen Person kann hier ganz normal in kompletten Sätzen gesprochen werden. Der Kontext wird berücksichtigt und hilft bei der korrekten Erkennung. Typisches Anwendungsgebiet: Texterfassung

Größe des Wortschatzes

- Kleiner Wortschatz
  Es kann nur eine geringe Anzahl verschiedener Worte (bis ca. 1000) erkannt werden. Typisches Anwendungsgebiet: Sprachdialoge, Command & Control-Systeme

- Großer Wortschatz
  Bei einem Wortschatz von über 1000 Wörtern spricht man von einem großen Wortschatz. Moderne Systeme beherrschen (ausgenommen sehr selten verwendete Worte) den kompletten Wortschatz einer Sprache (bis über 100.000 Worte). Typisches Anwendungsgebiet: Texterfassung

2.3.4 Funktionsweise

2.3.4.1 Einleitung

Bei der Durchführung der Spracherkennung gibt es verschiedene Vorgehensweisen, bei denen unterschiedliche Algorithmen zum Einsatz kommen. In diesem Kapitel wird eine Vorgehensweise dargestellt, wie sie z. B. häufig bei Texterfassungssystemen vorkommt. Größtenteils wird bei der Spracherkennung, welche später in diesem Projekt zum Einsatz kommt, auf dieselbe Weise verfahren.

Die folgende Darstellung orientiert sich dabei im Wesentlichen an einem Artikel aus der Zeitschrift c't [Hab98]. Abbildung 6 gibt einen Überblick über die durchzuführenden Schritte.

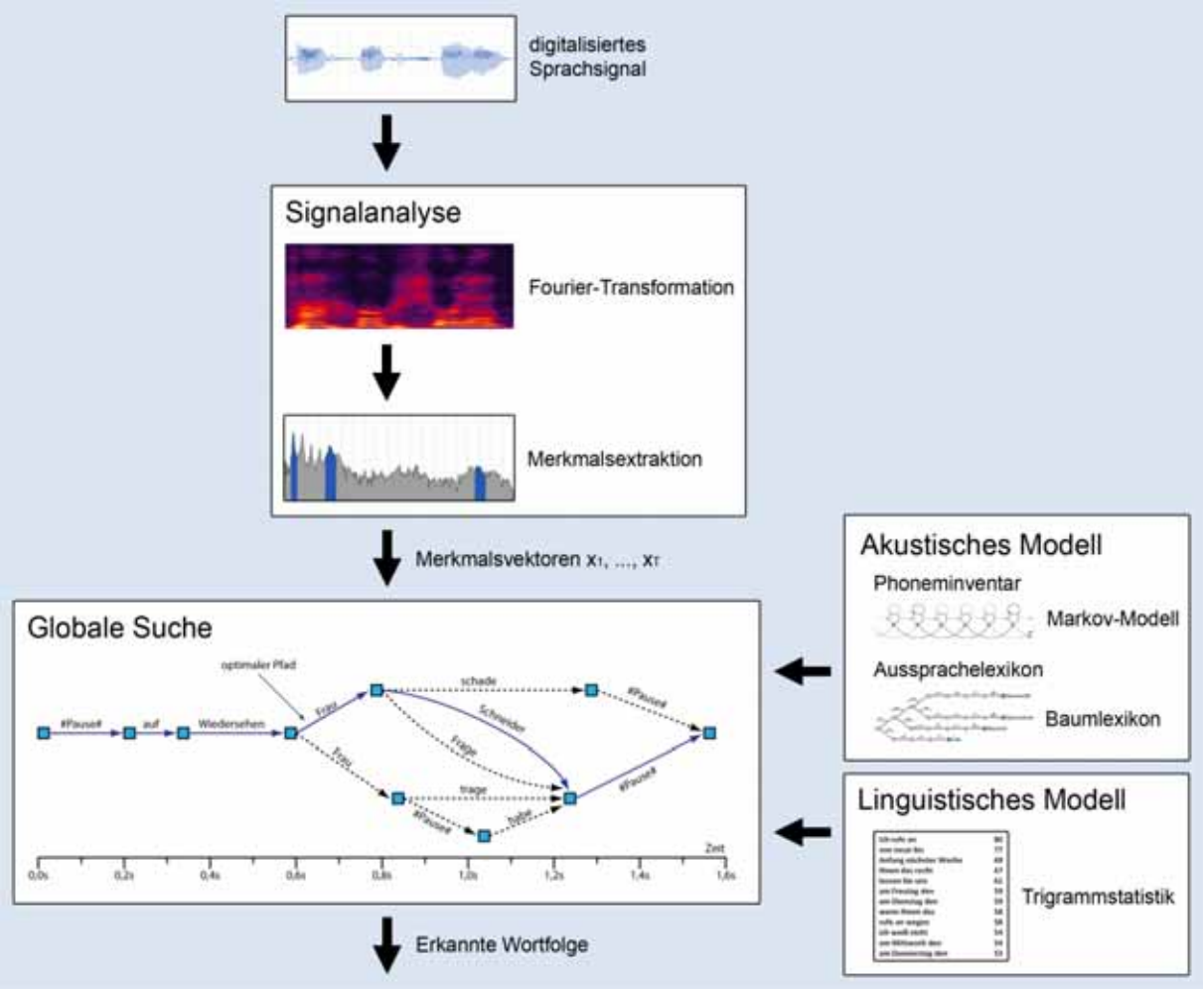


Abbildung 6: Vorgehensweise Spracherkennung

2.3.4.2 Signalanalyse

Als Ausgangspunkt liegt das digitalisierte Sprachsignal vor, wie es z.B. von der Soundkarte aufgezeichnet wurde. Dieses besteht aus einer zeitlichen Abfolge von Amplitudenwerten, es wird daher auch als Zeitsignal bezeichnet. Nun müssen aus diesem Signal Merkmale extrahiert werden, die typisch für die einzelnen Laute der Sprache sind. Dazu wird das Signal in etwa 25 Millisekunden lange Abschnitte unterteilt, wobei sich die einzelnen Abschnitte mit dem Vorgänger und dem Nachfolger teilweise überlappen. Für die Abschnitte wird nun jeweils eine Spektralanalyse durchgeführt, um zu ermitteln wie stark welche Frequenzen vertreten sind. Dies geschieht mit Hilfe der Fourier-Transformation.

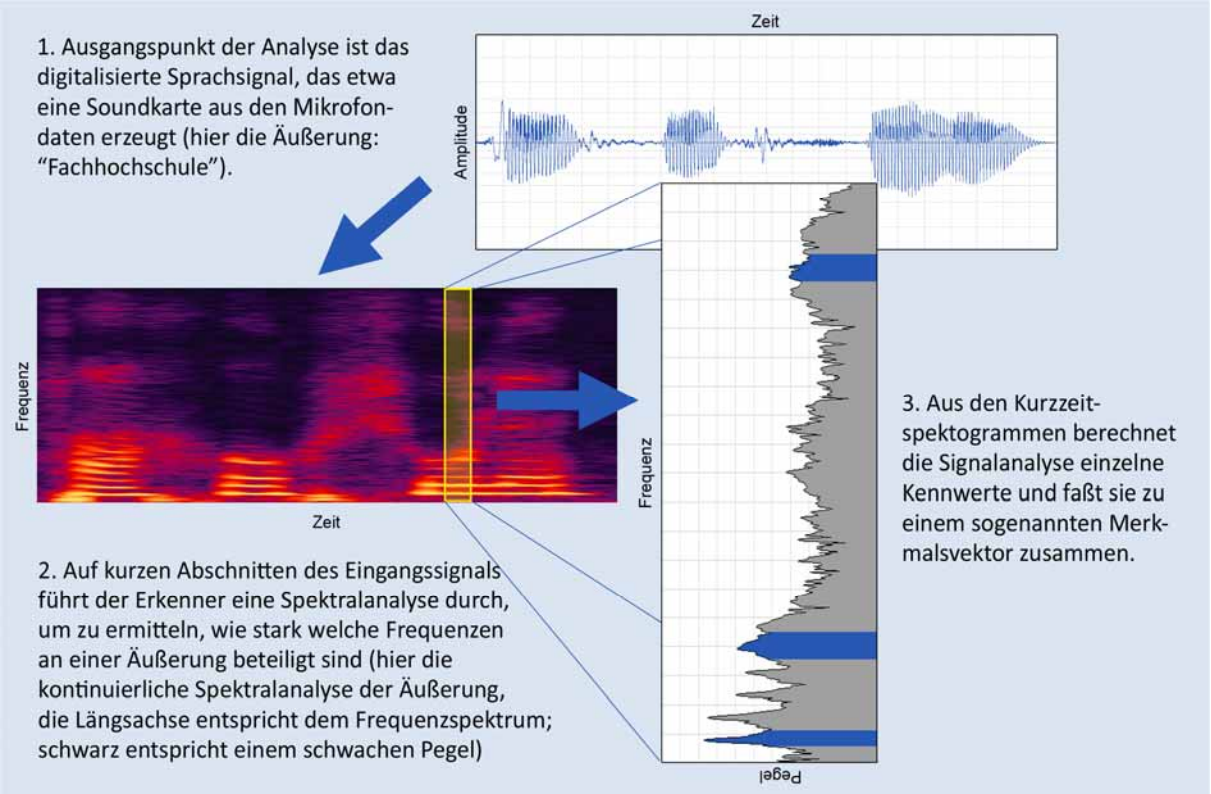


Abbildung 7: Signalanalyse

Aus den erhaltenen Kurzzeitspektren werden nun einzelne Kennwerte berechnet, die anschließend zu Merkmalsvektoren zusammengefasst werden. Mit Hilfe dieser Vektoren können die gesprochenen Laute identifiziert werden. Die zeitliche Abfolge dieser Vektoren bildet die Grundlage für die Entscheidung, welche Wortfolge gesprochen wurde.

Um den passenden Satz zu einer Folge von Merkmalsvektoren zu finden, werden Methoden der Statistik angewandt. Der Prozess lässt sich in die akustische und in die linguistische Modellierung aufteilen.

2.3.4.3 Akustische Modellierung

Mit Hilfe der akustischen Modellierung kann einer Folge von Merkmalsvektoren ein bestimmtes Wort aus dem Vokabular zugeordnet werden. Dies geschieht in zwei Schritten.

Der erste Schritt ist, die Wörter des Vokabulars als eine Abfolge von Phonemen zu beschreiben. Unter Phonemen versteht man dabei die kleinsten bedeutungsunterscheidenden Lautelemente einer Sprache. Im Deutschen sind etwa die Laute, die in Lautschrift mit [d] und [t] notiert werden und zwischen der Bedeutung von `Dorf´ und `Torf´ unterscheiden, Phoneme. Ein Spracherkennungssystem für die deutsche Sprache benutzt in der Regel zirka 40 derartiger Phoneme. Für diesen Zweck kommt ein Aussprachelexikon zum Einsatz. Dort ist jedem Wort eine Phonem-Folge zugeordnet, die der Standardaussprache entspricht (vergleichbar mit der Lautschrift im Duden).

Da bei einem großen Vokabular der Suchvorgang nach einer passenden Phonem-Folge sehr aufwändig sein kann, wird dort das Aussprachelexikon als Baum organisiert. Dabei werden Wörter, die mit der gleichen Phonem-Kette beginnen, zu Gruppen zusammengefasst. An den Blättern des Baumes stehen die Wörter des Lexikons und der Pfad vom Stamm zu einem Blatt entspricht der Phonem-Kette eines Wortes. Durch die Anordnung als Baum wird der Aufwand für den Suchvorgang gegenüber der linearen Anordnung etwa um den Faktor 1,5 bis 6 reduziert.

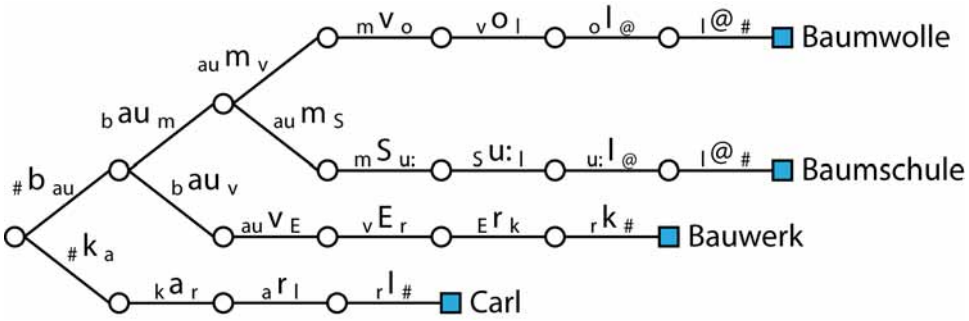


Abbildung 8: Beispiel für ein Aussprachelexikon mit Baumstruktur

Der zweite Schritt besteht darin, die Wahrscheinlichkeitsverteilungen der Merkmalsvektoren für die einzelnen Phoneme zu modellieren. Die Sprechgeschwindigkeit kann mitunter sehr stark schwanken, deshalb wird ein Modell benötigt, das diese Schwankungen berücksichtigt. Hier kommen die sogenannten Hidden Markov Modelle zum Einsatz.

„Das **Verborgene Markow-Modell** (VMM, engl. **Hidden Markov Model**, HMM), benannt nach dem russischen Mathematiker Andrei Andrejewitsch Markow, ist ein stochastisches Modell, das sich durch zwei Zufallsprozesse beschreiben lässt. Der erste Zufallsprozess entspricht dabei einer Markow-Kette, die durch Zustände und Übergangswahrscheinlichkeiten gekennzeichnet ist. Die Zustände der Kette sind von außen jedoch nicht direkt sichtbar (sie sind verborgen, hidden). Stattdessen erzeugt ein zweiter Zufallsprozess zu jedem Zeitpunkt beobachtbare Ausgangssymbole gemäß einer zustandsabhängigen Wahrscheinlichkeitsverteilung. Die Aufgabe besteht häufig darin, aus der Sequenz der Ausgabesymbole auf die Sequenz der verborgenen Zustände zu schließen.“ [Wik081]

Jedes Phonem wird dabei durch ein Hidden-Markov-Modell repräsentiert, wodurch sich der zeitliche Ablauf des Auftretens bestimmter Merkmale selbst innerhalb einzelner Phoneme optimal modellieren lässt.

Ein Phonem-Modell besteht aus mehreren Zuständen. Jeder Zustand ist verknüpft mit einer sogenannten Emissionsverteilung für die Merkmalsvektoren und mit Transitionswahrscheinlichkeiten für die drei möglichen Übergänge. Mit den Emissionsverteilungen wird einem Merkmalsvektor eine Wahrscheinlichkeit zugeordnet, mit der er im zugehörigen Zustand beobachtet wird. Die Parameter der Emissionsverteilungen werden in einer Trainingsphase anhand von umfangreichen Trainingsdaten geschätzt.

Durch die drei möglichen Übergänge zwischen den Zuständen lässt sich die Variation der Sprechgeschwindigkeit modellieren. Der direkte Übergang zum Nachfolgezustand entspricht einer normalen Sprechgeschwindigkeit, das Wiederholen eines Zustandes drückt eine langsamere Geschwindigkeit aus und das Überspringen von Zuständen eine schnellere.

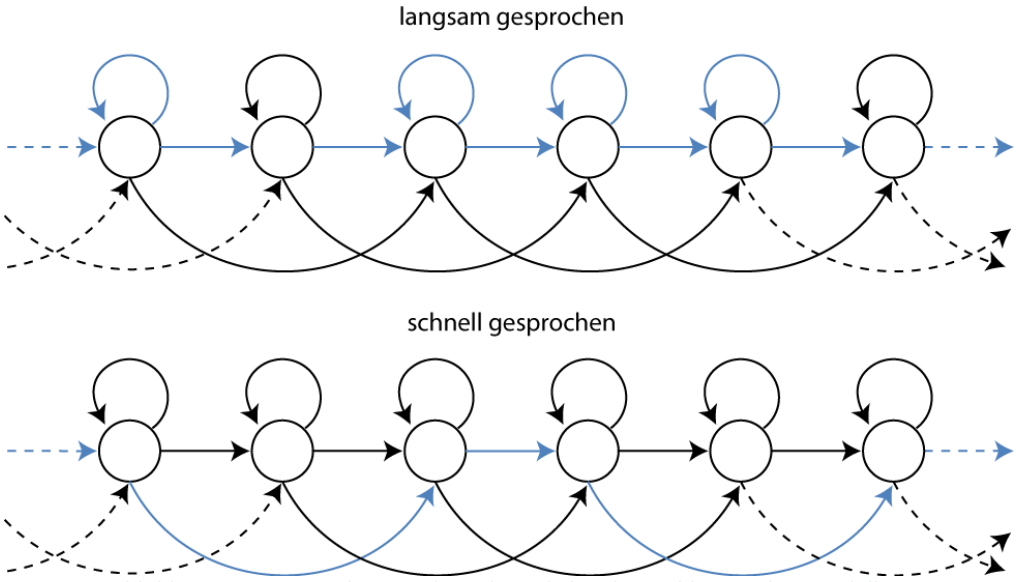


Abbildung 9: Variation der Sprechgeschwindigkeit bei Hidden Markov Modellen

Die Übergänge der Hidden Markov Modelle und die Abfolge der Merkmalsvektoren werden schließlich zu einer Gitterstruktur kombiniert. Es ist derjenige Pfad durch das Gitter zu finden, der das Produkt aus den zugehörigen Emissions- und Transitionswahrscheinlichkeiten maximiert. Das erhaltene Produkt entspricht dann der akustischen Wahrscheinlichkeit des Wortes.

2.3.4.4 Linguistische Modellierung

Mit Hilfe der linguistischen Modellierung soll die Wahrscheinlichkeit für das Auftreten einer bestimmten Satzhypothese berechnet werden. Diese lässt sich berechnen, indem man das Produkt der bedingten Wahrscheinlichkeiten aller im Satz enthaltenen Wörter bildet. Die Wahrscheinlichkeit für das Auftreten eines einzelnen Wortes ist dabei immer von allen vorhergegangenen Wörtern abhängig. Da es jedoch sehr viele unterschiedliche Folgen von Vorgängerworten geben kann, werden zur Vereinfachung nur die beiden letzten betrachtet. Die so erhaltene Gruppe aus drei Wörtern wird als Trigramm bezeichnet.

Die Wahrscheinlichkeiten für das Auftreten der Trigramme wird in einem vorher separat durchgeführten Training ermittelt. Hier ist es wichtig, einen großen Textkorpus zu verwenden, der für das spätere Anwendungsgebiet typisch ist. Da nicht jedes mögliche Trigramm im Trainingstext vorkommen kann, wird auch auf die Wahrscheinlichkeiten für das Auftreten von Wortpaaren (Bigrammen) und Einzelwörtern (Unigrammmen) zurückgegriffen.

Ich rufe an	80
von neun bis	77
Anfang nächster Woche	69
Ihnen das recht	67
lassen Sie uns	61
am Freutag den	59
am Dienstag den	59
wenn Ihnen das	58
rufe an wegen	58
ich weiß nicht	54
am Mittwoch den	54
am Donnerstag den	53
halten wir das	52
wäre Ihnen das	52
oder Anfang nächster	50
ich freue mich	49
es geht um	49
wenn es Ihnen	49
von mir aus	44
paßt es Ihnen	44

Tabelle 8: Trigramm-Häufigkeiten in einem Beispieltext

2.3.4.5 Die Suche

Das Ziel der Suche ist es, diejenige Wortfolge zu ermitteln, für die das Produkt aus akustischer und linguistischer Wahrscheinlichkeit am größten ist. Es können natürlich nicht alle möglichen Wortfolgen betrachtet werden, da der Aufwand dabei zu gigantisch wäre. Zudem sind bei der Erkennung sowohl die Wörter als auch die Wortgrenzen unbekannt. Der folgende Algorithmus führt die Optimierung über diese beiden Arten von Unbekannten in einem Schritt durch.

Zunächst wird ein dreidimensionales Gitter aufgespannt. Die erste Achse entspricht dabei der Zeitachse. Auf der zweiten Achse liegen die Zustandsketten der einzelnen Wörter und auf der dritten Achse befinden sich die zugehörigen Wortindizes. Innerhalb der Wörter sind nur die Transitionen der Hidden-Markov-Modelle erlaubt. An den Wortenden existieren Transitionen zu den Anfängen weiterer Wörter. Diese Übergänge werden mit Hilfe des Sprachmodells bewertet.

Die Aufgabe der Suche liegt nun darin, den 'besten' Pfad durch das Gitter zu finden und somit über Wortgrenzen und Wörter zu optimieren. Für die Reduzierung des Aufwandes ist es wichtig, dass unwahrscheinliche Pfade relativ früh verworfen werden.

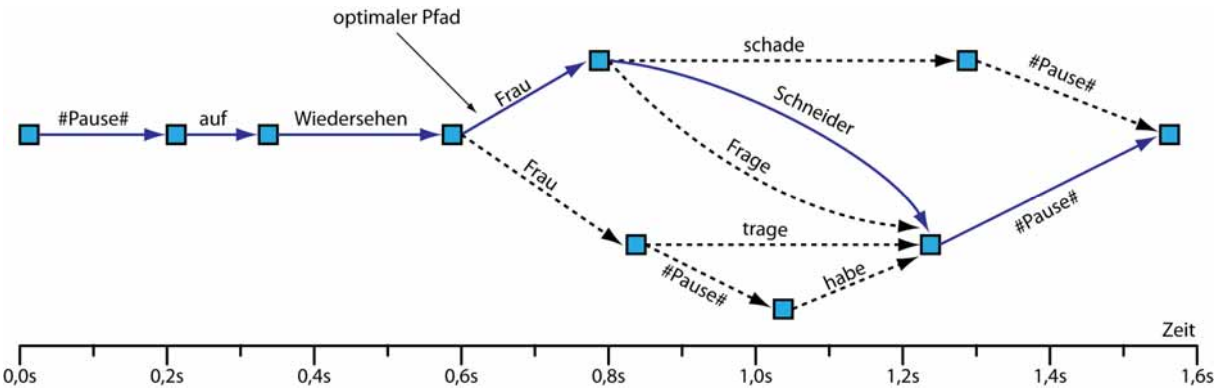


Abbildung 10: Worthypothesengraph

Der wahrscheinlichste Pfad, der bei der Suche ermittelt worden ist, entspricht dem erkannten Satz.

2.3.5 Schwierigkeiten

Obwohl im Bereich der Spracherkennung etliche Fortschritte gemacht wurden, ist eine komplett fehlerfreie Erkennung bisher noch nicht möglich. Folgende Umstände wirken sich erschwerend auf die Erkennung aus [Hab98]:

- Die Aussprache eines Lautes kann von einer Äußerung zur anderen stark variieren, selbst beim gleichen Wort und Sprecher.
- In der Sprechgeschwindigkeit treten starke Schwankungen auf.
- Die akustische Realisierung eines Lautes hängt im Allgemeinen von den vorangegangenen und den nachfolgenden Lauten ab. Dieser Effekt wird auch als Koartikulation bezeichnet.
- Im kontinuierlichen Redefluss gibt es keine deutlichen Laut- und Wortgrenzen.
- In der praktischen Anwendung treten häufig Störungen auf, wie z.B. Hintergrundgeräusche (Bürolärm, Fahrgeräusche) oder Rauschen (Telefonleitung).
- Gesprochene Äußerungen sind nicht immer grammatikalisch korrekt, d. h. Satzhypothesen, die grammatikalisch keinen Sinn ergeben, können nicht ausgeschlossen werden.

2.4 Sprachsynthese

2.4.1 Einleitung

Unter Sprachsynthese versteht man die maschinelle Reproduktion der menschlichen Sprache. Dabei wird ein schriftlicher Text in ein Sprachsignal umgesetzt. Die Sprachsynthese wird oft auch als Text-To-Speech (TTS)-System bezeichnet.

Die Sprachsynthese ist eine wichtige Komponente eines Sprachdialogsystems. Sie ist dafür verantwortlich, dass der Benutzer auch eine Rückmeldung vom System bekommt.

Typische Einsatzgebiete sind unter Anderem:

- Abhören von SMS-Nachrichten per Telefon
- Sprachausgabe in Lernmodulen (die Kombination von visuellen und auditiven Elementen erhöht die Wirksamkeit)
- Screen-Reader (Programme, die Bildschirminhalte vorlesen als Zugangshilfe für Sehbehinderte)
- Aussprache-Beispiele in elektronischen Wörterbüchern
- Übersetzungsprogramme mit Sprachausgabe
- Navigationsgeräte

2.4.2 Geschichte

Der Anfang der Sprachsynthese liegt lange vor dem Beginn des digitalen Zeitalters. Die ersten Versuche zur synthetischen Erzeugung der menschlichen Sprache wurden bereits in der zweiten Hälfte des 18. Jahrhunderts unternommen. Dabei handelte es sich noch um rein mechanische Apparaturen.

Ch. G. Kratzenstein, Professor der Physiologie in Kopenhagen, gelang es im Jahr 1773, Vokale durch an Orgelpfeifen angeschlossene Resonanzröhren zu erzeugen.

Etwa zur gleichen Zeit beschäftigte sich auch Wolfgang von Kempeln, ein Ingenius im Dienste von Maria Theresia in Wien, mit dem Bau einer sprechenden Maschine.

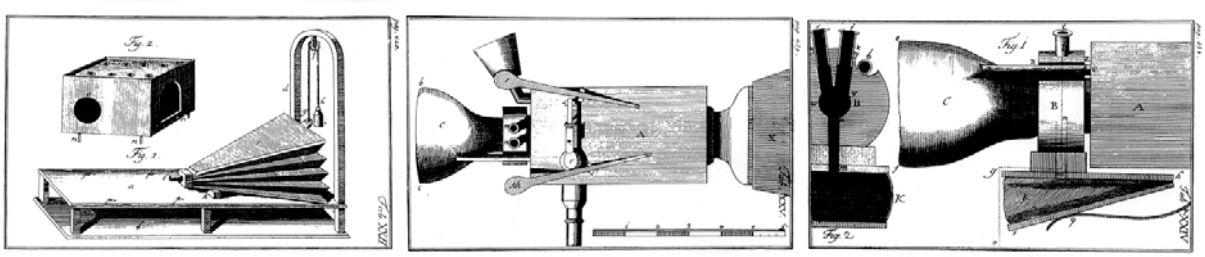


Abbildung 11: Zeichnungen von Kempelns sprechender Maschine

Im 19. Jh. wurden noch weitere ähnliche Maschinen mit einigen Verbesserungen entworfen, die aber keine große Neuerung auf diesem Gebiet darstellten.

Die Entwicklung der Elektrotechnik Anfang des 20. Jh. ermöglichte es, Sprache auch auf elektrischem Wege zu erzeugen. Das erste bekannte Gerät dieser Art war der von Homer Dudley entwickelte VODER, der 1939 auf der Weltausstellung in New York der Öffentlichkeit vorgestellt wurde.

Abbildung 12: Homer Dudleys VODER

Seit 1970 war die weitere Entwicklung der Sprachsynthesetechnik eng mit der der elektronischen Datenverarbeitung verknüpft. Man simulierte jetzt nicht mehr nur die natürliche Sprachproduktion mit elektrischen Kreisen, sondern diese elektrischen Kreise wurden ihrerseits auch wieder nur simuliert. Computer machten es möglich, Sprachsynthese für praktische Zwecke einzusetzen, und es wurden verschiedene Systeme zur Umwandlung von Text in Sprache entwickelt [Tra97].

2.4.3 Funktionsweise

2.4.3.1 Natural Language Processing (NLP)

Die Sprachsynthese wird in der Regel in zwei Schritten durchgeführt, wobei für jeden Schritt eine eigene Komponente zuständig ist. Der erste Schritt wird als Natural Language Processing (NLP) bezeichnet. Hierbei wird der Eingabetext in Lautschrift und Prosodie umgesetzt. Um die Lautschrift für die einzelnen Wörter zu ermitteln, kommen Aussprache-Lexika zum Einsatz. Die Prosodie oder auch Satzmelodie orientiert sich hauptsächlich an der Satzzeichensetzung [Spr081] [Bur08].

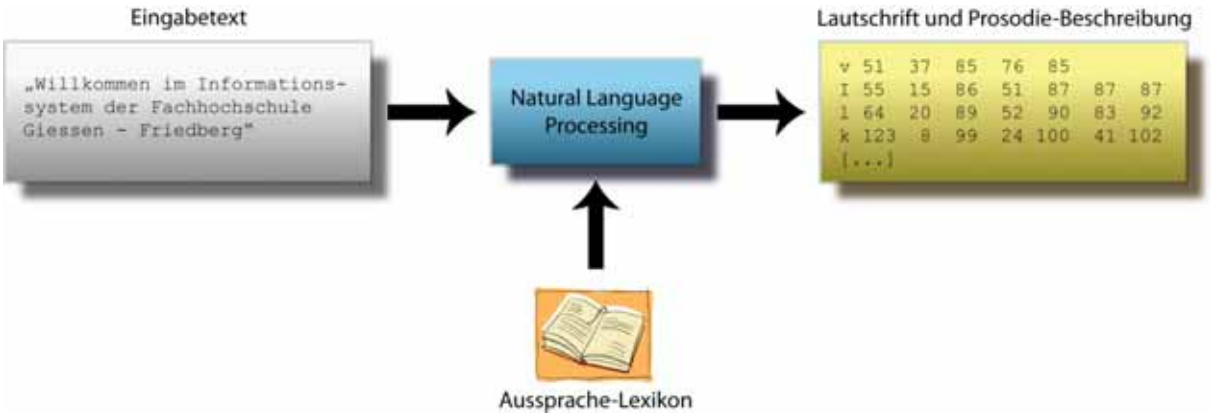


Abbildung 13: Natural Language Processing

2.4.3.2 Digital Speech Processing (DSP)

Der zweite Schritt wird als Digital Speech Processing (DSP) bezeichnet. Hier wird mit Hilfe der Ausgabe aus dem ersten Schritt das Sprachsignal synthetisiert.

Abbildung 14: Digital Speech Processing

Die Synthese des Sprachsignals erfolgt dabei nach einem der folgenden drei Hauptverfahren [Bur08]:

Formant Synthese

Das älteste Verfahren ist die Formant Synthese. Hier wird das Sprachsignal anhand von physikalischen Modellen berechnet (Formanten nennt man die Resonanz-Frequenzen im Sprechtrakt). Dieses Verfahren ist sehr flexibel und es werden nur geringe Ressourcen benötigt, es hat aber auch den Nachteil, dass das erzeugte Sprachsignal nicht sehr natürlich klingt.

Diphon Synthese

Unter einem Diphon versteht man die Kombination von zwei aufeinander folgenden Sprachlauten. Das Sprachsignal wird bei diesem Verfahren durch eine Verkettung solcher Diphone erzeugt. Die Prosodie-Anpassung (Rhythmus und Satzmelodie) wird durch Signal-Manipulation erreicht (Abhängig von der Kodierung der Diphone). Dieses Verfahren ist etwas aufwändiger als die Formant Synthese, das generierte Signal klingt dafür aber auch etwas natürlicher.

Non-uniform unit selection

Das modernste und komplexeste Verfahren ist die Non-uniform unit selection. Aus einer großen Datenbasis werden die am besten passenden Sprachteile (units) herausgesucht und miteinander verkettet, wobei diese eine variable Länge besitzen (non-uniform). Dabei wird eine doppelte Kostenfunktion minimiert: die Stücke sollen gut aneinander passen (Verkettungs-Kosten) und die Vorgaben der

Ziel-Prosodie erfüllen (Target-Kosten). Dieses Verfahren klingt sehr natürlich, hat dafür aber auch Nachteile gegenüber den anderen Verfahren: Es ist sehr unflexibel bzgl. Situationen, die nicht in der Datenbasis abgebildet sind und erfordert hohe Ressourcen. Meistens wird dieses Verfahren bei Server-basierten Anwendungen eingesetzt.

2.5 VoiceXML

2.5.1 Einführung

VoiceXML (Voice Extensible Markup Language) ist eine spezielle Sprache zur Formulierung von Sprachdialogen. Sie beruht auf der Auszeichnungssprache XML, welche häufig im Zusammenhang mit dem Internet eingesetzt wird. Mit Hilfe von XML können hierarchisch strukturierte Daten in Textform dargestellt oder verarbeitet werden.

Die Sprache VoiceXML beschreibt die Interaktion zwischen Mensch und Maschine und vereint dabei die einzelnen Komponenten des Dialogsystems, welche die benötigen Fähigkeiten des Systems zur Verfügung stellen. Zu diesen gehört:

- Ausgabe von synthetischer Sprache (Text-To-Speech)
- Ausgabe von Audiodateien
- Aufzeichnung von Sprache
- Erkennung von Spracheingaben
- Erkennung von DTMF-Eingaben
- Steuerung des Dialogflusses
- Telefonie-Merkmale wie Gesprächsannahme, Anrufumleitung oder Auflegen

VoiceXML stellt das Grundgerüst zur Verfügung, um Eingaben in Form von Zeichen oder Sprache zu sammeln, die Ergebnisse der Eingabe speziellen, im Dokument definierten, Variablen zuzuweisen und Entscheidungen zu treffen, die die weitere Interpretation der Dokumente beeinflussen [Mei07].

2.5.2 Ziele von VoiceXML

Das Hauptziel von VoiceXML ist es, interaktive Sprachdialogsysteme mit den Vorteilen von web-basierter Entwicklung zu versehen. Bei der Entwicklung der Sprache hatte man aber auch noch weitere Ziele im Sinn [Wor04]:

- Die Interaktionen zwischen Client und Server sollen minimiert werden, indem mehrere Interaktionen in einem einzelnen Dokument angegeben werden können.
- Der Anwendungsentwickler soll sich nicht um System- und Plattform-abhängige Details kümmern müssen.
- Der Programmcode für die Benutzerinteraktion (in VoiceXML-Dateien) soll von den internen Funktionen des Dienstes (z.B. in CGI-Skripten) getrennt werden.
- Die Portabilität des Dienstes soll über verschiedene Implementations-Plattformen hinweg gefördert werden. VoiceXML soll als gemeinsame Sprache für Inhaltsanbieter, Werkzeug-Entwickler und Plattform-Provider dienen.
- Die Erstellung von einfachen Anwendungen soll möglichst unkompliziert sein und gleichzeitig soll VoiceXML auch Funktionen für komplexe Dialoge bereithalten.

2.5.3 Grundaufbau

Das oberste Element in einer VoiceXML-Datei ist das Element <vxml>, welches in erster Linie als ein Behälter für Dialoge fungiert. Es gibt zwei Arten von Dialogen: Formulare (forms) und Menüs (menus). Formulare bieten Informationen an und erfassen Eingaben, während Menüs eine Auswahl an (Unter-)Dialogen anbieten, zu denen als nächstes gewechselt werden kann [Wor04].

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.0">
  <form>
    <block><prompt>Guten Tag!</prompt></block>
  </form>
</vxml>
```

Das obige Beispiel gibt den Satz „Guten Tag!“ als synthetische Sprache aus. Es besteht aus einem einzigen Formular, welches in einem Block die Anweisung <prompt> für die Sprachausgabe enthält. Das Element <block> stellt lediglich ein Behälter für ausführbaren Code dar. Weil kein Folgedialog angegeben ist, endet der Dialog nach der Ausgabe.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.0">
  <form>
    <field name="getraenk">
      <prompt>
        Waehlen Sie ein Getraenk:
        Cola, Fanta, Sprite oder Wasser.
      </prompt>
      <option>Cola</option>
      <option>Fanta</option>
      <option>Sprite</option>
      <option>Wasser</option>
    </field>
    <block>
      <prompt>Sie waehlten: <value expr="getraenk"/></prompt>
      <goto next="bestellung7.vxml"/>
    </block>
  </form>
</vxml>
```

In diesem Dialog wird der Benutzer aufgefordert, ein Getränk aus den vorgegebenen Möglichkeiten auszuwählen. Durch das Element <field> wird eine Spracheingabe angefordert. In der Regel besteht es aus einer Sprachausgabe und mehreren Antwortmöglichkeiten, die durch die <option>-Elemente festgelegt sind. Das Ergebnis der Eingabe wird in einer Variablen abgespeichert, die über das Attribut „name“ im <field>-Element definiert wurde.

Nach der Spracheingabe wird hier die Wahl des Benutzers nochmal ausgegeben. Dazu wird das Element <value> verwendet, mit dem der Wert eines bestimmten Ausdrucks in eine Sprachausgabe eingefügt werden kann, wie z.B. der Inhalt einer Variablen.

Die <goto>-Anweisung führt dazu, dass, im Gegensatz zum vorherigen Beispiel, der Dialog hier nicht endet, sondern mit der Datei „bestellung7.vxml“ fortgesetzt wird.

2.5.4 Weitere VoiceXML-Elemente

2.5.4.1 Verzweigungen

Eine Verzweigung wird durch den Bedingungsoperator <if> erreicht. Liefert die angegebene Bedingung den Wert „true“ (wahr) zurück, so werden die Anweisungen innerhalb des <if>-Elements ausgeführt, andernfalls wird mit der Ausführung des Codes nach dem Element fortgefahren. Dies kann genutzt werden um z.B. abhängig von den Benutzereingaben den Dialog in einem anderen VoiceXML-Dokument fortzusetzen.

Auf die <if>-Anweisung können zusätzlich noch die Operatoren <else> und <elseif> folgen, welche alternative Befehle enthalten, die ausgeführt werden, falls die vorherige <if>-Bedingung nicht zugetroffen hat.

2.5.4.2 Skripte

Mit dem <script>-Element kann, analog zu HTML-Dokumenten, ein Block in Skriptsprache angegeben werden. Als Skriptsprache muss dabei ECMAScript verwendet werden, welches oft auch als JavaScript bezeichnet wird. Das <script>-Element kann in den Elementen <vxml> und <form> vorkommen oder in anderen Elementen mit ausführbarem Inhalt (z.B. <block> oder <if>). Skripte im <vxml>-Element werden sofort beim Laden des Dokuments ausgewertet, Skripte in anderen Elementen werden immer dann ausgewertet, wenn das Element bei der Ausführung des Dialoges betreten wird.

2.5.4.3 Variablen

Mit Variablen können Informationen im laufenden Dialog gespeichert und abgerufen werden. Eine Variable kann mit der Anweisung

```
<var name="beispiel"/>
```

angelegt werden. Über den Namen kann dann später wieder darauf zugegriffen werden. Einige Elemente, darunter auch das <field>-Element legen Variablen automatisch an, z. B. um das Erkennungsergebnis festzuhalten.

Mit folgender Anweisung kann der Inhalt der Variable aus obigem Beispiel ausgelesen werden:

```
<value expr="beispiel"/>
```

2.5.4.4 Menüs

Das <menu>-Element erlaubt das Verzweigen in verschiedene Unterdialoge. Es stellt eine vereinfachte Form des <field>-Elements dar.

```
<menu>
      <prompt timeout="6s">
            Hauptmenü. Wählen Sie einen Bereich: Mensa, Notenaus-
            kunft oder Klausuranmeldung.
      </prompt>
      <choice next="info_mensa_welcome.vxml">
            Mensa
      </choice>
      <choice next="info_noten_welcome.vxml">
            Notenauskunft
      </choice>
      <choice next="info_klausur_welcome.vxml">
            Klausuranmeldung
      </choice>
</menu>
```

Das Ergebnis der Erkennung muss hier nicht manuell mit dem <if>-Operator überprüft werden. Auch die <goto>-Anweisung wird nicht benötigt. Die möglichen Optionen werden als <choice>-Elemente angegeben und gleich mit den entsprechenden Unterdialogen verknüpft.

3 Implementierung

3.1 Rahmenbedingungen

3.1.1 Einleitung

Bevor mit der Implementierung begonnen werden kann, sollte man sich noch Gedanken über die Rahmenbedingungen machen. Dazu gehört die Programmiersprache, die Entwicklungsumgebung und das System auf dem bzw. für das die Anwendung entwickelt werden soll.

Weiterhin sollte man sich schon vorher ein grobes Konzept überlegen, wie die gewünschten Programmfunktionen realisiert werden können.

3.1.2 Programmiersprache

Die Wahl der Programmiersprache fiel auf JAVA. Der Hauptgrund, der für JAVA spricht, ist, dass der Sprachserver (FBSV) bereits in dieser Sprache implementiert ist. Eine Alternative wäre u. A. ein von Grund auf neues Projekt in der Sprache C/C++, das sich z.B. auf die Programmierschnittstelle Microsoft SAPI stützt.

Die Verwendung von JAVA hat aber noch ein paar weitere Vorteile:

Portierbarkeit: Anwendungen, die in JAVA geschrieben worden sind, laufen auf beliebigen anderen Betriebssystemen und sind auch unabhängig von der Hardware (sofern eine JAVA Laufzeitumgebung unterstützt wird).

Nützliche Bibliotheksfunktionen: Viele Funktionen, die das Projekt erfordert, sind in der JAVA Laufzeitumgebung bereits enthalten. Dazu gehören z.B. Funktionen für den Netzwerk- / Internetzugriff oder auch Funktionen zur Ansteuerung der Hardware, wie z.B. Audio-Ein/Ausgabe.

Als letztes spricht noch die Tatsache für JAVA, dass der Autor in dieser Programmiersprache die größte Erfahrung besitzt.

3.1.3 Entwicklungsumgebung

Die Entwicklungsumgebung stellt grundlegende Werkzeuge bereit, die zum Programmieren benötigt werden. In der Regel besteht eine Entwicklungsumgebung mindestens aus einem Quelltext-Editor und einem Compiler.

Für dieses Projekt wurde Eclipse[8] als Entwicklungsumgebung ausgewählt. Eclipse ist ein kostenloses Open-Source-Framework, das schon seit einigen Jahren den Standard bei der JAVA-Entwicklung darstellt.

[8] URL: http://www.eclipse.org

Die Vorteile von Eclipse sind:

- kostenlos
- ständige Verbesserungen dank Open-Source
- Unterstützung einer Vielzahl von Programmiersprachen
- Echtzeit-Syntax-Überprüfung
- Anpassung der dargestellten Informationen durch sog. Views und Perspektiven
- Erweiterbarkeit durch Plug-Ins
- Für viele Plattformen verfügbar, da auf JAVA basierend

3.1.4 Betriebssystem

Als Betriebssystem wird, sowohl für die Entwicklung als auch für die Anwendung selbst, Microsoft Windows XP verwendet. Der Grund dafür ist, dass auf den Rechnern, die für die Entwicklung zur Verfügung stehen, Windows XP bereits installiert ist. Auch im späteren Gebrauchsumfeld stehen Rechner mit Windows XP zur Verfügung.

Dank der Portierbarkeit, die JAVA mit sich bringt, ist es auch denkbar, dass die Anwendung später auf Linux-Rechnern eingesetzt wird. Dazu wären voraussichtlich nur minimale Änderungen nötig.

3.1.5 Grundkonzept

Das Grundkonzept ist recht einfach: Der VoIP-Client wird beim Start des Sprachservers geladen. Wenn eine VoIP-Verbindung besteht, dann werden die ankommenden Audiodaten zum Sprachserver umgeleitet. Der Sprachserver verwendet für die Erkennung nun diese umgeleiteten Daten, anstatt derjenigen, die vom Audioeingang der Soundkarte kommen.

Genauso sendet der VoIP-Client nun nicht mehr die Daten, die vom Audioeingang kommen, sondern die Daten, die von der Sprachsynthese ausgegeben werden.

3.2 Vorstellung der Komponenten

3.2.1 Der Sprachserver

3.2.1.1 Einleitung

Der Sprachserver, auch als Friedberger Sprachserver (FBSV) bezeichnet, ist eine JAVA-Anwendung, die von Prof. Dr. Stephan Euler entwickelt wurde. Der Zweck des Sprachservers ist es, Sprachdialoge mittels Spracherkennung und Sprachsynthese zu ermöglichen.

Die Funktion des Sprachservers kann grob in vier Komponenten unterteilt werden:

1. Der VoiceXML-Parser, der Dialoge in Form von VoiceXML-Dateien abarbeitet
2. Der Wortgrenzendetektor, der eine Äußerung des Benutzers aufzeichnet und dabei Anfang und Ende erkennt
3. Der Spracherkenner, der die Äußerung analysiert und die wahrscheinlichste Antwort zurückliefert
4. Die Sprachsynthese, die Textausgaben in Audiosignale umwandelt und ausgibt

Das folgende Diagramm erklärt das Zusammenspiel der verschiedenen Komponenten an einem kleinen Beispiel:

Abbildung 15: Funktionsweise des Sprachservers

Im Folgenden werden nun die einzelnen Komponenten etwas näher beschrieben.

3.2.1.2 VoiceXML-Parser

Der VoiceXML-Parser ist das Grundgerüst des Sprachservers. Er öffnet und verarbeitet die .vxml-Dateien, in denen festgelegt ist, welche Aktionen der Server ausführen soll. Sie enthalten die Vorlage für den Ablauf des gesamten Dialogs.

Die Bestandteile einer .vxml-Datei werden als Items bezeichnet. Das Extrahieren der einzelnen Items aus der Datei geschieht mit Hilfe von SAX (Simple API for XML). SAX ist ein Standard, der eine Programmierschnittstelle zum Zugriff auf Daten im XML-Format beschreibt. Die nötigen Funktionen dazu sind bereits in der JAVA Laufzeitumgebung im Paket javax.xml enthalten.

Nutzt eine Anwendung den SAX-Standard, so können bestimmten Ereignissen spezielle Funktionen zugewiesen werden. Man spricht dabei von sogenannten callback-Funktionen. Ein Ereignis wäre z.B. das Erreichen eines Items für die Audioausgabe. Hier würde eine Funktion aufgerufen werden, die dafür sorgt, dass auch tatsächlich eine angegebene Audiodatei abgespielt wird.

Beim FBSV sind die einzelnen Funktionen jeweils in einer eigenen Klasse untergebracht. `VxItem` ist die Oberklasse mit Eigenschaften und Funktionen, die von unterschiedlichen Items genutzt werden. Die anderen Klassen sind jeweils Ableitungen dieser Klasse, die für das Item spezifische Funktionen enthalten.

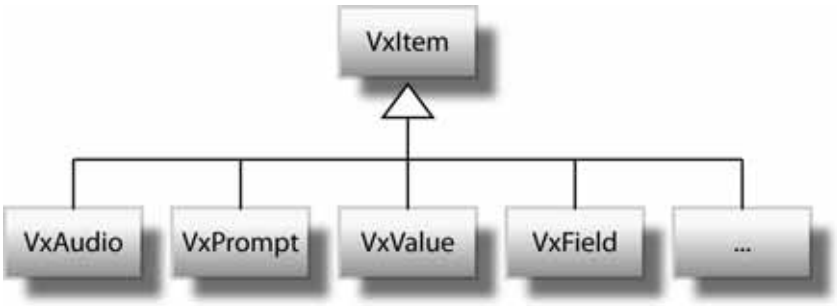


Abbildung 16: Hierarchie der Item-Klassen

Tabelle 9 listet ein paar der Item-Klassen und deren Funktionen auf.

Klasse	Verknüpft mit Item / Tag	Funktion
VxAudio	<audio>	Ausgabe einer Audio-Datei
VxPrompt	<prompt>	Text-/Sprachausgabe
VxValue	<value>	Wert einer Variablen zuweisen/auslesen
VxField	<field>	Benutzereingabe (z.B. durch Spracherkennung)
VxOption	<option>	Auswahlmöglichkeit für Spracherkennung
VxGoto	<goto>	Navigationsanweisung (z.B. Sprung in andere Datei)

Tabelle 9: Item-Klassen des VoiceXML-Parsers

3.2.1.3 Der Wortgrenzen-Detektor

Ein Bestandteil vieler Spracherkennungs-Systeme ist ein Wortgrenzen-Detektor. Er ist dafür zuständig, zu erkennen, wann eine Äußerung beginnt und wann sie endet. Wichtig ist, dass Hintergrundgeräusche nicht als Äußerung des Benutzers gedeutet werden und fälschlicherweise an den Erkenner weitergeleitet werden.

Im folgenden Diagramm ist das Zeitsignal einer Äußerung zu sehen. Die roten Balken markieren die Stellen, die im Idealfall vom Wortgrenzen-Detektor erkannt werden.

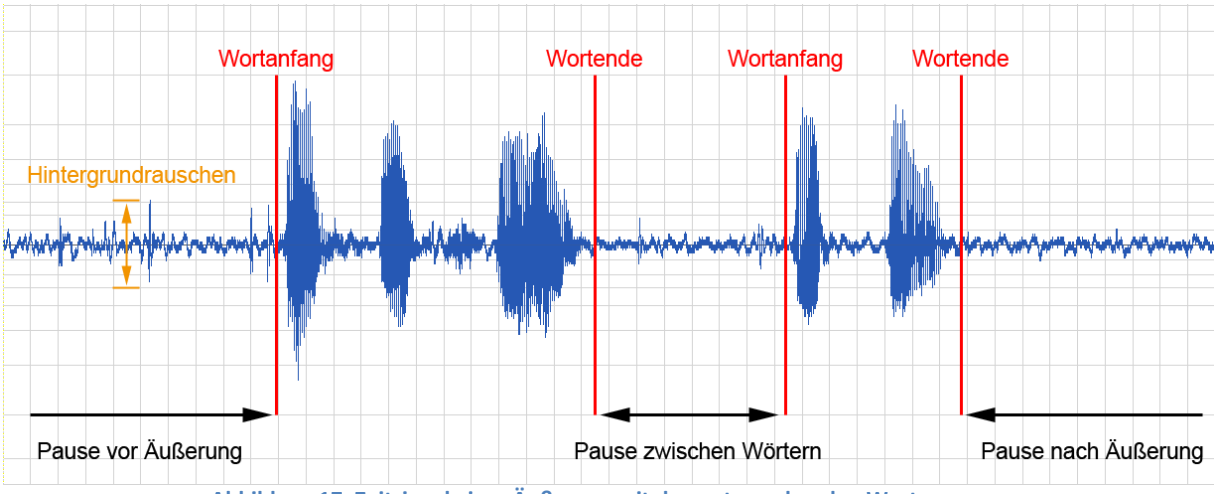


Abbildung 17: Zeitsignal einer Äußerung mit den entsprechenden Wortgrenzen

Beim FBSV ist die Klasse WGD für die Erkennung der Wortgrenzen zuständig. Hier ist ein Beispiel, wie der Aufruf erfolgen kann:

```
WGD wgd = new WGD(sampleRate);

if (wgd.isUtt( timeout ) )
{
     wgd.saveUtt( outputFilename );
}
else
{
     System.out.println("Timeout" );
}
```

Zuerst wird eine Instanz der Klasse WGD gebildet. Die gewünschte Abtastrate wird im Konstruktor übergeben, z.B. 8000 für eine Abtastrate von 8 kHz. Als nächstes wird die Methode `isUtt()` dieser Instanz aufgerufen. Übergeben wird eine Zeit in Millisekunden, welche angibt, wie lange bis zum Anfang einer Äußerung gewartet werden soll. Wird innerhalb dieser Zeit eine Äußerung erkannt, liefert die Methode den Wert `true` zurück, andernfalls `false`. Durch Aufruf der Methode `saveUtt()` kann die Äußerung unter dem angegebenen Dateinamen gespeichert werden.

Die Klasse WGD ist dabei nicht nur für die Erkennung der Wortgrenzen zuständig, sondern auch gleichzeitig für die Aufnahme der Audio-Daten, das Herausschneiden einer Äußerung und die Speicherung dieser Äußerung.

Funktionsweise:

Für die Erkennung der Wortgrenzen werden über kurze Abschnitte, auch Frames genannt, die Quadrate der Amplitudenwerte addiert. Der erhaltene Wert wird als Energie bezeichnet. Zu Beginn der Aufzeichnung wird das Hintergrundrauschen ermittelt. Übersteigt die Energie eines folgenden Frames die des Hintergrundrauschens, so wird ein Wortanfang markiert. Wenn die Energie wieder auf das Niveau des Hintergrundrauschens zurückfällt, wird ein mögliches Wortende markiert. Bleibt die Energie für eine vorbestimmte Zeit auf diesem Niveau, dann wird das Wortende festgelegt und die Aufzeichnung wird beendet.

Die Äußerung kann nun dem Erkenner übergeben werden.

3.2.1.4 Hidden Markov Toolkit (HTK)

3.2.1.4.1 Einleitung

Das Hidden Markov Toolkit (HTK) ist eine kleine Sammlung von Kommandozeilenprogrammen rund um Hidden Markov Modelle. Unter anderem lassen sich damit die für die Spracherkennung benötigten Markov Modelle erstellen.

Zur Laufzeit des Sprachservers werden aber nur zwei dieser Programme benötigt:

3.2.1.4.2 HBuild

Das Programm „HBuild.exe“ dient der Erstellung von Wortnetzwerken. Diese Netzwerke legen fest, welche Wortfolgen bei der Erkennung zulässig sind. Analog zur geschriebenen Sprache spricht man hier auch von Grammatik.

Beim Sprachserver wird HBuild nur zur Konvertierung eingesetzt. Die eigentliche Grammatik wird vom Sprachserver aus den möglichen Optionen selbst erzeugt und dann mit HBuild in ein für den Spracherkenner brauchbares Format umgewandelt.

3.2.1.4.3 HVite

Für die Spracherkennung ist das Programm „HVite.exe“ zuständig. Es berechnet mit Hilfe der Hidden Markov Modelle die wahrscheinlichste Modellabfolge und gibt dann ein erkanntes Wort oder erkannten Satz wieder zurück. HVite wird, genauso wie HBuild, extern mit Hilfe der Java-Anweisung `Runtime.getRuntime().exec()` aufgerufen. Neben dem Programmnamen enthält die Befehlszeile noch eine Reihe weiterer Angaben.

Ein möglicher Aufruf könnte z. B. folgendermaßen aussehen (hier zur Übersicht auf mehrere Zeilen aufgeteilt, in der Praxis eine Zeile):

```
HVite -S tmp.scp -w tmp.lat
-i autosave\unknown\in#0.mlf
-C arbeit\htkfiles\confs\hvite_audio_8k.cf
-H arbeit\htkfiles\hmms\hmm.p\phoneme_3_8k arbeit\htkfiles
\dicts\vokabular_info
arbeit\htkfiles\dicts\phoneme
autosave\unknown\in#0.wav
```

Die Bedeutung der einzelnen Parameter wird in Tabelle 10 erläutert:

Parameter	Beschreibung
`-S tmp.scp`	Angabe einer Skript-Datei. Sie enthält in diesem Fall nur den Dateinamen der Audiodatei.
`-w tmp.lat`	Ein Wortnetzwerk, das für die Erkennung verwendet werden soll.
`-i autosave\unknown\ in#0.mlf`	Das Erkennungsergebnis wird in dieser Datei abgespeichert.
`-C arbeit\htkfiles\ confs\hvite_audio_8k.cf`	Konfigurationsdatei mit Voreinstellungen
`-H arbeit\htkfiles\hmms\ hmm.p\phoneme_3_8k`	Diese Datei enthält die zu verwendenden Hidden Markov Modelle (HMMs).
`arbeit\htkfiles\dicts\ vokabular_info`	Das Aussprachelexikon mit allen erkennbaren Worten und deren Phonem-Folgen.
`arbeit\htkfiles\dicts\ phoneme`	Eine Datei mit einer Liste aller HMMs. In diesem Fall sind das die Phoneme.
`autosave\unknown\in#0.wav`	Die Eingabedatei mit den zu erkennenden Audio-Daten.

Tabelle 10: Parameter beim externen Aufruf von HVite.exe

Nach dem Aufruf von HVite liest der Sprachserver die erzeugte MLF-Datei ein um das Erkennungsergebnis zu erhalten.

3.2.1.5 Die Sprachsynthese

Der FBSV unterstützt zwei Systeme zur Sprachsynthese: MBROLA und FreeTTS.

MBROLA[9] ist eine externe Applikation, die für viele Betriebssysteme zur Verfügung steht. Sie unterstützt sehr viele Sprachen und für jede Sprache gibt es auch einige Stimmen. Des Weiteren gibt es noch diverse zusätzliche Programme, die zusammen mit MBROLA verwendet werden können.

FreeTTS[10] ist eine in Java geschriebene Open-Source Bibliothek. Es wird hauptsächlich nur die englische Sprache unterstützt.

Wegen der besseren Unterstützung der deutschen Sprache hat man sich bei diesem Projekt für die Verwendung von MBROLA entschieden.

In Abbildung 18 wird der Ablauf bei der Verwendung von MBROLA in vier Schritten veranschaulicht:

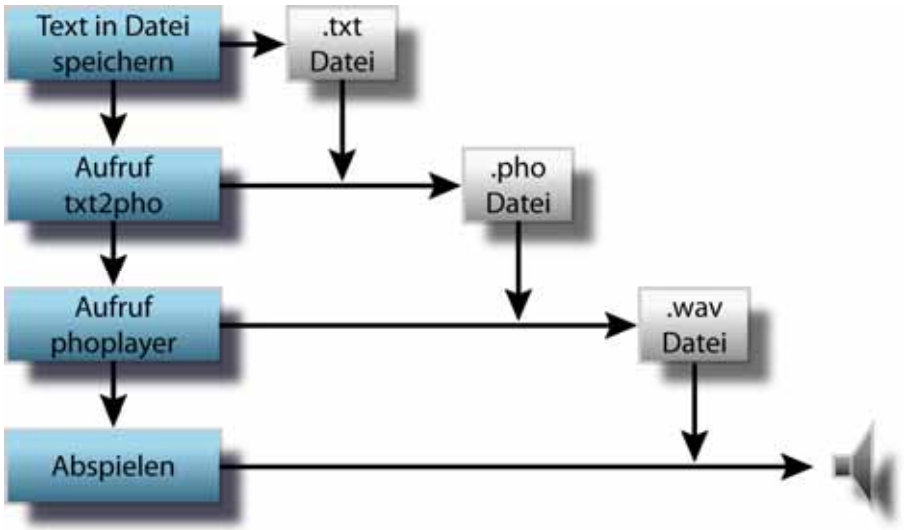


Abbildung 18: Ablauf bei der Verwendung von MBROLA

1. Der auszugebende Text wird als Textdatei gespeichert

2. Das Programm txt2pho wird aufgerufen, wobei die zuvor gespeicherte Textdatei als Parameter übergeben wird. txt2pho ist ein Zusatzprogramm für MBROLA, das den Text in eine Phonem-Folge umwandelt. Es ist speziell für die deutsche Sprache entwickelt worden. Das Ergebnis wird in einer Datei mit der Endung .pho gespeichert.

   Die folgenden zwei Abbildungen zeigen eine Textdatei und die dazu von txt2pho generierte Phonem-Folge. Neben den Phonemen enthält die Datei auch noch einige Zahlen, die MBROLA Informationen liefern, wie die Phoneme auszusprechen sind (Prosodie).

[9] URL: http://tcts.fpms.ac.be/synthesis/mbrola.html
[10] URL: http://freetts.sourceforge.net/

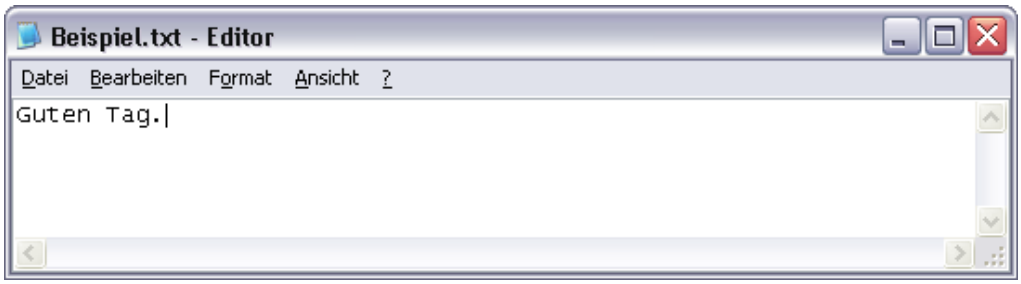


Abbildung 19: Eine Beispieldatei zur Demonstration von txt2pho

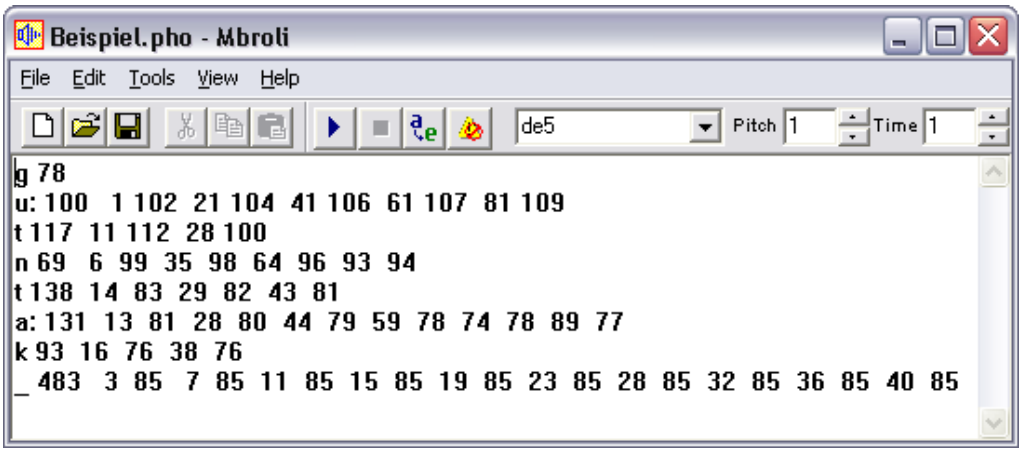


Abbildung 20: Die von txt2pho erzeugte .pho-Datei

3. phoplayer wird mit der .pho-Datei als Parameter aufgerufen. Phoplayer ist Bestandteil von MBROLA und wandelt die Phonem-Folge in ein akustisches Signal um. Dabei können verschiedene Stimmen, auch als Datenbanken bezeichnet, benutzt werden. Phoplayer kann das akustische Signal direkt abspielen oder auch als Datei speichern. Für den Demonstrator ist es notwendig, dass die Datei gespeichert wird, damit der Inhalt später auch über das Netzwerk gesendet werden kann.

4. Die vom phoplayer ausgegebene Audiodatei mit der Endung .wav kann nun abgespielt werden.

3.2.2 Der Voice-over-IP Client

3.2.2.1 Einleitung

Der Voice-over-IP Client, der bei diesem Projekt zum Einsatz kommt, wurde entwickelt von Luca Veltri, einem Dozent an der Universität von Parma, Italien. Die Arbeit wurde veröffentlicht unter der GNU General Public License.

Hier in kürze die vier Freiheiten der GNU General Public License:

1. Das Programm darf ohne jede Einschränkung und für jeden Zweck genutzt werden. Auch kommerzielle Nutzung ist erlaubt.

2. Kopien des Programms dürfen kostenlos oder gegen Geld verteilt werden, allerding muss der Quellcode zur Verfügung gestellt werden.

3. Die Arbeitsweise eines Programms darf studiert und den eigenen Bedürfnissen angepasst werden.

4. Auch die veränderten Versionen dürfen gemäß Freiheit 2 vertrieben werden. Der Quellcode der veränderten Version muss dann ebenfalls zur Verfügung gestellt werden.

Neben dem eigentlichen Programm werden keine weiteren externen Komponenten benötigt. Es werden aber externe Audio/Video-Applikationen unterstützt, z.B. für Videotelefonie. Diese kommen bei diesem Projekt jedoch nicht zum Einsatz, daher wird darauf nicht weiter eingegangen.

3.2.2.2 Die wichtigsten Klassen

3.2.2.2.1 UserAgent

Die Klasse UserAgent bildet das Fundament des VoIP-Clients. Bildet man eine Instanz von dieser Klasse, so wird der VoIP-Client komplett geladen und ist danach funktionsbereit. Weiterhin enthält diese Klasse die Callback-Funktionen für alle möglichen SIP-Events. Hier wird entschieden, welche Aktionen beim Auftreten der verschiedenen SIP-Ereignisse unternommen werden.

Eine Übersicht über die Callback-Funktionen bietet die Klasse `CallListenerAdapter`, die Elternklasse von UserAgent:

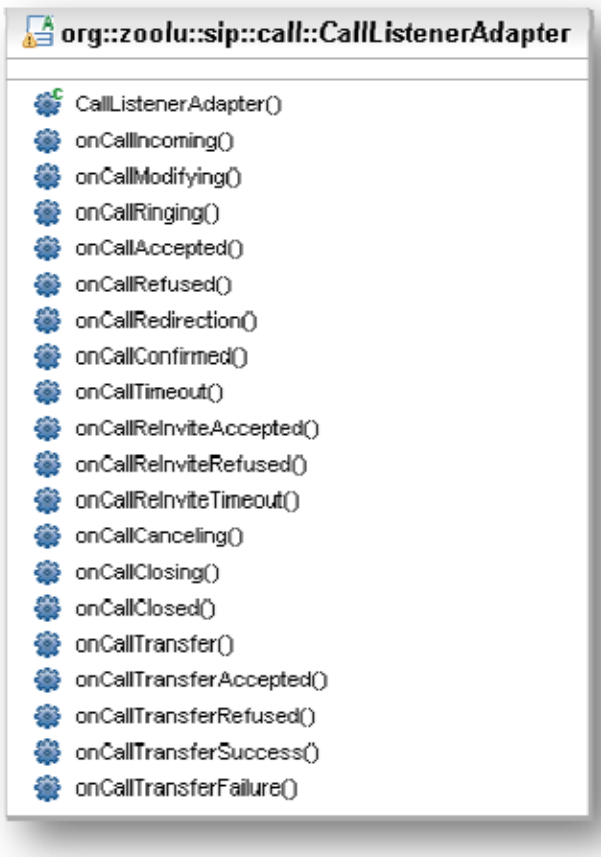


Abbildung 21: Methoden der Klasse CallListenerAdapter

Ein paar der möglichen SIP-Ereignisse sind z.B.

- ein eingehender Anruf: `onCallIncoming()`
- der Gesprächspartner hat den Anruf akzeptiert bzw. abgehoben: `onCallAccepted()`
- der Gesprächspartner hat nicht abgehoben: `onCallTimeout()`
- der Gesprächspartner möchte das Gespräch beenden / hat aufgelegt: `onCallClosing()`

3.2.2.2.2 UserAgentProfile

In der Klasse `UserAgentProfile` werden alle Einstellungen abgespeichert. Hier kann das Verhalten des VoIP-Clienten angepasst werden. Besonders wichtig sind hier die Zugangsdaten, die benötigt werden, um sich beim SIP-Provider anzumelden. Dazu gehören:

- der Realm (Netzwerk des SIP Providers, z.B. sipgate.de)
- die eigene SIP-Adresse (z.B. sip:4969123456@sipgate.de)
- die eigene IP-Adresse, welche bei Verbindungen über Router benötigt wird
- die Kontakt-Adresse für eine direkte Verbindung in der Form sip:nummer@IP-Adresse:Port
- Benutzername und Passwort des SIP-Kontos

Die Einstellungen können direkt im Quellcode bei der Zuweisung der Variablen vorgenommen werden oder man legt eine Konfigurationsdatei an, welche dann von der Klasse `UserAgentProfile` geladen wird.

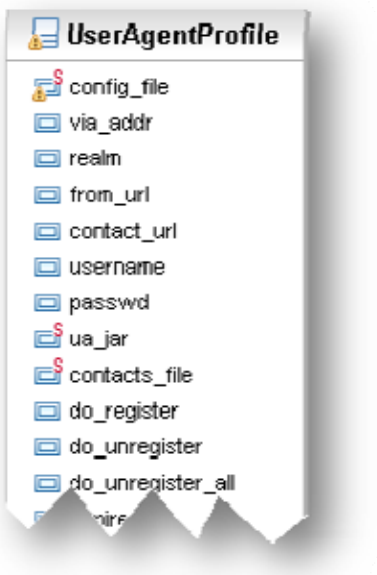


Abbildung 22: Eigenschaften der Klasse UserAgentProfile (Auszug)

3.2.2.2.3 RtpStreamSender

Die Klasse `RtpStreamSender` ist für das Übertragen der Multimedia-Datenströme mittels des Realtime Transfer Protocols (RTP) zuständig.

Die ausgehenden Datenströme werden in Pakete aufgeteilt, die dann anschließend, mit den nötigen Header-Informationen versehen, über das Netz geschickt werden. Im Header ist u. A. die Zieladresse enthalten.

Weil das Versenden der Pakete ein kontinuierlicher Prozess ist, läuft dieser in einem eigenen Thread ab. So kann auf Daten gewartet werden, ohne dass der Ablauf des Hauptprogramms behindert wird.

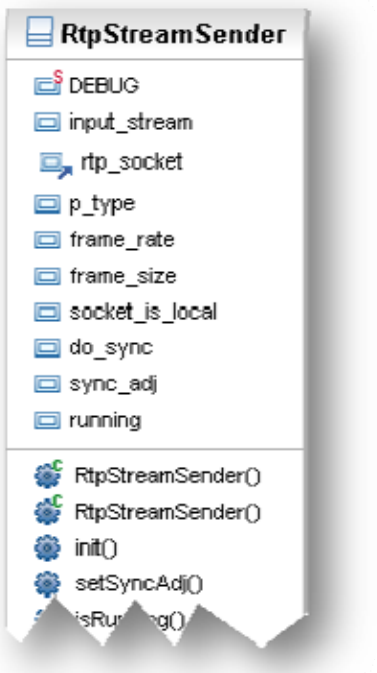


Abbildung 23: Eigenschaften und Methoden der Klasse RtpStreamSender (Auszug)

3.2.2.2.4 RTPStreamReceiver

Das Gegenstück zum `RtpStreamSender` ist die Klasse `RtpStreamReceiver`. Hier werden die Header der ankommenden RTP-Pakete entfernt und der Inhalt wird wieder als Datenstrom bereitgestellt.

Das Empfangen der Pakete läuft ebenfalls in einem eigenen Thread ab.

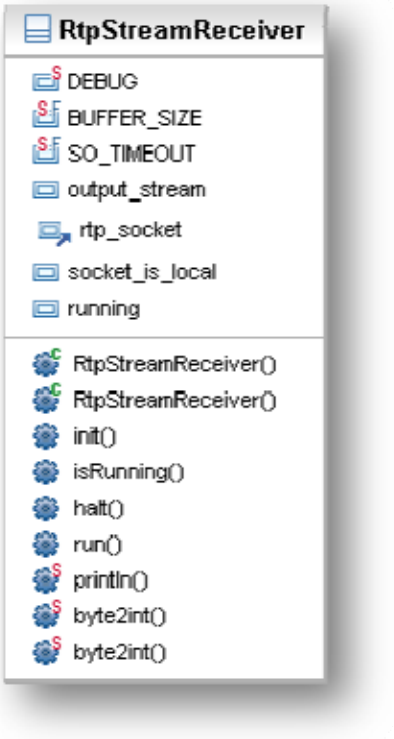


Abbildung 24: Eigenschaften und Methoden der Klasse RtpStreamReceiver

3.2.2.2.5 Weitere wichtige Klassen

In Tabelle 11 sind noch ein paar weitere Klassen aufgelistet, die ebenfalls eine zentrale Rolle spielen.

Klasse	Aufgabe
RegisterAgent	Zuständig für die Anmeldung beim SIP-Provider. Die Anmeldung kann periodisch wiederholt werden, damit die Verbindung erhalten bleibt.
JAudioLauncher	Initialisiert die Audio-Ein-/Ausgabe
AudioInput	Liest Daten vom Audio-Eingang
AudioOutput	Wiedergabe der empfangenen Audiodaten

Tabelle 11: Weitere wichtige Klassen des VoIP-Clienten

3.3 Vorbereitungen

3.3.1 Einleitung

Bevor mit der Entwicklung begonnen werden kann, sind noch ein paar Modifikationen an der Softwareumgebung nötig. Die externen Programme, welche vom Sprachserver benötigt werden, müssen installiert werden. Des Weiteren muss die Umgebungsvariable „Path“ angepasst werden, damit die externen Programme gefunden werden können. Schließlich ist für die Entwicklung mit Eclipse noch ein „Projekt“ zu erstellen.

3.3.2 Installieren der externen Programme

3.3.2.1 MBROLA

Die MBROLA Sprachsynthese kann auf der MBROLA Internetseite[11] im Bereich Download bezogen werden. Kompilierte Binärdateien sind für diverse Betriebssysteme verfügbar. In diesem Projekt werden die Binärdateien für PC/Windows verwendet.

Die Datei „MbrolaTools35.exe“ wird auf den Computer heruntergeladen und durch Doppelklick gestartet. Es erscheint der Begrüßungsbildschirm des Installers:

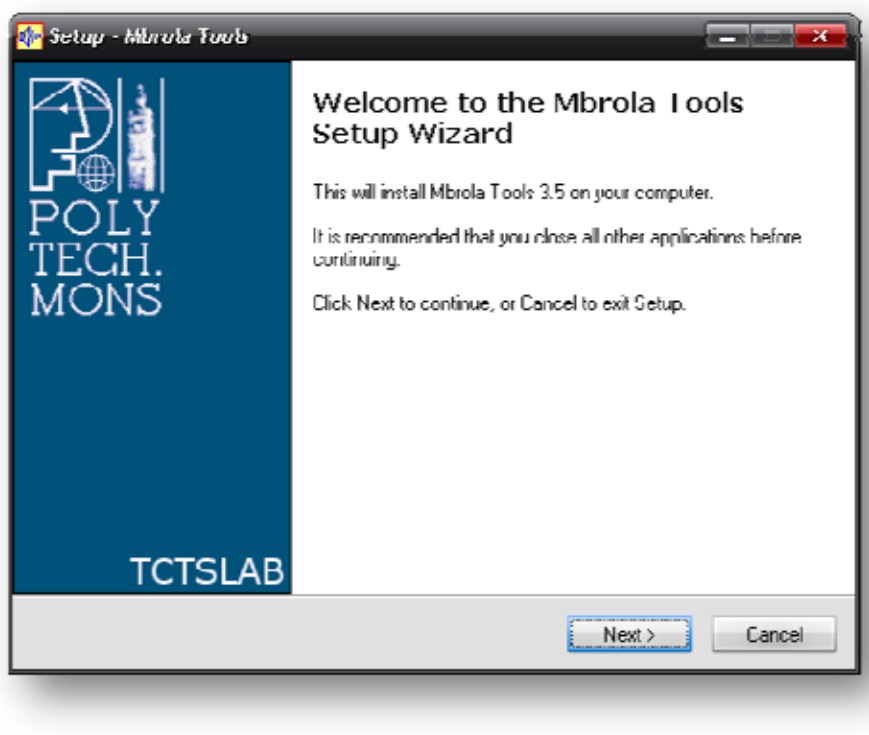


Abbildung 25: Begrüßungsdialog beim MBROLA Setup

Nach einem Klick auf die Schaltfläche „Next“ und dem anschließenden Akzeptieren der Lizenzvereinbarung kann der Installationspfad angegeben werden. Die Voreinstellung ist „C:\Programme\Mbrola Tools“ und kann übernommen werden. Alle weiteren Dialoge können mit einem Klick auf die Schaltfläche „Next“ beendet werden, bis die Schaltfläche „Install“ erscheint. Mit Klick auf diese wird die Installation abgeschlossen.

Als nächstes wird noch mindestens eine Stimmdatenbank für MBROLA benötigt. Diese befinden sich auch im Downloadbereich der MBROLA-Seite. Es ist darauf zu achten, eine deutsche Stimme zu wählen. Davon gibt es genau acht Stück, welche davon gewählt wird ist beliebig. Für dieses Projekt wurde die Stimme „de5“ ausgewählt. Es können auch mehrere Stimmen installiert werden. Die verwendete Stimme kann später noch angepasst werden.

[11] URL: http://tcts.fpms.ac.be/synthesis/mbrola.html

Die Archivdatei mit der Stimmdatenbank muss mit einem geeigneten Programm entpackt werden. Der Speicherort ist beliebig, hier wurde das Verzeichnis „C:\Database“ verwendet. Der Pfad zur Stimmdatenbank muss später in der Konfigurationsdatei „fbsv.ini“ angegeben werden, z.B. mit der Zeile „mbrolaDB=c:/database/de5/de5“.

3.3.2.2 Txt2pho

Das Programm Txt2pho kann von dessen Homepage[12] heruntergeladen werden. Hier wird ebenfalls die Version für Windows gewählt. Die Datei „tx2phdll.zip“ wird anschließend in ein temporäres Verzeichnis entpackt und mit einem Doppelklick auf die Datei „setup.exe“ wird der Installer gestartet.

Abbildung 26: Begrüßungsdialog des Txt2pho-Setups

Nachdem die Schaltfläche „Weiter“ ein paar mal gedrückt wurde, kann auch hier der Installationspfad angegeben werden. Standardmäßig wird Txt2Pho im Verzeichnis „C:\Programme\Txt2pho“ installiert. Nachdem der Pfad bestätigt worden ist, wird mit dem Betätigen der Schaltfläche „Weiter“ die Installation durchgeführt.

[12] URL: http://www.ikp.uni-bonn.de/dt/forsch/phonetik/hadifix/HADIFIXforMBROLA.html

3.3.2.3 HTK (Hidden Markov Toolkit)

Das Hidden Markov Toolkit kann von der HTK-Seite[13]bezogen werden. Bevor man jedoch etwas von der Seite herunterladen kann, ist eine kostenlose Registrierung nötig. Die aktuelle Version 3.4 gibt es nur als Quellcode, aber für die Version 3.3 sind vorkompilierte Binärdateien für Windows erhältlich.

Die Datei „htk-3.3-windows-binary.zip“ muss auf den Rechner heruntergeladen und mit einem geeigneten Programm entpackt werden. Als Zielverzeichnis kann z.B. „C:\Programme\HTK“ angegeben werden.

Nach dem Entpacken befinden sich im angegebenen Verzeichnis eine größere Anzahl von Programmen (EXE-Dateien). Die Installation des Toolkits ist damit abgeschlossen, es muss nur noch das verwendete Verzeichnis in der Umgebungsvariable „Path“ angegeben werden.

3.3.3 „Path“ anpassen

Bei Windows-Betriebssystemen gibt es eine Umgebungsvariable mit dem Namen „Path“. Sie spezifiziert alle Verzeichnisse, die nach einem ausführbaren Programm durchsucht werden, falls dieses nicht im aktuellen Verzeichnis gefunden werden kann. Damit die externen Programmaufrufe vom Demonstrator erfolgreich durchgeführt werden können, müssen die Verzeichnisse mit den verwendeten Programmen in „Path“ angegeben werden.

In Windows XP kann die Umgebungsvariable „Path“ folgendermaßen geändert werden:

Auf das Symbol „Arbeitsplatz“ mit der rechten Maustaste klicken. Im Kontextmenü den Punkt „Eigenschaften“ wählen. Die Systemeinstellungen können alternativ auch in der Systemsteuerung durch einen Doppelklick auf das Symbol „System“ aufgerufen werden.

Abbildung 27: Aufruf des Dialogs „Systemeigenschaften“

[13] URL: http://htk.eng.cam.ac.uk/

Im Dialog Systemeigenschaften den Karteireiter „Erweitert“ wählen. Hier befindet sich im unteren Drittel ein Knopf mit der Aufschrift „Umgebungsvariablen“:

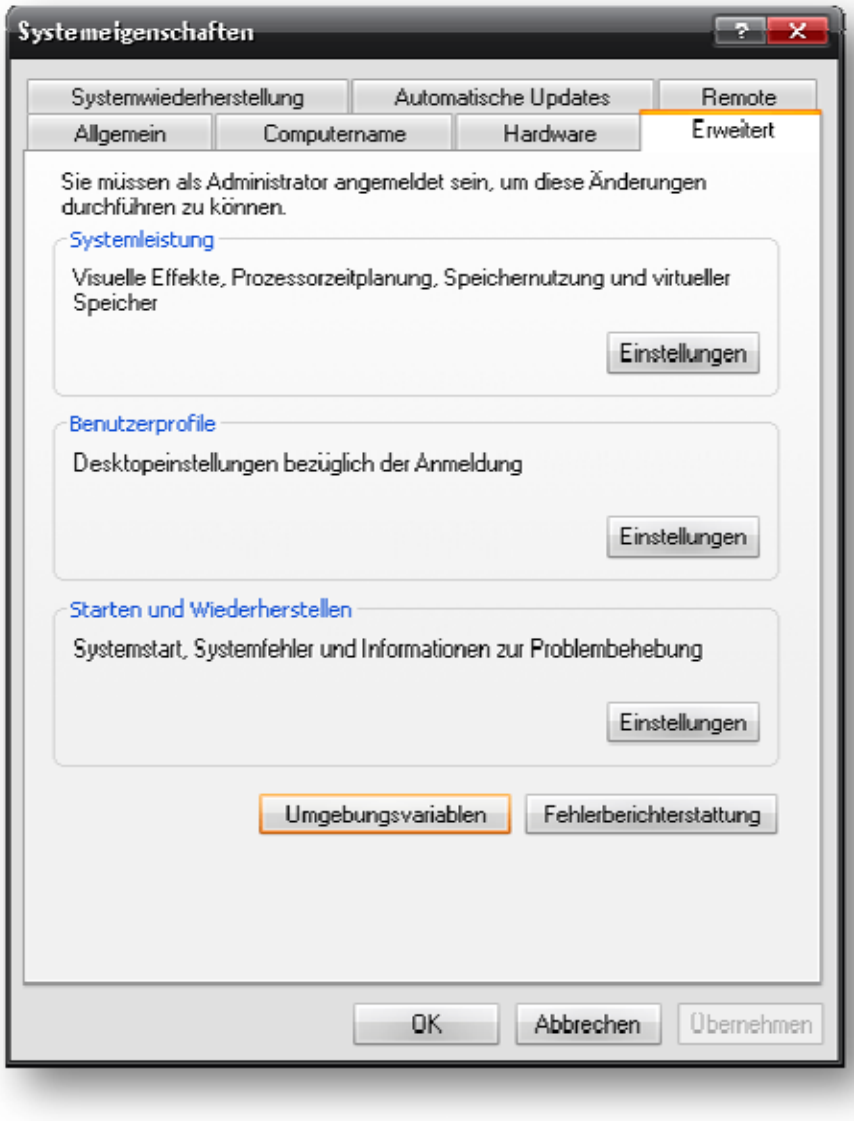


Abbildung 28: Ansicht „Erweitert“ im Dialog „Systemeigenschaften“

Im Dialog „Umgebungsvariablen“ die Variable „Path“ aus der Liste mit einem Mausklick auswählen und den Knopf „Bearbeiten“ darunter betätigen:

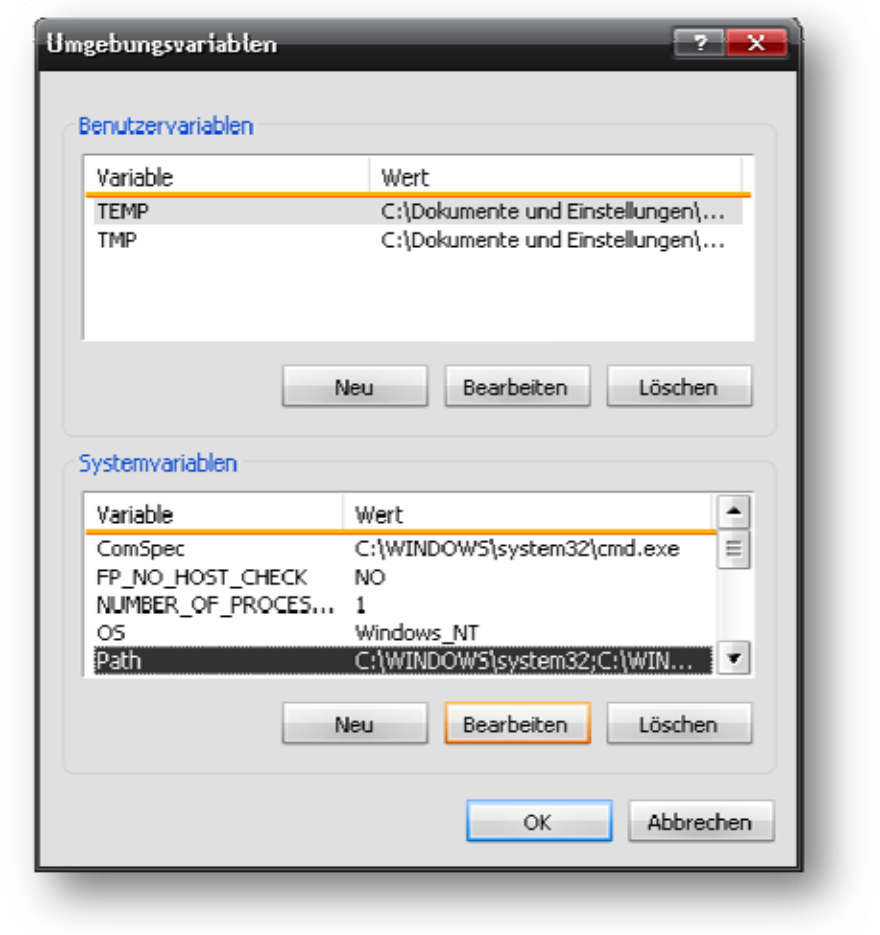


Abbildung 29: Dialog „Umgebungsvariablen“

Es können nun im unteren Feld die zusätzlichen Verzeichnisse angefügt werden. Hier ist zu beachten, dass die Einträge jeweils durch einen Strichpunkt getrennt werden.

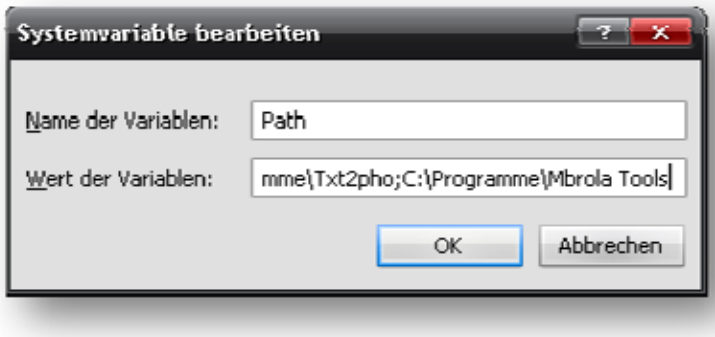


Abbildung 30: Dialog zur Bearbeitung einer Systemvariablen

Auf dem Entwicklungssystem wurde die Variable „Path“ um folgende Einträge erweitert:

- C:\Programme\HTK
- C:\Programme\Txt2pho
- C:\Programme\Mbrola Tools

Die tatsächlichen zu verwendenden Einträge hängen vom Installationsordner der externen Programme ab.

3.3.4 Erstellen eines Eclipse-Projektes

Bevor der Quellcode mit Eclipse bearbeitet werden kann, muss ein Projekt erstellt werden. In der folgenden Beschreibung werden beide Programmteile (FBSV und VoIP-Client) in einem Projekt untergebracht.

Zunächst werden die zwei Archive mit dem Quellcode der Programme in ein temporäres Verzeichnis entpackt. Hier wurde dafür das Verzeichnis „C:\Temp\Eclipse Setup“ gewählt. Nach dem Entpacken befindet sich der Quellcode des Sprachservers im Unterordner „fbsv“ und der des VoIP-Clienten im Unterordner „VoIP-Client“.

Als nächstes wird Eclipse gestartet. Falls bisher noch kein Projekt erstellt wurde, so ist der Begrüßungs-Bildschirm zu sehen, welcher über das Symbol „Go to workbench“ unten rechts beendet werden kann. In der Hauptansicht wird nun der kleine Pfeil neben dem Symbol „New“, welches sich ganz links in der Symbolleiste befindet, angeklickt. Aus dem Drop-Down-Menü wird der erste Eintrag „Java Project“ gewählt.

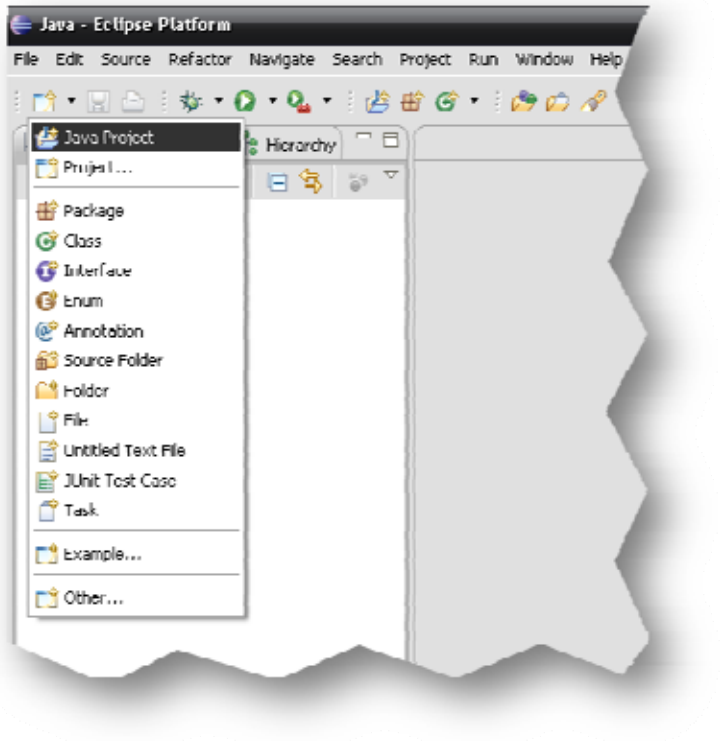


Abbildung 31: Das Drop-Down-Menu „New“ in Eclipse

In dem Dialog, welcher sich daraufhin öffnet, wird im Feld ganz oben „demonstrator“ als Name des Projektes eingegeben. Die anderen Einstellungen können auf den Standardwerten belassen werden. Durch Betätigen der Schaltfläche „Finish“ wird der Dialog wieder verlassen.

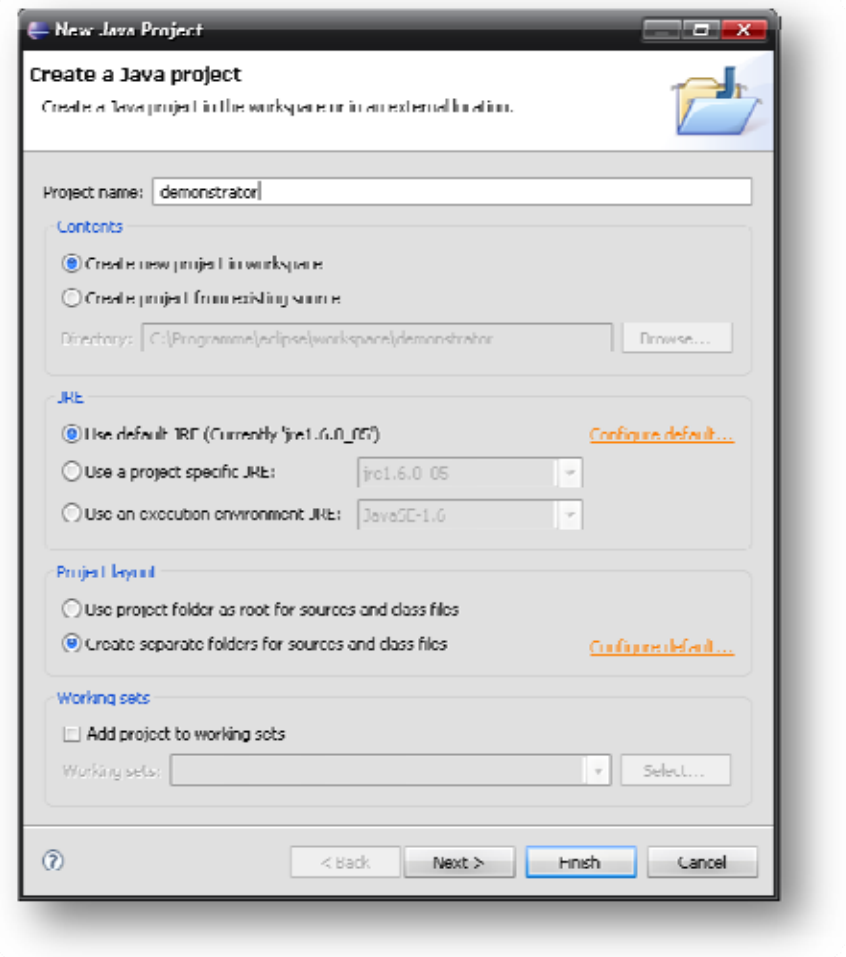


Abbildung 32: Dialog zum Erstellen eines neuen Java Projektes

Das soeben erstellte Projekt ist bis jetzt noch leer. Es müssen zuerst die Quelldateien importiert werden. Dazu wird mit der rechten Maustaste auf den Ordner „demonstrator“ in der Ansicht „Package Explorer“ geklickt und aus dem Kontextmenü der Eintrag „Import…“ ausgewählt.

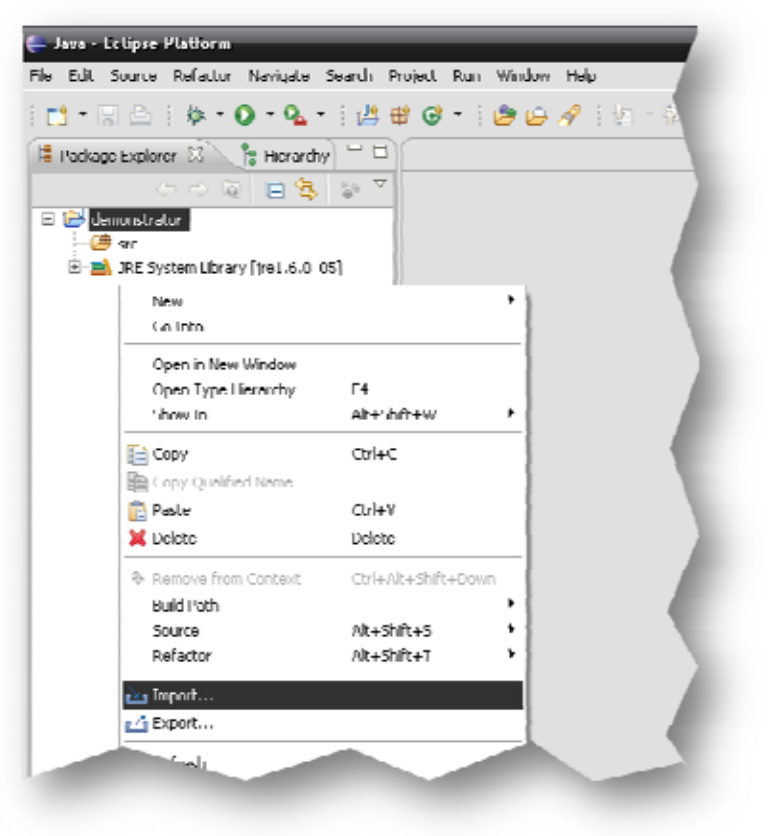


Abbildung 33: Import des Quellcodes über das Kontextmenü

In dem nun erscheinenden Dialogfenster wird der Ast „General“ durch Klick auf das Plus-Zeichen daneben expandiert und der Unterpunkt „File System“ ausgewählt. Anschließend wird der Dialog über die Schaltfläche „Next“ fortgesetzt.

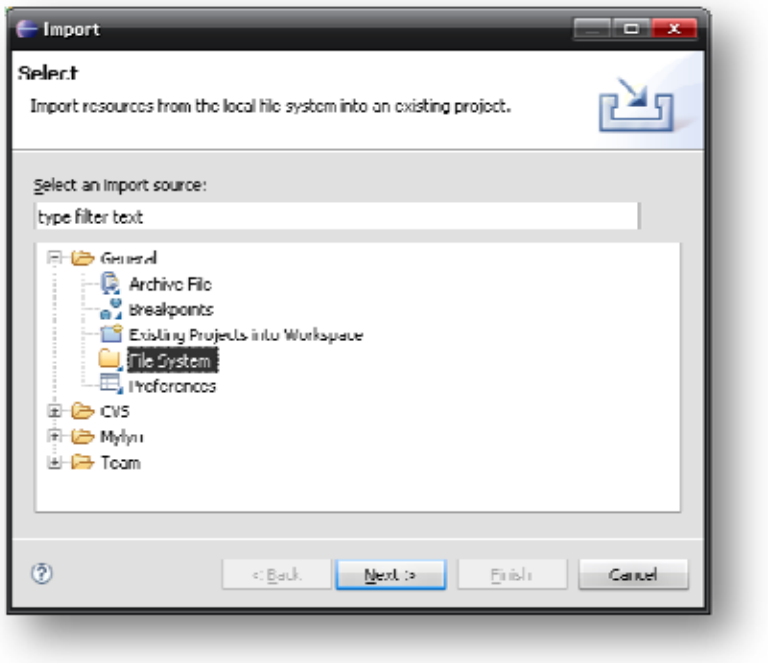


Abbildung 34: Der Import-Dialog Teil 1

Hier muss nun zuerst das Verzeichnis angegeben werden, aus dem die Dateien importiert werden sollen. Dazu ist auf die obere „Browse“-Schaltfläche zu klicken und das Verzeichnis auszuwählen (hier „C:\Temp\Eclipse Setup\VoIP-Client“). Jetzt kann eine Auswahl der Dateien getroffen werden. Um das Projekt übersichtlich zu halten werden nur die benötigten Dateien ausgewählt.

Der Ast „VoIP-Client“ wird wie gewohnt durch einen Klick auf das Plus-Symbol erweitert und die Unterordner „lib“ und „src“ werden durch setzen des Häkchens markiert. Danach wird der Hauptordner markiert und die Datei „contacts.lst“ auf der rechten Seite ausgewählt.

Im unteren Textfeld steht der Zielordner „demonstrator“ (es gibt bisher noch keine anderen Ordner im Projekt). Der Dialog kann nun über die Schaltfläche „Finish“ beendet werden.

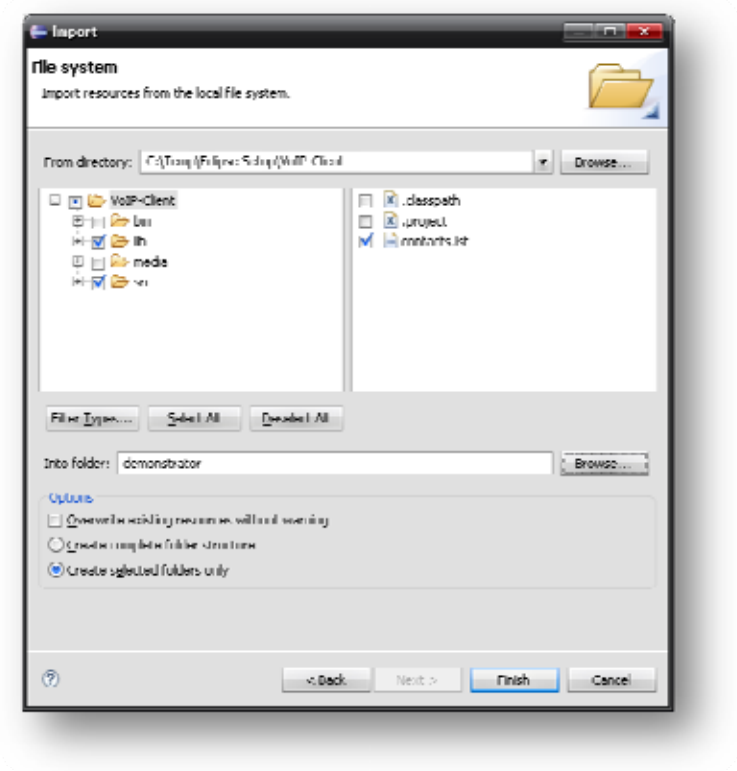


Abbildung 35: Der Import-Dialog Teil 2

Nach dem Import werden einige Dateien wahrscheinlich mit einem roten Kästchen, in dem sich ein „x“ befindet, markiert sein. Dieses „x“ weist auf Fehler im Quellcode hin. Der Grund für die hier gemeldeten Fehler liegt darin, dass Teile des VoIP-Clienten das JMF (Java Media Framework) verwenden. Ist dieses nicht installiert, so können die benötigten Abhängigkeiten nicht aufgelöst werden.

Zur Lösung des Problems werden die Klassen, welche das JMF verwenden, deaktiviert (diese werden vom Demonstrator nicht gebraucht). Dazu müssen sechs Klassen (wie in der folgenden Abbildung zu sehen) markiert werden. Durch Linksklick mit gedrückter STRG-Taste können mehrere Dateien ausgewählt werden. Nun wird das Kontextmenü mit einem Rechtsklick aufgerufen. Hier wird nun die Maus über dem Eintrag „Build Path“ positioniert, wodurch sich ein Untermenü mit dem Punkt „Exclude“ öffnet. Nach einem Klick auf „Exclude“ werden die entsprechenden Dateien beim kompilieren nicht mehr berücksichtigt.

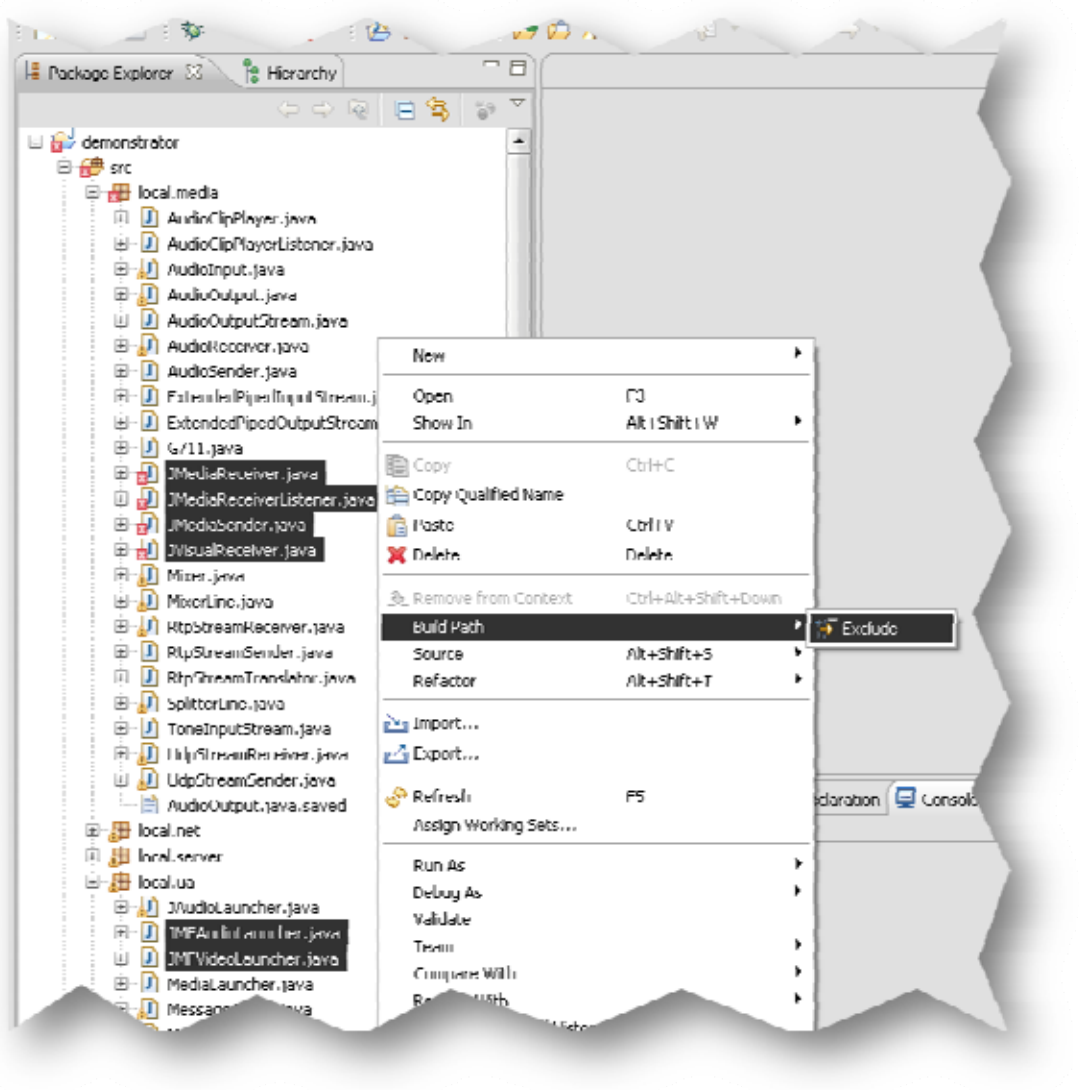


Abbildung 36: Ausschließen bestimmter Klassen

Danach sind die Kästchen mit dem „x“ verschwunden und der VoIP-Client kann kompiliert werden.

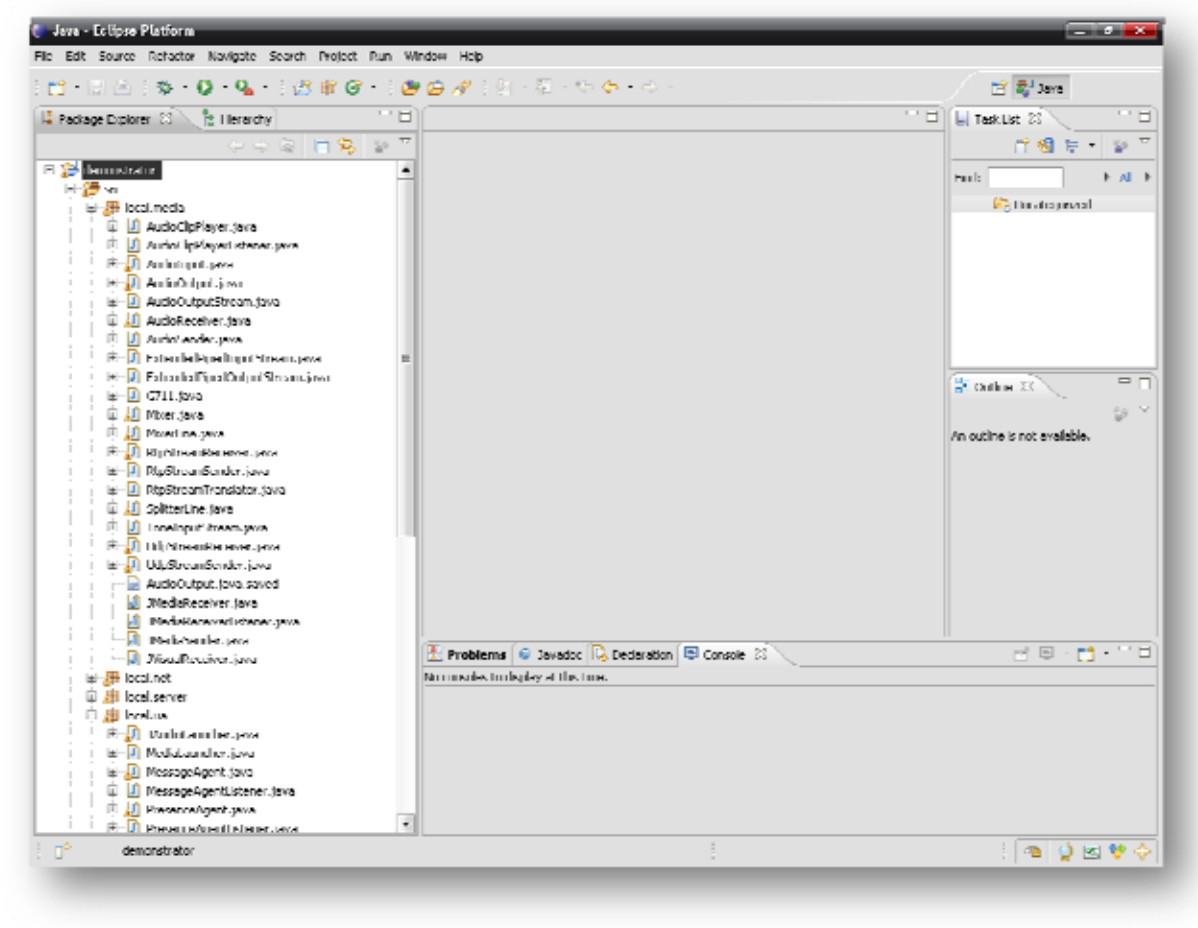


Abbildung 37: Der VoIP-Client wurde erfolgreich importiert

Damit der VoIP-Client ausgeführt werden kann, muss noch ein neues Verzeichnis „log“ im Projektordner angelegt werden. Dies kann durch Rechtsklick auf den Hauptordner „demonstrator“ im „Package Explorer“ getan werden. Im Kontextmenü ist „New“ und dann „Folder“ auszuwählen. Alternativ kann der Projektordner auch im Windows Explorer geöffnet werden und der neue Ordner wie gewohnt angelegt werden.

Bevor mit dem Import des Sprachserver-Quellcodes begonnen wird, werden vorher ein paar Aufräumarbeiten durchgeführt. Dazu wird der Ordner mit dem Quellcode im Windows Explorer geöffnet (hier: „C:\Temp\Eclipse Setup\fbsv“). Danach wird ein Verzeichnis mit dem Namen „fbsv“ erstellt. Nun werden alle .java und .jar-Dateien aus dem aktuellen Verzeichnis in das neue Verzeichnis verschoben. Bis auf die Dateien „fbsv.ini“, „TXT2PHO.INI“, „vxml.dtd“ und „vxmlfiles.txt“ werden alle verbleibenden Dateien im aktuellen Ordner gelöscht.

Nun kann der Quellcode genauso wie beim VoIP-Clienten in Eclipse importiert werden. Durch setzen des Häkchens vor dem Hauptordner werden alle Dateien und Unterordner selektiert. Zu beachten ist hier, dass im unteren Feld der Ordner „demonstrator“ gewählt sein muss. Falls ein anderer Ordner gewählt ist, muss die Schaltfläche „Browse“ neben dem Feld betätigt werden und im anschließenden Dialog der Hauptordner „demonstrator“ gewählt werden. Der Dialog kann nun über die Schaltfläche „Finish“ beendet werden.

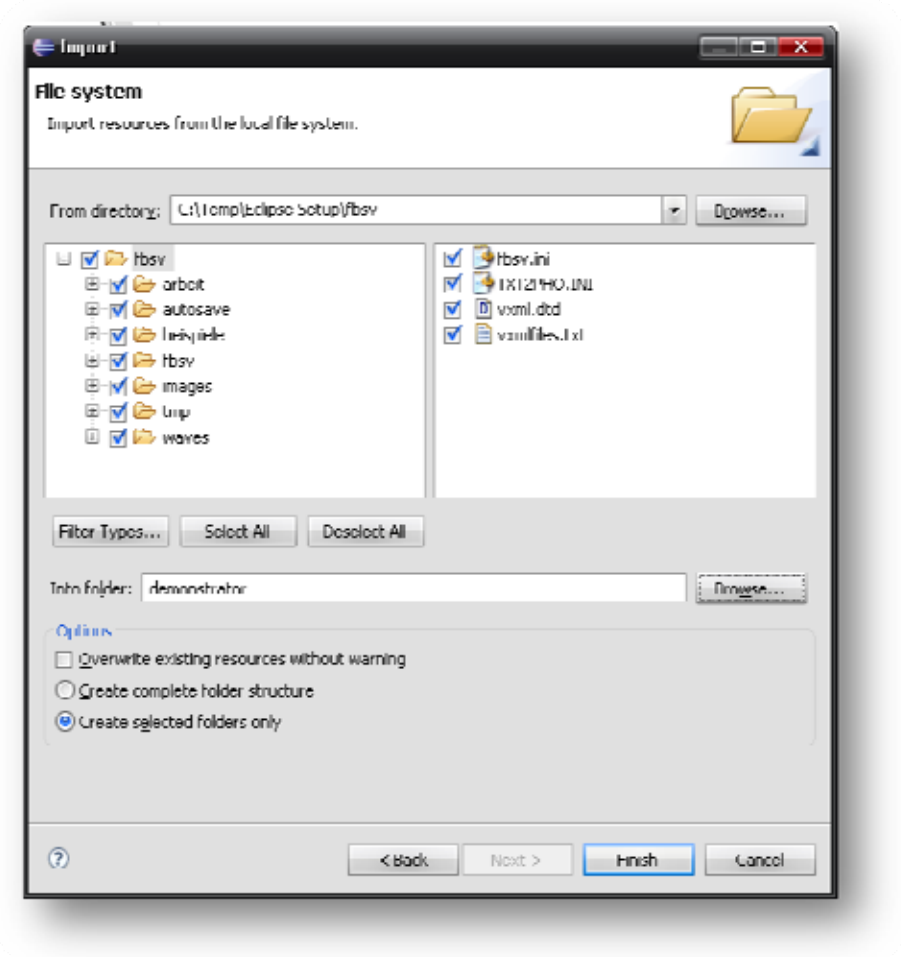


Abbildung 38: Import des Sprachserver-Quellcodes

Der Ordner „src“ beim VoIP-Clienten wurde automatisch als Quellcode-Ordner erkannt. Der Ordner „fbsv“ muss jedoch manuell als Quellcode festgelegt werden. Dazu wird mit einem Rechtsklick auf den Ordner „fbsv“ im „Package Explorer“ das Kontextmenü geöffnet. Hier wird die Maus auf dem Element „Build Path“ platziert. In dem sich nun öffnenden Untermenü wird der Eintrag „Use as source folder“ ausgewählt.

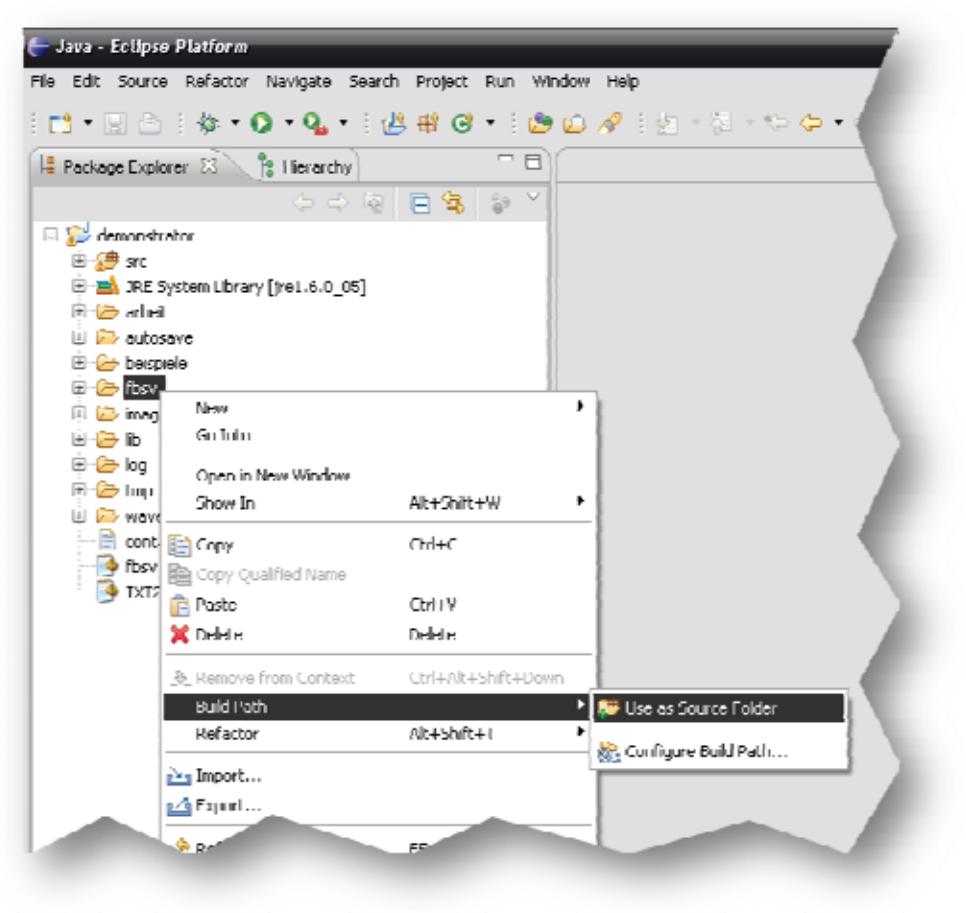


Abbildung 39: Festlegen des Ordners „fbsv“ als Quellcode

Anschließend werden wieder einige Klassen mit einem „x“ markiert sein. Die zwei Bibliotheken „js.jar“ und „utils.jar“ müssen noch eingebunden werden. Dies geschieht, indem beide .jar-Dateien im Ordner „fbsv“ markiert werden und mit Rechtsklick das Kontextmenü aufgerufen wird. Hier nun den Eintrag „Build Path“ und dann „Add to Build Path“ auswählen.

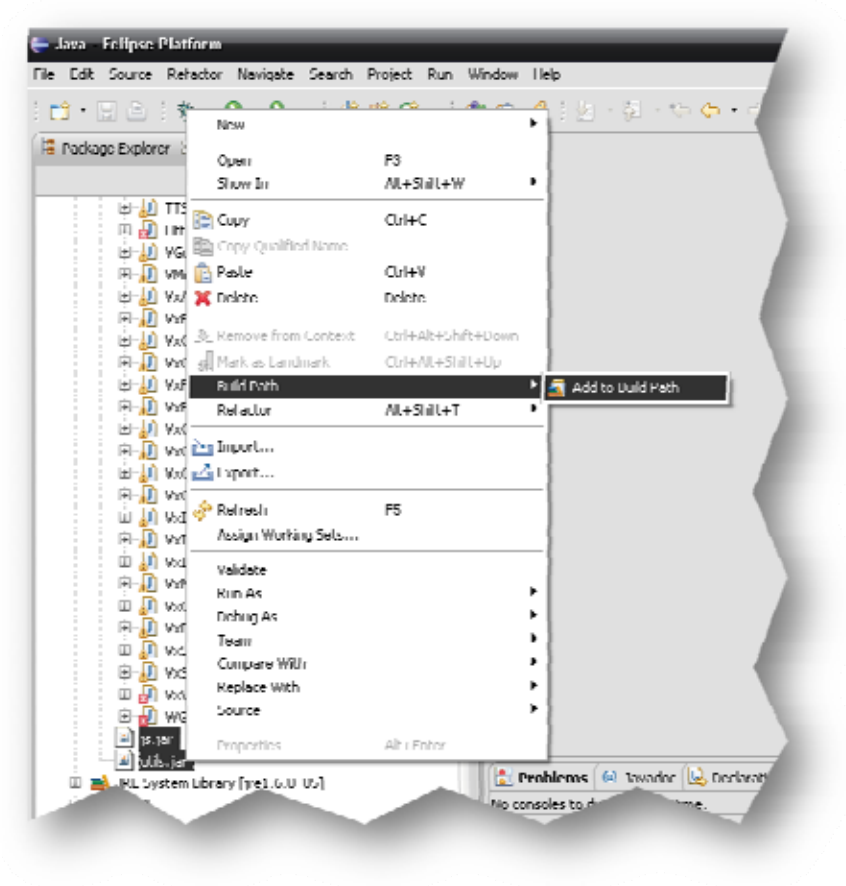


Abbildung 40: Einbindung der Bibliotheken

Mit diesem Schritt ist das Eclipse-Projekt vollständig erstellt und es kann mit der Bearbeitung des Quellcodes begonnen werden.

3.3.5 VoIP-Zugang besorgen

Um den VoIP-Client verwenden zu können wird ein Zugang bei einem VoIP-Anbieter benötigt. Die Auswahl ist recht groß. Auf der Seite OnlineKosten.de[14] befindet sich eine Liste aller Anbieter in Deutschland. Zusätzlich finden sich dort Angaben über die Gebühren und Leistungen der einzelnen Anbieter.

In Hinsicht auf dieses Projekt sollte der Anbieter folgende Voraussetzungen erfüllen:

- keine Grundgebühr
- kein Mindestumsatz
- inkl. Telefonnummer, die aus dem Festnetz erreichbar ist
- Telefonnummer zum Ortstarif
- Unabhängig vom Internet-Provider

Ein Anbieter, der diese Bedingungen erfüllt, ist Sipgate[15]. Der Autor hat sich dazu entschlossen, diesen Anbieter für die Entwicklung des Demonstrators zu verwenden.

[14] URL: http://www.onlinekosten.de/voip/anbieter/uebersicht
[15] URL: http://www.sipgate.de

3.4 Verbinden der Komponenten

3.4.1 Audio-Eingabe über VoIP

3.4.1.1 Analyse des Datenflusses

3.4.1.1.1 Einleitung

Die wichtigste Eigenschaft des Demonstrators ist die Spracheingabe über VoIP. Lässt sich diese Eigenschaft realisieren, so kann man bereits abschätzen, ob das Projekt machbar ist. Aus diesem Grund ist der erste Schritt die Umleitung der über VoIP empfangenen Sprachdaten an den Spracherkenner. Dazu muss untersucht werden, wie der Fluss der Daten im VoIP-Client verläuft und wo sich eine geeignete Stelle zum Abgriff der eingehenden Audiodaten befindet.

3.4.1.1.2 Datenfluss VoIP-Client

Am Anfang steht ein Paketstrom, der vom VoIP-Clienten über das Netzwerk empfangen wird. VoIP benutzt, wie schon im Grundlagenteil erwähnt, das Real Time Transfer Protokoll (RTP) zur Übertragung der Medienströme. Der Empfang der RTP-Pakete wird von Klasse `RtpStreamReceiver` durchgeführt. Es handelt sich dabei um eine Ableitung der Klasse `Thread`. Ein Thread ist ein Ausführungsstrang, der parallel zum Hauptprogramm läuft und dabei meistens eine kontinuierliche Arbeit in einer Schleife verrichtet. Die entscheidende Methode der Klasse `Thread` ist `run()`, in der die Tätigkeit des Ausführungsstrangs festgelegt ist.

In der Methode `run()` des `RtpStreamReceivers` wird zuerst ein Buffer angelegt, in dem später ein eingehendes Datenpaket abgespeichert wird:

```
byte[] buffer = new byte[BUFFER_SIZE];
RtpPacket rtp_packet = new RtpPacket(buffer,0);
```

In einer `while`-Schleife wird nun kontinuierlich auf ein eingehendes Datenpaket gewartet. Die Anweisung dafür lautet folgendermaßen:

```
rtp_socket.receive(rtp_packet);
```

Nachdem ein Paket empfangen worden ist, wird der Inhalt des Paketes in eine Instanz von `OutputStream` kopiert. Die Daten aus dem Paketheader werden verworfen:

```
output_stream.write(rtp_packet.getPacket(),
rtp_packet.getHeaderLength(), rtp_packet.getPayloadLength());
```

Anschließend beginnt die Ausführung der Schleife von Vorne und es wird auf das nächste Paket gewartet. Der Paketstrom wird so in einen kontinuierlichen Datenstrom zurückgewandelt.

Als nächstes muss die soeben erwähnte Instanz der Klasse `OutputStream` verfolgt werden. Diese wurde dem `RtpStreamReceiver` bereits bei der Instanziierung übergeben, daher muss nach der Stelle gesucht werden, wo der `RtpStreamReceiver` erstellt wird. Die entsprechende Stelle findet man in der Klasse `JAudioLauncher`:

```
AudioFormat format = new AudioFormat(codec, sample_rate,
8*sample_size, 1, sample_size, sample_rate, big_endian);

audio_output = new AudioOutput(format);

receiver = new RtpStreamReceiver(audio_output.getOuputStream(),
socket);
```

Hier wird eine Instanz der Klasse `AudioOutput` gebildet. Dabei wird auch das Format angegeben, in dem die Audiodaten vorliegen. Auf das Format der Audiodaten wird im späteren Verlauf noch eingegangen. Der `OutputStream` wird dem `RtpStreamReceiver` folglich von der Klasse `AudioOutput` mit Hilfe der Methode `getOutputStream()` zur Verfügung gestellt.

Betrachtet man die Klasse `AudioOutput`, so stellt man fest, dass dort die eigentliche Audioausgabe erfolgt. Dies geschieht mit Hilfe der Methode `AudioSystem.getLine()`. Die Klasse `AudioSystem` ist Teil von Java und stellt Methoden zur Verfügung, um die Audio-Hardware anzusprechen. Das Ergebnis ist eine Instanz der Klasse `SourceDataLine`[16]. Mit Hilfe der Methode `write()` dieser Instanz kann man nun Audiodaten ausgeben lassen. Auch der `OutputStream` wird hier angelegt:

```
audio_output_stream = new AudioOutputStream(source_line, format);
```

Es ist zu erkennen, dass es sich bei diesem `OutputStream` um eine spezialisierte Form mit dem Namen `AudioOutputStream` handelt. Diese Klasse ist Bestandteil des VoIP-Clienten. Der Instanz des `AudioOutputStreams` wird die `SourceDataLine` übergeben. Somit wurde die Verbindung zwischen den ankommenden Paketen bis hin zur Audioausgabe hergestellt. Allerdings lässt sich bisher noch keine geeignete Stelle finden, an der man an die Audiodaten auch tatsächlich herankommt.

Nun ist es an der Zeit, die Klasse `AudioOutputStream` genauer zu untersuchen. Im Konstruktor der Klasse `AudioOutputStream` findet man folgende Zeilen:

```
this.source_line = source_line;

PipedInputStream piped_input_stream = new PipedInputStream();
output_stream = new PipedOutputStream(piped_input_stream);

AudioInputStream audio_input_stream = new AudioInputStream( pi-
ped_input_stream, format, -1);
```

[16] Eigentlich würde man unter Source (engl., die Quelle) eine Audioeingabe vermuten (z.B. Mikrofon). Aus der Sichtweise des Audio-Mixers handelt es sich aber um eine Datenquelle.

In der ersten Zeile wird hier der bei der Instanziierung übergebene Verweis auf die Audio-Ausgabe (`SourceDataLine`) festgehalten. Danach wird ein `PipedInputStream` angelegt, der mit einem `PipedOutputStream` verbunden wird. Dadurch entsteht eine sogenannte Pipe[17]. Das ist ein gepufferter Datenstrom zwischen zwei Prozessen. So kann der Schreib- und Leseprozess voneinander entkoppelt werden. Die Daten werden in den `PipedOutputStream` geschrieben und können dann von einem anderen Thread aus dem `PipedInputStream` gelesen werden.

Des Weiteren findet dort auch eine Umwandlung zwischen den Audioformaten statt:

```
input_stream = AudioSystem.getAudioInputStream(
source_line.getFormat(), audio_input_stream);
```

Das Zielformat ist im ersten und die Quelle im zweiten Parameter angegeben. Damit die Audiodaten auch korrekt ausgegeben werden können, muss das Format mit dem der Audioausgabe übereinstimmen. Das Eingabeformat ist beim verwendeten VoIP-Clienten üblicherweise 8 Bit ULAW und das Ausgabeformat 16 Bit PCM. Die Konversion wird von der bereits erwähnten Klasse `AudioSystem` durchgeführt, welche ein Bestandteil von Java ist.

Da es sich bei der Klasse `AudioOutputStream` um eine Ableitung der Klasse `OutputStream` handelt, findet man dort auch die `write()`-Methoden, welche die Methoden der Oberklasse überschreiben. Es gibt insgesamt drei dieser Methoden, jeweils mit unterschiedlichen Parametern. Es muss nur die `write()`-Methode für ein einzelnes Byte implementiert werden, die anderen können durch mehrfaches Aufrufen dieser einen Methode nachgeahmt werden[18]. Im Folgenden wird nur die Methode mit einem Byte als Parameter behandelt.

Vorausgesetzt, die oben erwähnte Pipe wurde erfolgreich erstellt, so wird das Byte `b` in die Pipe geschrieben und die konvertierten Daten werden anschließend aus der Pipe gelesen, welche dann auf der Audiohardware ausgegeben werden:

```
output_stream.write(b);
int len = input_stream.read(buff, 0, buff.length);
source_line.write(buff, 0, len);
```

Diese Stelle ist auch ideal dazu geeignet, die konvertierten Daten für den Spracherkenner abzuzweigen. Der Datenfluss ist im folgenden Diagramm nochmal zusammenfassend dargestellt:

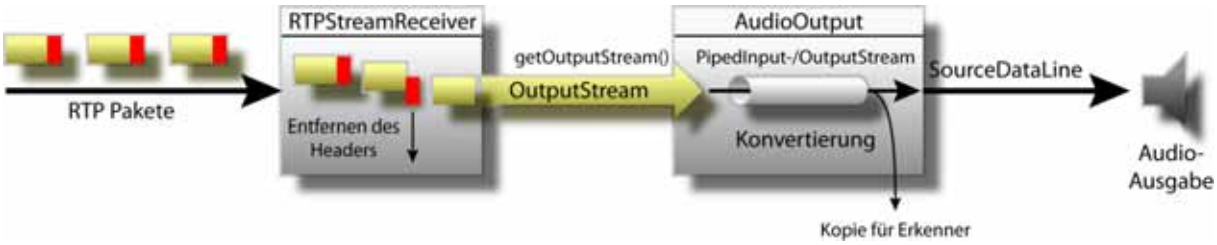


Abbildung 41: Der Weg der Audiodaten im VoIP-Clienten

[17] engl. für Rohr, Röhre
[18] Die Implementierung der anderen write()-Methoden ist sinnvoll um die Performance zu optimieren.

3.4.1.1.3 Datenfluss Sprachserver

Nachdem der Weg der Daten im VoIP-Client bekannt ist, muss untersucht werden, wie der Sprachserver die Audiodaten bezieht. Für die Spracherkennung ist die Klasse `Recognizer` zuständig. In der Methode `recognize()` findet man folgende Zeile:

```
boolean response = wgd.isUtt(timeout);
```

Hier wird der Wortgrenzendetektor angewiesen, eine Äußerung aufzuzeichnen. Wie schon bei der Beschreibung der Komponenten erwähnt, ist es auch die Aufgabe der Klasse WGD, die Audiodaten zu beziehen.

Die Aufzeichnung wird in der Methode `isUtt()` der Klasse `WGD` durch folgenden Aufruf eingeleitet:

```
jaudio.startRecording();
```

Mit Hilfe der Methode `readSamples()`der Klasse `JAudio` können später Audiodaten gelesen werden:

```
buff = jaudio.readSamples();
```

Die Aufgabe wird hier an die Klasse `JAudio` weitergereicht. In deren Konstruktor befinden sich diese Anweisungen:

```
audioFormat = new AudioFormat(sampleRate, 16, 1, true, true);
DataLine.Info info = new DataLine.Info( TargetDataLine.class,
audioFormat);
[…]
line = (TargetDataLine) AudioSystem.getLine( info );
line.open( audioFormat );
```

Hier wird die Hardware für die Audioeingabe angesprochen, d. h. der Audioeingang wird geöffnet. Die Klasse `TargetDataLine` ist das Gegenstück zu `SourceDataLine`. Die Referenz auf eine Instanz von `TargetDataLine` wird wieder von der Klasse `AudioSystem` bereitgestellt. Ebenso wird ein Format übergeben, in dem später die Audiodaten zur Verfügung stehen.

In der Methode `readSamples()` können anschließend Daten von der Audioquelle gelesen:

```
numBytesRead = line.read(data, 0, data.length);
```

Die Variable `data` ist hier ein Byte-Array in dem die gelesenen Daten abgespeichert werden.

Damit ist auch bekannt, wie der Sprachserver die Audiodaten bezieht.

3.4.1.2 Umleitung der Daten

3.4.1.2.1 Konzept

Da nun die Datenquelle sowie die neue Datensenke bekannt sind, müssen Überlegungen dazu angestellt werden, wie man die Daten am besten dorthin umleitet. Eine Möglichkeit wäre, die Klasse `JAudio` direkt von dem `PipedInputStream` aus der Klasse `AudioOutputStream` lesen zu

lassen. Der Nachteil dabei ist, dass dann keine Ausgabe mehr über die Lautsprecher erfolgt. Die Ausgabe über die Lautsprecher ist aber durchaus sinnvoll um eine Rückmeldung über das übertragene Audiosignal zu erhalten und somit evtl. auftretende Probleme besser erkennen zu können. Außerdem kann es zu einem Pufferüberlauf kommen, denn dabei werden nur Daten vom Puffer eingelesen, wenn der Wortgrenzendetektor gerade aktiv ist.

Eine Alternative ist es, Daten aus dem `PipedInputStream` in einen zusätzlichen Puffer zu kopieren, aus dem die Klasse `JAudio` dann die Daten lesen kann. Falls der Wortgrenzendetektor inaktiv ist, können die ankommenden Daten verworfen werden.

Hierzu wird eine eigene Klasse mit dem Namen `JAudioInBuffer` angelegt. Voraussichtlich wird für den umgekehrten Prozess, die Umleitung des lokalen Audioeingangs zur Übertragung über VoIP, eine ähnliche Klasse benötigt, die dann den Namen `JAudioOutBuffer` erhält. Die Klasse `JAudioInBuffer` implementiert einen einfachen Ringpuffer und die notwendigen Methoden um Daten in den Puffer zu schreiben oder aus dem Puffer zu lesen.

3.4.1.2.2 Grundlagen Ringpuffer

Ein Ringpuffer ist eine Warteschlange nach dem FIFO (First In First Out)-Prinzip. Er besteht aus einem Speicherbereich mit einer festen Größe, einem Zeiger, der den Anfang der Nutzdaten markiert, und einem Zeiger, der das Ende der Nutzdaten markiert.

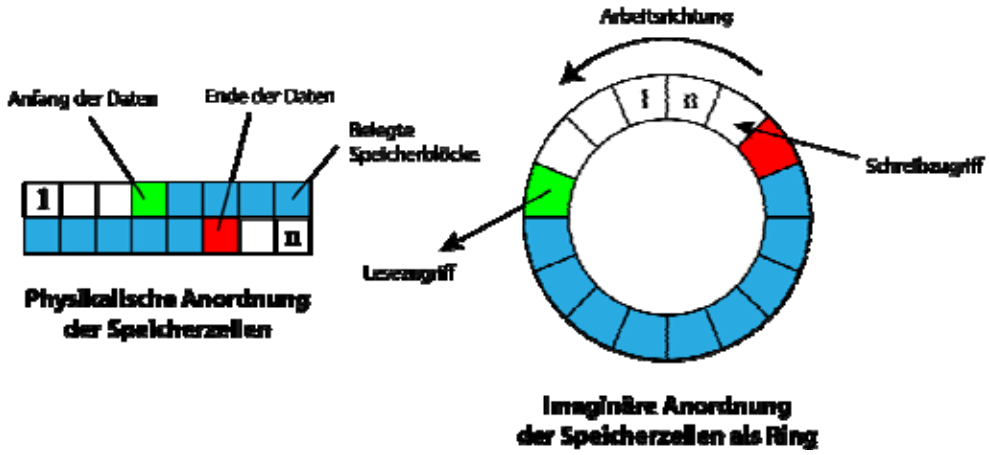


Abbildung 42: Funktionsweise eines Ringpuffers

Schreibvorgang:

Werden Daten in den Puffer geschrieben, so erhöht sich die Endposition. Ist die letzte Speicherzelle erreicht, so wird als nächstes wieder die erste Speicherzelle verwendet. Ist der Puffer voll belegt, so werden ältere Daten überschrieben. In diesem Fall erhöht sich auch die Startposition. Die ältesten Daten gehen dabei verloren.

Lesevorgang:

Beim Lesen der Daten wird bei der Startposition begonnen. Wurde eine Speicherzelle gelesen, dann erhöht sich die Startposition. Nach der letzten Speicherzelle wird der Vorgang wieder bei der ersten Speicherzelle fortgesetzt. Wenn die Startposition und die Endposition auf dieselbe Speicherzelle zeigen, so ist der Puffer leer und es können keine weiteren Daten gelesen werden.

3.4.1.2.3 Die Klasse JAudioInBuffer

Da sich die Klasse `JAudioInBuffer` zum Teil so verhalten soll wie ein `OutputStream`, wird sie als Ableitung von dieser Klasse implementiert:

```
public class JAudioInBuffer extends OutputStream
```

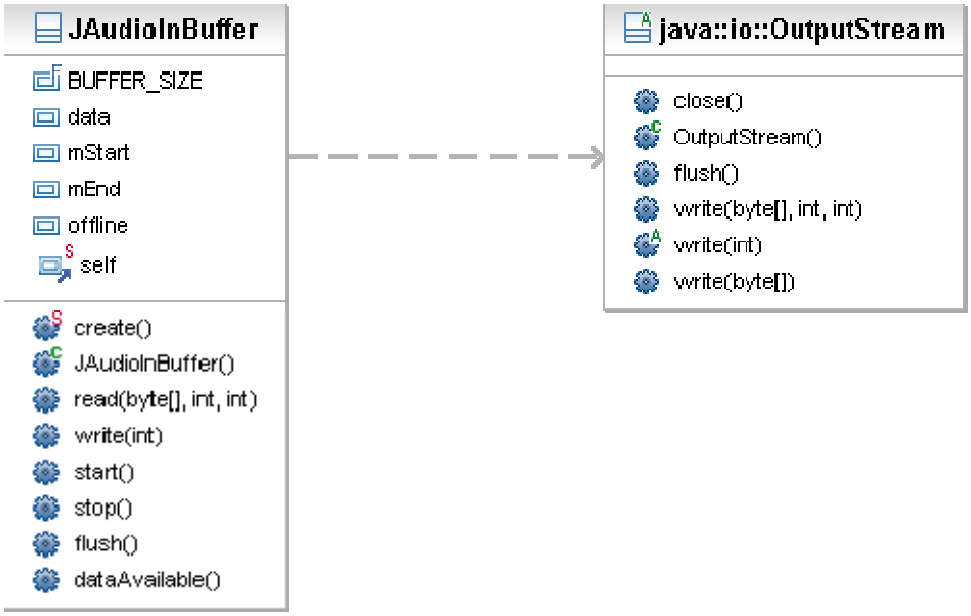


Abbildung 43: Die Klasse JAudioInBuffer als Ableitung von OutputStream

Außerdem kommt das Singleton Pattern[19] zum Einsatz. Damit wird sichergestellt, dass nur eine Instanz des Objektes angelegt werden kann und ein zusätzlicher Vorteil besteht darin, dass von überall auf den Puffer zugegriffen werden kann, ohne dass eine Referenz durchgereicht werden muss.

Die Variablen:

```
final int BUFFER_SIZE = 64000;
private byte[] data = new byte[BUFFER_SIZE];
private int mStart = 0;
private int mEnd = 1;
private boolean offline = true;
private static JAudioInBuffer self = null;
```

Als erstes wird hier eine konstante Puffergröße mit 64000 Bytes festgelegt. Danach wird der Puffer erstellt. Die Integer-Werte `mStart` und `mEnd` sind die Zeiger auf den Anfang und das Ende der Nutzdaten. Mit dem Bool-Wert `offline` kann der Puffer abgeschaltet werden. Die Variable `self` ist Teil des Singleton-Patterns und enthält eine Referenz auf den Puffer selbst, sofern er instanziiert wurde.

[19] Das **Singleton** (auch **Einzelstück** genannt) ist ein in der Softwareentwicklung eingesetztes Entwurfsmuster [Wik088]. Ein **Entwurfsmuster** (engl. *design pattern*) beschreibt eine bewährte Lösungs-Schablone für ein Entwurfsproblem. Es stellt damit eine wiederverwendbare Vorlage zur Problemlösung dar, die in einem spezifischen Kontext einsetzbar ist [Wik089].

Der Konstruktor:

Gemäß dem Singleton Pattern ist der Konstruktor der Klasse nicht sichtbar. Wird eine Instanz des Puffers gebildet, so wird eine Referenz auf dieses Objekt abgespeichert.

```
private JAudioInBuffer (){
      self = this;
}
```

Der Aufruf des Konstruktors ist nur über die statische Methode `create()` möglich:

```
static public JAudioInBuffer create (){
      if (self != null)
            return self;
      else
            return new JAudioInBuffer();
}
```

Existiert bereits ein Puffer, so wird die Referenz auf den existierenden Puffer zurückgegeben, andernfalls wird ein neuer Puffer angelegt und dessen Referenz zurückgegeben.

Die Methoden:

Der Schreibzugriff geschieht mit der Methode `write()`. Da die Klasse `JAudioInBuffer` eine Ableitung von `OutputStream` ist, muss nur die Methode `write()` für ein einzelnes Byte implementiert werden. Die Methoden um mehrere Bytes in den Puffer zu schreiben werden automatisch von der Oberklasse vererbt.

```
public void write (int data){
      if (offline) return;

      this.data[mEnd] = (byte)data;
      mEnd = (mEnd + 1) % BUFFER_SIZE;

      if (mEnd == mStart)
            mStart = (mStart + 1) % BUFFER_SIZE;
}
```

Ist der Puffer im Offline-Modus, d. h. er ist abgeschaltet, so können keine Daten in den Puffer geschrieben werden, andernfalls wird ein Byte an der aktuellen Endposition eingefügt. Die Endposition wird dann um eins erhöht. Durch die Modulo-Operation % springt die Endposition automatisch auf den Anfang des Speicherbereichs, wenn das Ende erreicht ist. Befinden sich die Start- und Endposition nach der Schreiboperation an derselben Stelle, so wurden ältere Daten überschrieben und die Startposition muss ebenfalls erhöht werden.

Lesezugriff ist in der Oberklasse `OutputStream` nicht vorgesehen, deshalb sind eigentlich alle gängigen `read()`-Methoden selbst zu implementieren. Allerdings reicht es aus, die Methoden zu implementieren, die auch tatsächlich verwendet werden.

In der Methode `readSamples()` der Klasse `JAudio` findet man folgenden Aufruf:

```
numBytesRead = line.read(data, 0, data.length);
```

Hier kann man erkennen, dass die Methode `read(byte[] b, int off, int len)` der Klasse `TargetDataLine` mit den Parametern Daten-Array, Offset und Länge verwendet wird. Es ist sinnvoll, die `read()`-Methode mit denselben Parametern zu implementieren.

```
public int read (byte[] dataOut, int offset, int length){
      int numRead = 0;

      while (numRead < length){
            if (mStart == mEnd)
                  break;
            else {
                  dataOut[numRead] = this.data[mStart];
                  numRead ++;
                  mStart = (mStart + 1) % BUFFER_SIZE;
            }
      }
      return numRead;
}
```

Beim Aufruf der Methode wird das Array `dataOut` übergeben, in dem die gelesenen Daten abgelegt werden können. Mit dem Integer-Wert `offset` kann angegeben werden, wie viele Bytes übersprungen werden sollen. Da der `offset`-Wert in diesem Projekt nicht verwendet wird, bleibt die Angabe vorerst ohne Effekt. Der dritte Parameter `length` gibt an, wie viele Bytes maximal gelesen werden sollen. Der Wert `length` ist üblicherweise gleich der Größe des übergebenen Arrays.

Der Leseprozess läuft in einer Schleife ab. Diese wird solange durchlaufen, bis die Anzahl der gelesenen Bytes nicht mehr geringer ist als die geforderte Anzahl. Falls die Anfangs- und Endposition des Ringpuffers identisch sind, so ist der Puffer leer und die Ausführung der Schleife wird beendet. Befinden sich noch Daten im Puffer, so wird ein Byte aus dem Puffer in das Array `dataOut` kopiert. Der Lesezähler wird um eins erhöht und ebenso die Anfangsposition des Puffers. Analog zur `write()`-Methode wird hier auch der Modulo-Operator % eingesetzt, um die Anfangsposition auf Null zurückzusetzen, sofern das Ende des Pufferspeichers erreicht ist.

Neben den Methoden für den Lese- und Schreibzugriff gibt es noch Methoden, mit denen der Zustand des Puffers kontrolliert werden kann. In Anlehnung an die Klasse `TargetDataLine` heißen diese `start()`, `stop()` und `flush()`. Mit dem Aufruf von `start()` wird der Puffer aktiviert und mit `stop()` deaktiviert. Ist der Puffer aktiviert, so können Daten gespeichert werden, andernfalls werden alle Schreibzugriffe ignoriert. Die Methode `flush()` löscht den Inhalt des Puffers.

Zuletzt gibt es noch die Methode `dataAvailable()`, die die Anzahl der Bytes im Puffer zurückliefert.

3.4.1.2.4 Kopieren der Daten in den Puffer

Die `write()`-Methoden der Klasse `AudioOutputStream` müssen nun so abgeändert werden, dass die ankommenden Audiodaten in den neuen Puffer kopiert werden.

Hier als Beispiel die Änderungen in der `write()`-Methode mit einem Byte als Parameter (Änderungen in grüner Farbe):

```
public void write(int b) throws IOException
{
      if (output_stream!=null)
      {
            output_stream.write(b);
            int len=input_stream.read(buff,0,buff.length);
            source_line.write(buff,0,len);
            JAudioInBuffer.create().write(buff,0,len);
       }
       else
       {
            buff[0]=(byte)b;
            source_line.write(buff,0,1);
            JAudioInBuffer.create().write(buff,0,1);
       }
}
```

Der Aufruf von `create()` liefert eine Referenz auf den Puffer zurück und die Schreiboperation für die Audioausgabe wird einfach auf dem Puffer wiederholt.

3.4.1.2.5 Lesen der Daten aus dem Puffer

Die Klasse `JAudio` muss nun so abgeändert werden, dass die Audiodaten aus dem Puffer gelesen werden. Als erstes wird bei der Variablendeklaration ein Zeiger auf den Puffer angelegt:

```
TargetDataLine  line    = null;
JAudioInBuffer  aBuffer = JAudioInBuffer.create();
```

Die Methoden `startRecording()` und `stopRecording()` müssen nun erweitert werden, damit auch der Zustand des Puffers entsprechend gesetzt wird:

```
public void startRecording(){
      if( audioMode ) {
            line.flush();
            line.start();
            aBuffer.flush();
            aBuffer.start();
      }
}
public void stopRecording(){
      if( audioMode ) {
            line.stop();
            aBuffer.stop();
      }
}
```

Schließlich wird in der Methode `readSamples()` die Audioquelle geändert:

```
// numBytesRead = line.read( data, 0, data.length );
numBytesRead = aBuffer.read( data, 0, data.length );
```

Die Umleitung des über VoIP eingehenden Audiosignals an den Spracherkenner ist nun abgeschlossen. Allerdings gibt es noch zwei Probleme, die erst behoben werden müssen, bevor die Spracherkennung über VoIP funktioniert.

3.4.1.3 Behebung der aufgetretenen Probleme

3.4.1.3.1 Buffer Underflow

Beim Start des Demonstrators besteht noch keine VoIP-Verbindung und daher ist der Puffer der Klasse `JAudioInBuffer` leer. Der Wortgrenzendetektor versucht aber sofort Daten aus dem Puffer zu lesen, wobei der Lesevorgang den Wert Null zurückliefert. Dies führt im weiteren Verlauf zu Ausnahmefehlern und die Ausführung des Wortgrenzendetektors wird beendet.

Eine vorläufige Lösungsmöglichkeit ist die Modifikation der Methode `readSamples()` (Klasse `JAudio`):

```
numBytesRead = aBuffer.read( data, 0, data.length );

if (numBytesRead == 0)
   numBytesRead = line.read( data, 0, data.length );
```

Damit später die Möglichkeit besteht, zwischen lokaler Ein-/Ausgabe und VoIP umzuschalten, wurde der Zugriff auf die Audiohardware komplett beibehalten. Ist der Puffer leer, weil z.B. noch keine VoIP-Verbindung besteht, kann einfach auf den lokalen Audioeingang zurückgegriffen werden.

Allerdings stellte sich heraus, dass durch diese Methode die Qualität des Audiosignals bei einer VoIP-Verbindung negativ beeinflusst wird. Durch die unterschiedliche Laufzeit der Datenpakete (Jitter) läuft der Puffer auch während einer Verbindung von Zeit zu Zeit leer und so entstehen Aussetzer im Audiosignal.

Eine bessere Möglichkeit ist es, den Wortgrenzendetektor solange warten zu lassen, bis sich Daten im Puffer befinden. Dies kann z.B. durch die Methode `Thread.sleep()` erreicht werden. Der Aufruf von `Thread.sleep()` veranlasst das Betriebssystem, den aktuellen Ausführungsstrang (Thread) zu unterbrechen und die Ausführung erst nach einer angegebenen Zeit wieder aufzunehmen.

Die Warteanweisung muss vor jedem Zugriff auf den Puffer ausgeführt werden. Deshalb wird sie in der Methode `readSamples()` der Klasse `JAudio` unmittelbar vor dem Zugriff auf den Puffer eingefügt:

```
byte[] data = new byte[BUFFSIZE];

while (aBuffer.dataAvailable() == 0){
      try {
            Thread.sleep(100);
      } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }
}
numBytesRead = aBuffer.read( data, 0, data.length );
```

Solange keine Daten im Puffer vorliegen, wird eine Schleife ausgeführt. Bei jedem Durchgang wird das Betriebssystem angewiesen, die Ausführung für 100 Millisekunden zu unterbrechen. Wurden in dieser Zeit Daten in den Puffer geschrieben, so wird die Schleife beim nächsten Durchgang abgebrochen und es kann mit dem Einlesen der Audiodaten begonnen werden. Die Methode `Thread.sleep()` kann die Ausnahme `InterruptedException` auslösen, wenn der Wartevorgang frühzeitig beendet wird. Deshalb ist es hier nötig den Aufruf mit einem try-catch-Block zu umgeben, um die potentielle Ausnahme abzufangen.

Das Problem des leeren Puffers ist damit behoben.

3.4.1.3.2 Unterschiede im Audio-Format

Das zweite Problem besteht darin, dass die empfangen Audiodaten in einem anderen Format vorliegen als in dem, welches der Spracherkenner erwartet und dieser somit das Audiosignal falsch zusammensetzt.

	Codierung	Vorzeichen	Byte-Reihenfolge	Abtastrate	Auflösung	Kanäle
Spracherk.	Linear PCM	Signed	Big Endian	16000 Hz	16 bit	Mono
VoIP-Client	Linear PCM	Signed	Little Endian	8000 Hz	16 bit	Mono

Tabelle 12: Audioformate von Spracherkenner und VoIP-Client

Die Formate unterscheiden sich in der Byte-Reihenfolge (Little/Big Endian) und in der Abtastrate (8000/16000 Hz). Bei der Byte-Reihenfolge kann entweder das niedriger-wertige Byte an der letzten Position stehen (Little Endian) oder das höher-wertige Byte (Big Endian). Wird die Reihenfolge falsch interpretiert, so erhält man einen anderen Zahlenwert.

Beispiel: Interpretation der Byte-Folge A3 04

$$\text{Little Endian: } A3\ 04 = (10 \times 16^3 + 3 \times 16^2) + (0 \times 16^1 + 4 \times 16^0) = 41732$$

$$\text{Big Endian: } A3\ 04 = (10 \times 16^1 + 3 \times 16^0) + (0 \times 16^3 + 4 \times 16^2) = 1187$$

Die Interpretation der Byte-Reihenfolge zu ändern ist nicht sehr aufwändig. Dazu muss nur die Stelle im VoIP-Client gesucht werden, wo das Ausgabe-Format festgelegt wird. Diese befindet sich in der Methode `initAudioLine()` der Klasse `AudioOutput`:

```
float fFrameRate=8000.0F;

AudioFormat format=new AudioFormat(
AudioFormat.Encoding.PCM_SIGNED, fFrameRate, 16, 1, 2, fFrameRate,
false);
```

Der letzte Parameter (hier `false`) gibt die Byte-Reihenfolge an. Der Wert `false` steht für Little Endian. Durch Änderung in `true` wird die Reihenfolge korrekt als Big Endian interpretiert.

Die Anpassung der Abtastrate gestaltet sich schwieriger. Grund: Zur Änderung der Abtastrate muss das Audiosignal neu Abgetastet werden (Resampling), dies unterstützt Java jedoch nicht von Haus aus.

Hier lässt sich ebenfalls für Testzwecke eine provisorische Lösung anbringen: Die vorhandenen Amplitudenwerte werden zweimal verwendet, damit sich die Anzahl von 8000 Werten pro Sekunde auf 16000 Werte pro Sekunde verdoppelt.

Dies geschieht in der bekannten Methode `jaudio.readSamples()`:

```
//samples = new short[ numBytesRead / 2 ];
samples = new short[ numBytesRead ];

int pb = 0;
for (int j=0; j < samples.length;  j++)
{
      int ix = (((data[pb] & 0xff) << 8) | (data[pb + 1] & 0xff));
      pb += 2;
      //samples[j] = (short) ix;
      samples[j] = samples[j+1] = (short) ix;
      j++;
}

[…]
```

Zuerst muss die Größe des Arrays `samples` verdoppelt werden, damit es die neue Anzahl der Werte erfassen kann. Dieses Array wird später von der Methode `readSamples()` zurückgegeben. Das Array `data` enthält die Audiodaten, die aus dem Audioeingang bzw. Puffer gelesen wurden. In der Schleife werden jeweils zwei Bytes aus dem Array `data` zu einem Short-Wert zusammengefasst, der dann im Array `samples` abgelegt wird. Für die Verdoppelung der Werte wird der Short-Wert auch an der nächsten Position im Array `samples` abgelegt. Dann wird der Zähler `j` um eins erhöht, damit die Kopie des Wertes beim nächsten Schleifendurchgang übersprungen wird.

Das erhaltene Sprachsignal ist immer noch gut verständlich, allerdings hört es sich verzerrt an, so als käme es z.B. aus einem Radio mit winzigen Lautsprechern.

Die Ursache für den Qualitätsverlust kann man im unteren linken Diagramm erkennen. Da die digitale Audioverarbeitung immer bemüht ist, eine Kurve durch alle Amplitudenwerte zu legen, entstehen kleine Wellenberge und Täler zwischen den Werten, die zu neuen Frequenzanteilen im Signal führen.

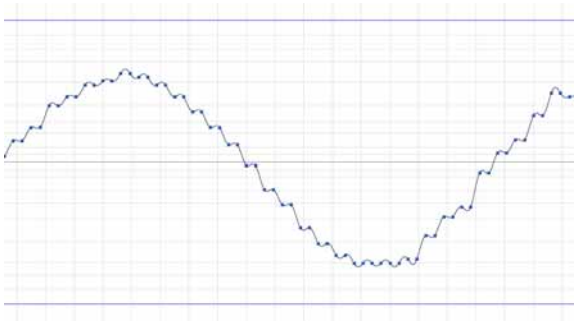


Abbildung 44: Zeitsignal (Werte dupliziert)

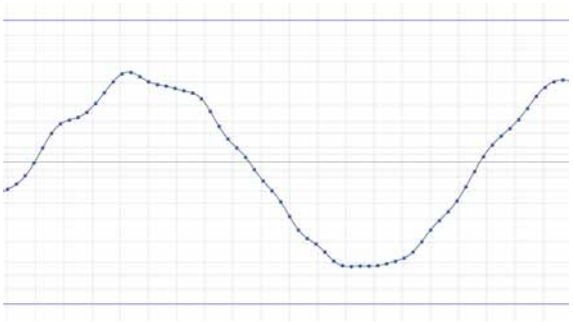


Abbildung 45: Zeitsignal (resampled)

Im Frequenz-Spektrum ist zu erkennen, dass sich die originalen Frequenzen an der Nyquist-Frequenz $f_{abt} \,/\, 2 = 8000\ Hz/2 = 4000\ Hz$ spiegeln. Durch Anwendung eines Tiefpassfilters kann der zusätzliche Frequenzanteil wieder entfernt werden. Ein ähnliches Verfahren wird auch beim Resampling angewendet.

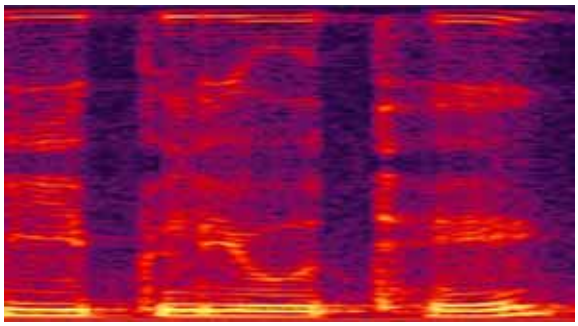


Abbildung 46: Frequenzspektrum (Werte dupliziert)

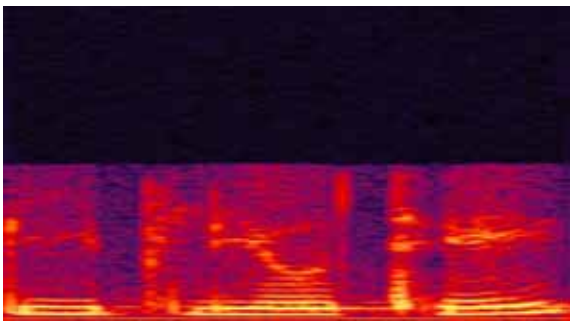


Abbildung 47: Frequenzspektrum (resampled)

Eine Recherche im Internet hat ergeben, dass Java auch zwischen verschiedenen Abtastfrequenzen konvertieren kann, wenn man eine zusätzliche Java-Bibliothek einbindet. Diese Bibliothek ist Teil des Tritonus-Projektes[20]. Bei Tritonus handelt es sich um eine Implementierung der Java Sound API. Die nötige Bibliothek findet man auf der Seite „Plug-Ins“ unter „Tritonus miscellaneous“.

Nach der Einbindung muss nur noch die Abtastfrequenz für die Ausgabe auf 16000 Hz in der Methode `audioOutput.initAudioLine()` angepasst werden:

```
//float fFrameRate=8000.0F;
float fFrameRate=16000.0F;

AudioFormat format=new AudioFormat(
AudioFormat.Encoding.PCM_SIGNED, fFrameRate, 16, 1, 2, fFrameRate,
true);
```

[20] URL: http://www.tritonus.org/

Die Audioformate stimmen jetzt überein und die Spracherkennung über VoIP funktioniert soweit. Allerdings stellt sich heraus, dass die Erkennungsgenauigkeit stark abgenommen hat. Das liegt daran, dass die Wortmodelle mit 16000 Hz erstellt worden sind und durch die Übertragung mit 8000 Hz ein gewisser Anteil des Frequenzbereichs verloren geht. Hier hilft nur noch das Neutrainieren des Spracherkenners mit Sprachdaten in einer Abtastfrequenz von 8000 Hz. Die nötigen Maßnahmen zur Anpassung werden an einem späteren Punkt beschrieben.

3.4.2 Audio-Ausgabe über VoIP

3.4.2.1 Einleitung

Etwas leichter als die Audioeingabe über VoIP lässt sich die Audioausgabe realisieren. Die Quelle für die Audioausgabe besteht hauptsächlich aus der Sprache, die von der Sprachsynthese ausgegeben wird. Weiterhin können auch Audiodateien direkt abgespielt werden.

3.4.2.2 Analyse des Datenflusses

Der Aufruf der Sprachsynthese geschieht in der Klasse `TTS`. Da der Demonstrator die MBROLA-Sprachsynthese verwendet, muss nur die Methode `playMBROLA()` genauer untersucht werden. Standardmäßig wird dort die Sprache direkt vom Programm phoplayer ausgegeben, dies ist jedoch ungünstig, weil dann keine Möglichkeit besteht, an die Audiodaten heranzukommen. Allerdings gibt es auch die Option, die Sprachausgabe in eine Datei schreiben zu lassen, die dann aus Java heraus abgespielt wird. Dazu muss nur die Variable `saveAudio` auf `true` gesetzt werden.

Auszug aus der Methode `playMBROLA()`:

```
if( saveAudio )
{
      cmdline = "phoplayer database=" + mbrolaDB + " /OUT=" +
      tmpWavFile + " /T=WAV " + tmpPhoFile;
      System.out.println( cmdline );
      p = Runtime.getRuntime().exec(cmdline);
      p.waitFor();
      (new VxAudio( "file:" + tmpWavFile )).play();
}
```

Ist `saveAudio` auf `true` gesetzt, dann wird der phoplayer angewiesen, die Ausgabe in einer WAV-Datei `tmpWavFile` zu speichern.

Das Abspielen der gespeicherten Datei übernimmt die Klasse `VxAudio`, von der eine temporäre Instanz gebildet wird und von dieser wird dann die Methode `play()` aufgerufen. Dort wird mit Hilfe der Klasse AudioSystem ein `AudioInputStream` erstellt:

```
AudioInputStream stream = AudioSystem.getAudioInputStream(new
URL(uri));
```

Von diesem `AudioInputStream` wird dann kontinuierlich gelesen und die erhaltenen Daten werden an die Audioausgabe `line` weitergeleitet:

```
int numRead = 0;
byte[] buf = new byte[line.getBufferSize()];

while ((numRead = stream.read(buf, 0, buf.length)) >= 0)
{
      int offset = 0;

      while (offset < numRead){
            offset += line.write(buf, offset, numRead-offset);
      }
}
```

Hiermit ist die Bezugsquelle für die Übertragung der Audiodaten bekannt. Jetzt muss nur noch das neue Ziel für die Umleitung gefunden werden. Analog zur Audioeingabe über VoIP, wo die Klasse `RtpStreamReceiver` die Datenpakete empfängt, sendet bei der Audioausgabe die Klasse `RtpStreamSender` die Pakete übers Netz.

Es handelt sich hierbei wiederum um eine Ableitung der Klasse `Thread`, bei der die Methode `run()` die Anweisungen für die kontinuierlich zu verrichtende Arbeit enthält. Genau wie beim `RtpStreamReceiver` wird hier zuerst ein Buffer für ein Paket angelegt:

```
byte[] buffer = new byte[frame_size+12];
RtpPacket rtp_packet = new RtpPacket(buffer,0);
```

Später wird in einer Schleife kontinuierlich von einem Eingabestrom (Audioeingang) gelesen und die Daten werden im Buffer des Paketes abgelegt, welches anschließend abgeschickt wird.

```
int num=input_stream.read(buffer, 12, buffer.length-12);

if (num > 0){
      rtp_socket.send(rtp_packet);
      try {Thread.sleep(frame_time);}
      catch (Exception e){}
}
```

Nun sind Quelle und Ziel bekannt und es kann wieder eine Klasse implementiert werden, um eine Verbindung zwischen diesen herzustellen.

3.4.2.3 Umleitung der Daten

Um die Audiodaten von der Quelle zum neuen Ziel umzuleiten, wird wieder eine eigene Klasse implementiert: `JAudioOutBuffer`. Sie ist etwas weniger umfangreich als die zuvor erstellte Klasse `JAudioInBuffer`. Ein Puffer wird dieses Mal nicht benötigt, es muss nur der Zugriff auf einen `AudioInputStream` bereit gestellt werden.

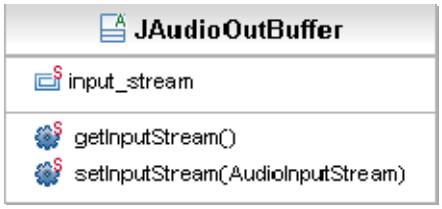


Abbildung 48: Methoden und Eigenschaften der Klasse JAudioOutBuffer

```
public abstract class JAudioOutBuffer {

	static protected AudioInputStream input_stream;

	public static AudioInputStream getInputStream (){
		return input_stream;
	}
	public static void setInputStream (AudioInputStream stream){
		input_stream = stream;
	}
}
```

Die Klasse hat den Modifikator `abstract`, was dazu führt, dass keine Instanz der Klasse gebildet werden kann. Da nur eine statische Variable gelesen und geschrieben wird, ist eine Instanziierung der Klasse nicht nötig.

In der Methode `play()` der Klasse `VxAudio` wird nun ein weiterer `AudioInputStream` zu dem bereits vorhandenen für die VoIP-Übertragung erstellt:

```
AudioInputStream stream = AudioSystem.getAudioInputStream(new
URL(uri));
AudioInputStream uStream = AudioSystem.getAudioInputStream(new
URL(uri));
```

Das Format des neuen Audioeingabestroms muss für VoIP konvertiert werden:

```
format = new AudioFormat( 8000, 16, 1, true, true);
uStream = AudioSystem.getAudioInputStream(format, uStream);
uStream = AudioSystem.getAudioInputStream(
AudioFormat.Encoding.ULAW, uStream);
```

Das Ausgabeformat des phoplayers hat eine Abtastrate von 22 kHz. Mit Hilfe der Tritonus-Bibliothek können die Audiodaten auf eine Abtastrate von 8 kHz resampled werden. Durch den Aufruf von `AudioSystem.getAudioInputStream()` wird die Konvertierung veranlasst. Es ist jedoch nicht möglich, die Abtastrate und das Kodierungsverfahren in einem einzigen Aufruf zu ändern. Deshalb wird hier die Methode noch ein zweites Mal Aufgerufen, um die Kodierung von PCM in ULAW zu ändern.

Anschließend wird der neue `AudioInputStream` an die Klasse `JAudioOutBuffer` übergeben:

```
JAudioOutBuffer.setInputStream(uStream);
```

In der Methode `RTPStreamSender.run()` kann nun auf diesen `AudioInputStream` zugegriffen werden:

```
//num = input_stream.read(buffer, 12, buffer.length-12);
AudioInputStream in2 = JAudioOutBuffer.getInputStream();
num = in2.read(buffer, 12, buffer.length-12);
```

Anstatt vom Audioeingang werden die zu sendenden Audiodaten nun aus der von der Sprachausgabe gespeicherten Datei gelesen. Es sind jedoch noch zwei Ausnahmen zu beachten:

1. Bisher hat noch keine Sprachausgabe stattgefunden. Demzufolge liefert der Aufruf von `JAudioOutBuffer.getInputStream()` den Wert `null` zurück.
2. Die letzte Datei wurde komplett übertragen. Es können keine weiteren Bytes mehr gelesen werden.

Mit Hilfe eines try-ctach-Blocks können die Ausnahmen abgefangen werden:

```
int data = 0;

AudioInputStream in2 = JAudioOutBuffer.getInputStream();

try{
      data = in2.available();
}
catch (Exception e){ data = 0; }
```

Die Variable `data` hält fest, wie viele Bytes noch gelesen werden können. Der genaue Wert wird durch den Aufruf von `audioInputStream.available()` ermittelt. Liefert der Aufruf von `JAudioOutBuffer.getInputStream()` den Wert `null` zurück, so wird später im try-Block eine `NullPointerException` ausgelöst. Diese wird im catch-Block abgefangen und die Anzahl der lesbaren Bytes wird auf Null gesetzt.

Wenn der Wert `data` größer als Null ist, dann können Daten wie oben beschrieben aus dem `AudioInputStream` `in2` gelesen werden. Jetzt bleibt aber noch zu klären, was für den Fall, dass keine Daten gelesen werden können, unternommen werden soll.

Eine Möglichkeit wäre, das Lesen der Daten zu überspringen und kein Paket mit Sprachdaten zu versenden. Wenn man aber den Quellcode etwas weiter verfolgt, stellt man fest, dass sich in diesem Fall der `RTPStreamSender` von selbst beenden würde.

Eine zweite Lösung, die in Frage käme, wäre eine Warteschleife, wie sie schon zuvor in der Methode `jaudio.readSamples()` verwendet wurde. D. h. es wird solange gewartet, bis Daten gelesen werden können und erst dann wird mit dem Senden fortgefahren. Dieses Vorgehen ist jedoch nicht sinnvoll, da sich das Ausbleiben der Pakete für den Empfänger wie ein Abbruch der Verbindung darstellt.

Die dritte und als einzige praktikable Lösung ist das Senden von alternativen Daten. Dabei sollte es sich sinngemäß um Audiodaten handeln, die Stille repräsentieren.

Nach dem obigen try-catch-Block wird folgende Verzweigung eingefügt:

```
if (data == 0){
      num = buffer.length-12;
      for (int i=12; i < buffer.length-12; ++i){
            buffer[i] = (byte)255;
      }
}
else {
      num=in2.read(buffer,12,buffer.length-12);
}
```

Für den Fall, das keine Daten gelesen werden können, wird der komplette Buffer mit Bytes vom Wert 255 gefüllt. Dies entspricht bei der ULAW-Kodierung einer Amplitude von Null. Können jedoch Daten gelesen werden, dann werden ganz normal die gelesenen Daten in den Buffer geschrieben.

Mit diesem letzten Schritt ist die Voice-over-IP-Anbindung des Sprachservers komplett und Audiodaten können über eine VoIP-Verbindung sowohl empfangen als auch gesendet werden.

3.5 Anpassung

3.5.1 Samplefrequenz des Erkenners

3.5.1.1 Einleitung

Da der Sprachserver ursprünglich auf die Erkennung von Eingangsdaten mit einer Abtastrate von 16 kHz ausgelegt war, bleibt die Erkennungsgenauigkeit trotz des durchgeführten Resamplings unter dem zu erwartenden Wert. Verantwortlich dafür sind Frequenzbereiche im Signal, die der Erkenner erwartet, die aber nicht in dem neuabgetasteten Signal vorhanden sind. Um die Erkennungsgenauigkeit zu verbessern muss der Erkenner auf die neue Abtastrate von 8 kHz angepasst werden. Dazu sind mehrere Schritte notwendig.

3.5.1.2 Anpassen der Audioaufzeichnung

Die Digitalisierung des analogen Sprachsignals wurde bisher mit 16 kHz durchgeführt. Um die Abtastrate zu ändern wurde eine neue Einstellung in die Konfigurationsdatei „fbsv.ini“ mit der Bezeichnung „sampleRate“ aufgenommen. Der Quellcode wurde an den entsprechenden Stellen so geändert, dass anstatt einem festen Wert von 16 kHz die Einstellung aus der Konfigurationsdatei verwendet wird. Mit der Zeile „sampleRate=8000“ in der Konfigurationsdatei zeichnet der Sprachserver nun die Audiodaten mit einer Rate von 8 kHz auf.

3.5.1.3 Anpassen der Hidden Markov Modelle

Die vorhandenen Hidden Markov Modelle (HMMs) müssen durch neue ersetzt werden, die mit 8 kHz Audiodaten antrainiert wurden. Dazu werden die originalen Audiodaten neu abgetastet und das Training wird anschließend nochmal wiederholt. In der Konfigurationsdatei muss nun nur noch über den Wert „htkHMMMacro“ die Datei mit den 8 kHz-Modellen angegeben werden.

3.5.1.4 Anpassen der HVite-Konfiguration

Das Programm HVite, das für die Erkennung benutzt wird, muss nun über die geänderten Verhältnisse informiert werden. Dazu wird die Konfigurationsdatei von HVite entsprechend abgeändert:

vorher		nachher	
SOURCEKIND	= WAVEFORM	SOURCEKIND	= WAVEFORM
SOURCEFORMAT	= WAVE	SOURCEFORMAT	= WAVE
SOURCERATE	= 625.0		
ZMEANSOURCE	= FALSE	ZMEANSOURCE	= FALSE
ENORMALISE	= FALSE	ENORMALISE	= FALSE
TARGETFORMAT	= HTK	TARGETFORMAT	= HTK
TARGETKIND	= MFCC_E_D	TARGETKIND	= MFCC_E_D
TARGETRATE	= 100000	TARGETRATE	= 100000
NUMCHANS	= 21	NUMCHANS	= 14
NUMCEPS	= 12	NUMCEPS	= 12
SAVEWITHCRC	= FALSE	SAVEWITHCRC	= FALSE
WINDOWSIZE	= 200000.0	WINDOWSIZE	= 250000.0
USEHAMMING	= TRUE	USEHAMMING	= TRUE
PREEMCOEF	= 0.97	PREEMCOEF	= 0.97
LOFREQ	= 330.0	LOFREQ	= 300.0
HIFREQ	= 5500.0	HIFREQ	= 3400.0
CEPLIFTER	= 22	CEPLIFTER	= 22
DELTAWINDOW	= 2	DELTAWINDOW	= 2
USEPOWER	= TRUE	USEPOWER	= TRUE

Tabelle 13: Änderungen in der HVite-Konfigurationsdatei

Die Unterschiede sind in Tabelle 13 rot markiert. Besonders wichtig ist hier der Wert „HIFREQ“, der die maximale Frequenz spezifiziert. Damit das Shannon-Theorem $f_{max} = f_{abt}/2$ nicht verletzt wird, muss die maximale Frequenz bei einer Abtastrate von 8 kHz unter 4000 Hz liegen.

Damit wären alle Anpassungen für die neue Abtastrate von 8 kHz durchgeführt.

3.5.2 Phonembasierte Erkennung

Der Sprachserver war bisher auf die Verwendung von Hidden-Markov-Ganzwortmodellen konfiguriert. Für die Erstellung des Beispieldialogs ist dies aber wenig praktikabel, die Phonem-basierte Erkennung ist hier besser geeignet. Die folgende Tabelle vergleicht die beiden Methoden:

Wort-basierte Erkennung	Phonem-basierte Erkennung
Anzahl der Modelle ist abhängig von der Anzahl der zu erkennenden Worte.	Anzahl der Modelle ist fest. Für die Erkennung der deutschen Sprache werden ca. 40 Phonem-Modelle benötigt.
Jedem Wort ist ein Modell zugeordnet.	Jedem Wort ist ein Eintrag im Aussprachelexikon zugeordnet, welcher die Phonem-Folge enthält.
Erkennungsaufwand steigt deutlich mit Anzahl der Worte.	Erkennungsaufwand steigt pro Wort nur in geringem Maße.
Nur kleine Wortschätze möglich.	Auch große Wortschätze möglich
Hoher Trainingsaufwand. Jedes Wort muss trainiert werden.	Es müssen nur die Phonem-Modelle trainiert werden. Kein Training für neue Worte nötig.
Wenig flexibel. Wörter müssen vorher bekannt sein.	Dynamische Generierung während der Laufzeit möglich.
Gute Erkennungsrate.	Erkennungsrate ist gegenüber Ganzwortmodellen etwas niedriger.

Tabelle 14: Vergleich von Wort-basierter mit Phonem-basierter Erkennung

Der Hauptgrund, der gegen die Verwendung von Ganzwortmodellen im Beispieldialog spricht, ist der hohe Trainingsaufwand. Jedes mal, wenn ein neues Wort benötigt wird, müssten erst Aufnahmen mit einer großen Anzahl von Probanden durchgeführt werden um anschließend das Modell generieren zu können.

Bei der Phonem-basierten Erkennung muss für ein neues Wort nur ein Eintrag im Aussprachelexikon hinzugefügt werden. Diese können in den meisten Fällen aus bereits existierenden Aussprache-Datenbanken wie Hadi-Bomp entnommen werden.

Die für die Phonem-basierte Erkennung benötigten Änderungen werden in der Konfigurationsdatei „fbsv.ini“ durchgeführt:

Mit dem Wert „htkHMMMacro“ wird die Datei spezifiziert, die die Hidden-Markov-Modelle (HMMs) enthält. Hier muss nun die Datei mit den Phonem-Modellen angegeben werden:

```
htkHMMMacro=arbeit\\htkfiles\\hmms\\hmm.p\\phoneme_3_8k
```

Der Wert „htkHMMList“ gibt eine Datei an, die eine Liste aller HMMs enthält. In diesem Fall ist das eine Liste von 44 Phonem-Modellen:

```
htkHMMList=arbeit\\htkfiles\\dicts\\phoneme
```

Schließlich muss noch das Aussprache-Lexikon angegeben werden, welches die zu erkennenden Worte samt Phonem-Folge enthält:

```
htkDictionary=arbeit\\htkfiles\\dicts\\vokabular_info
```

Die Einträge im Lexikon für die Wochentage Montag - Sonntag würden beispielsweise folgendermaßen lauten:

```
Montag              pau m oo n t aa k pau
Dienstag            pau d ii n s t aa k pau
Mittwoch            pau m gi t v go x pau
Donnerstag          pau d go n schwa r s t aa k pau
Freitag             pau f r ai t aa k pau
Samstag             pau z a m s t aa k pau
Sonntag             pau z go n t aa k pau
```

Nach diesen Änderungen ist der Sprachserver bereit für die Phonem-basierte Erkennung.

3.5.3 DTMF-Erkenner

3.5.3.1 Einleitung

Die Abkürzung DTMF steht für „Dual Tone Multiple Frequency“ und ist äquivalent zu dem deutschen Begriff Mehrfrequenzwahlverfahren (MFV) oder Tonwahlverfahren. Dabei handelt es sich um ein gebräuchliches Verfahren um Rufnummern im analogen Telefonnetz zu übermitteln [Wik08].

DTMF eignet sich aber auch besonders gut als Alternative zur Spracheingabe. Der Vorteil dabei ist, dass weniger Zeit bei der Eingabe über die Telefontastatur benötigt wird und dass die Erkennung zuverlässiger funktioniert. Besonders bei Nummern-Eingaben, wie z. B. PIN oder Kreditkarten-Nummern, ist die Verwendung von DTMF zu empfehlen.

3.5.3.2 Funktionsweise

Die Information über die gedrückte Taste bzw. gewählte Ziffer wird bei DTMF durch eine Überlagerung von zwei Tönen übertragen. Die zwei Einzeltöne ergeben sich aus der Position der gedrückten Taste. Die Zeile, in der sich die Taste befindet, wird durch einen tiefen Ton und die Spalte durch einen hohen Ton kodiert.

DTMF ist ein sogenanntes In-Band-Signalisierungsverfahren, d. h. die übertragenen Töne liegen im gleichen Frequenzbereich wie das Sprachsignal und sind für den Menschen auch wahrnehmbar. Um Fehlerkennungen zu vermeiden, wurden Frequenzen gewählt, deren Kombination üblicherweise nicht im Sprachsignal auftreten. Die Werte der einzelnen Frequenzen sind in Abbildung 49 zu sehen.

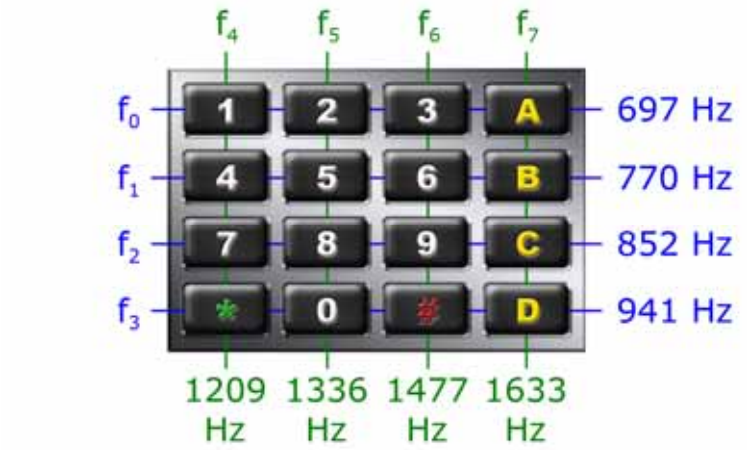


Abbildung 49: Frequenzen bei DTMF

Drückt der Benutzer z. B. die Taste 1, so wird ein Ton aus der Überlagerung der Frequenzen 697 Hz und 1209 Hz erzeugt. Das entsprechende Zeitsignal mit zugehörigem Spektrum ist in Abbildung 50 und 51 zu sehen.

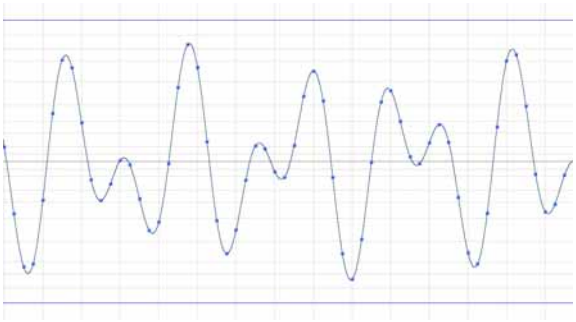


Abbildung 50: Zeitsignal des Tons für die Taste 1

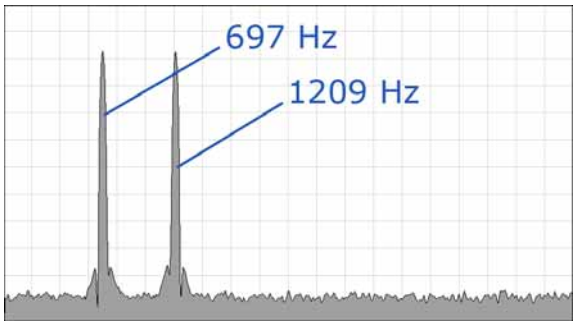


Abbildung 51: Frequenzspektrum des Tons für die Taste 1

Zur Erkennung der gedrückten Taste wird auf der Gegenseite eine Spektralanalyse durchgeführt. Dies geschieht meistens mit Hilfe des Goertzel-Algorithmus, einem auf der Diskreten Fourier-Transformation beruhendem Algorithmus, mit dem es möglich ist, das Signal auf das Vorhandensein einzelner Frequenzanteile zu untersuchen. Wenn die Töne für Zeile und Spalte gefunden wurde, kann die Taste bestimmt werden [Wik08].

3.5.3.3 Implementierung

Für die Erkennung der DTMF-Töne kommt das externe Open-Source-Programm DTMF2NUM[21] von Luigi Auriemma zum Einsatz. Mit diesem Kommandozeilenprogramm kann eine Folge von DTMF-Tönen aus einer Wave-Datei ausgelesen werden. Der Aufruf erfolgt mit einer Wave-Datei als Parameter. Die Ausgabe des Programms ist in Abbildung 52 zu sehen.

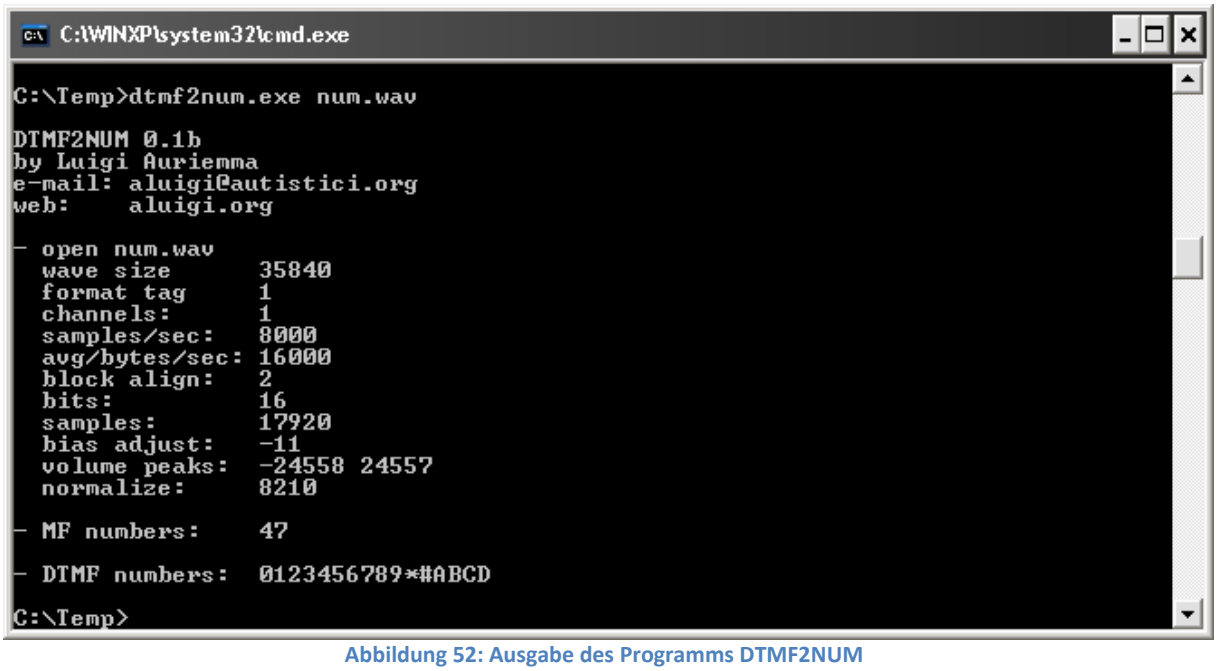


Abbildung 52: Ausgabe des Programms DTMF2NUM

[21] Erhältlich auf der Seite http://aluigi.altervista.org/mytoolz.htm

Ein Blick in den Quellcode zeigt, dass das Programm auf die Signalverarbeitungsroutinen des Asterisk Projektes[22] zurückgreift. Dabei handelt es sich um ein bekanntes Open-Source-Toolkit zum erstellen von Telefonanwendungen.

Damit das Ergebnis an den Demonstrator übermittelt werden kann, wurde das Programm um die Fähigkeit erweitert, die erkannte Nummernfolge in einer Datei zu speichern.

Bevor der Demonstrator DTMF2NUM verwenden kann, muss er mitgeteilt bekommen, wann denn das Programm aufgerufen werden soll. Dazu wird ein neues Element <dtmf> eingeführt, welches die DTMF-Erkennung für das umgebende Element aktivieren soll. Wird das <dtmf>-Element beispielsweise innerhalb eines <field>-Elements platziert, so sollen bei dieser Eingabe auch DTMF-Töne akzeptiert werden.

In der Klasse `VxItem`, die für alle Elemente zuständig ist, wird eine neue Variable erstellt, welche festhält, ob eine DTMF-Erkennung für das aktuelle Element gewünscht ist:

```
boolean dtmf = false;
```

Damit der VoiceXML-Parser das neue <dtmf>-Element auch verarbeiten kann, wird die Funktion `startElement()` der Klasse `S1` um eine entsprechende Abfrage erweitert:

```
[…]
else if ( matchesType( "dtmf", localname, qName ) ) {
      active.dtmf = true;
} […]
```

Die Funktion `recognize()` der Klasse `Recognizer`, die das Aufzeichnen und Auswerten einer Äußerung veranlasst, wird nun um den Parameter `dtmf` erweitert. Der Funktionsaufruf in der Klasse `VxField` sieht nun folgendermaßen aus:

```
erkannt = recognizer.recognize(dtmf);
```

Der Aufruf von DTMF2NUM kann jetzt in der Funktion `recognize()` direkt nach der Aufzeichnung einer Äußerung durchgeführt werden:

```
if (dtmf) {
      String dtmfFile = saveWave.replaceAll( "wav$", "dtmf");
      String cmdline = "dtmf2num -f " + dtmfFile + " " + saveWave;
      Process p = Runtime.getRuntime().exec(cmdline);
      p.waitFor();

      FileReader reader = new FileReader(dtmfFile);
      int result = reader.read();
      reader.close();

      if (result != -1) return "dtmf_" + (char)result;
}
```

[22] URL: http://www.asterisk.org/

Ist eine DTMF-Erkennung erwünscht (`dtmf = true`), so wird das Programm DTMF2NUM mit der soeben abgespeicherten Äußerung als Parameter aufgerufen. Danach wird versucht, aus der Datei mit dem Erkennungsergebnis zu lesen. Wurden DTMF-Töne erkannt, so enthält die Datei nun ein oder mehrere Zeichen. In diesem Fall wird die Erkennung beendet und eine Zeichenkette, beginnend mit „dtmf_“ und gefolgt von dem ersten erkannten Zeichen, wird zurückgegeben. Für den Fall, dass kein DTMF-Ton erkannt wurde und die Datei leer ist, wird mit der Spracherkennung mittels HVite fortgefahren.

Drückt der Benutzer während der Spracheingabe z.B. die Taste 5 am Telefon, so wird „dtmf_5“ als Erkennungsergebnis zurückgeliefert. Die „dtmf“-Zeichenketten müssen aber noch mit den vorhandenen Optionen verknüpft werden. Dazu wird eine JavaScript-Funktion verwendet:

```
function dtmf2word (string, array)
{
      var defaultString = "default";

      if (!string.substring(0,5).equals("dtmf_"))
            return string;

      for (var i=0; i < array.length; ++i)
      {
            if (array[i][0].equals("default"))
                  defaultString = array[i][1];
            if (array[i][0].equals(string))
                  return array[i][1];
      }
      return defaultString;
}
```

Der Parameter `string` ist eine Zeichenkette mit dem Erkennungsergebnis und `array` ist ein Feld, welches die Verknüpfungen zwischen DTMF-Tönen und den vorhandenen Option spezifiziert. Das Feld für die Zuordnung der Tasten 1-7 zu den Wochentagen könnte so aussehen:

```
var dtmf_tage = [
      [ "default", "keine Option" ],
      [ "dtmf_1", "Montag" ],
      [ "dtmf_2", "Dienstag"],
      [ "dtmf_3", "Mittwoch" ],
      [ "dtmf_4", "Donnerstag" ],
      [ "dtmf_5", "Freitag" ],
      [ "dtmf_6", "Samstag" ],
      [ "dtmf_7", "Sonntag" ]
];
```

Hat der Benutzer wie im obigen Beispiel die Taste 5 gedrückt, so wird das Erkennungsergebnis durch den Funktionsaufruf von „dtmf_5“ in „Freitag“ geändert. Der Eintrag „default“ steht hier für alle nicht festgelegten Tasten. Wird z. B. die Taste 8 gedrückt, so wird „keine Option“ zurückgeliefert.

Es ist wichtig, dass alle nicht zugeordneten Tasten abgefangen werden. Nach dem <field>-Element zur Erkennung der Wochentage sollte ein entsprechender Block folgen:

```
<block>
      <if cond="(tag = dtmf2word(tag, dtmf_tage)) ==
      'keine Option'">
            <prompt>
                  Es sind nur die Tasten von eins bis sieben
                  möglich.
            </prompt>
            <goto nextitem="tag"/>
      </if>
</block>
```

Falls der Benutzer hier eine andere Taste als 1-7 drückt, so wird er darauf hingewiesen, dass nur die Tasten 1-7 möglich sind und anschließend wird die Abfrage wiederholt. Das folgende Diagramm zeigt eine schematische Darstellung der Abfangschleife:

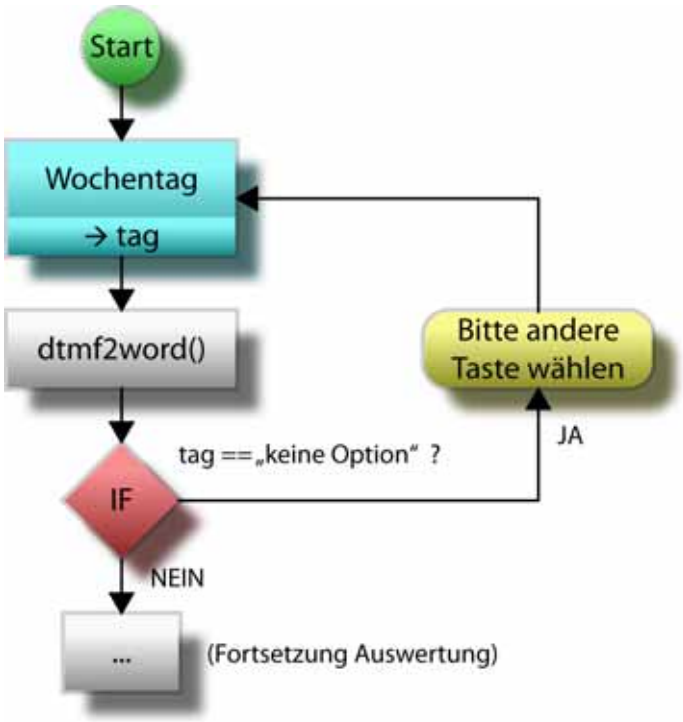


Abbildung 53: Abfangschleife für nicht zugeordnete Tasten

Anmerkung: Aus Gründen der Übersicht werden die Abfangschleifen in den später folgenden Diagrammen des Beispieldialogs nicht weiter dargestellt. Prinzipiell wird nach jedem Feld mit DTMF-Unterstützung eine solche Schleife benötigt.

Mit der Zuordnung der einzelnen Tasten zu den Optionen und dem Abfangen von nicht zugeordneten Tasten ist die Implementierung des DTMF-Erkenners abgeschlossen.

3.6 Erstellen eines Beispieldialogs

3.6.1 Einleitung

Nachdem der Demonstrator fertig gestellt ist soll nun eine konkrete Anwendung entwickelt werden, mit der der Demonstrator getestet und ausprobiert werden kann. Für diesen Zweck soll ein Dialog in VoiceXML erstellt werden.

Als praktischer Anwendungsfall wurde ein „FH Informationsserver“ gewählt. Dieser soll Studierenden an der Fachhochschule Friedberg ermöglichen, verschiedene Informationen bezüglich des Studiums über das Telefon abzufragen.

Folgende Dienste würden für den Informationsserver in Frage kommen:

Notenauskunft
Studierende könnten sich hier über die Ergebnisse einer bestimmten Klausur erkundigen, die Noten für ein spezifisches Semester abfragen oder sich alle bisher erbrachten Leistungen auflisten lassen.

Klausuranmeldung
Die Klausuranmeldung würde den Studierenden die Möglichkeit bieten, sich die Klausuren auflisten zu lassen, für die sie aktuell angemeldet sind. Weiterhin hätten sie die Option, sich für weitere Klausuren anzumelden oder bestimmte Anmeldungen zu revidieren.

Bibliotheksauskunft
Hier könnten Informationen wie Öffnungszeiten und Verfügbarkeiten von diversen Büchern abgefragt werden. Es könnte auch eine Liste ausgeliehener Bücher abgefragt werden, mit der Möglichkeit, die Ausleihfrist für gewisse Titel verlängern zu lassen. Schließlich wäre auch noch eine Reservation für bestimmte Bücher denkbar.

Mensaauskunft
In der Mensaauskunft könnten sich Studierende über die Öffnungszeiten sowie die Speisekarte für die verschiedenen Wochentage inkl. der Preise erkundigen.

Schwarzes Brett
Das digitale Schwarze Brett wäre eine Alternative zu den Aushängen in der Fachhochschule. Aktuelle Informationen könnten hier von Verwaltung, Dozenten oder Administratoren abgelegt werden und würden den Studenten zur Verfügung stehen. Die Informationen könnten bestimmten Kursen zugeordnet werden, so dass der einzelne Student nur die Neuigkeiten erhält, die für ihn relevant sind.

Generelle Auskunft
Es wäre auch eine generelle Auskunft denkbar, die sich auch an Nicht-Studierende richtet. Hier könnten Informationen wie die Öffnungszeiten des FH Gebäudes, die Vorlesungszeiten, Einführungsveranstaltungen etc. hinterlegt werden.

3.6.2 Das Konzept

Für den Beispieldialog wurden die Dienste Mensaauskunft, Notenauskunft und Klausuranmeldung ausgewählt. In Abbildung 54 ist der schematische Aufbau des Dialogs dargestellt.

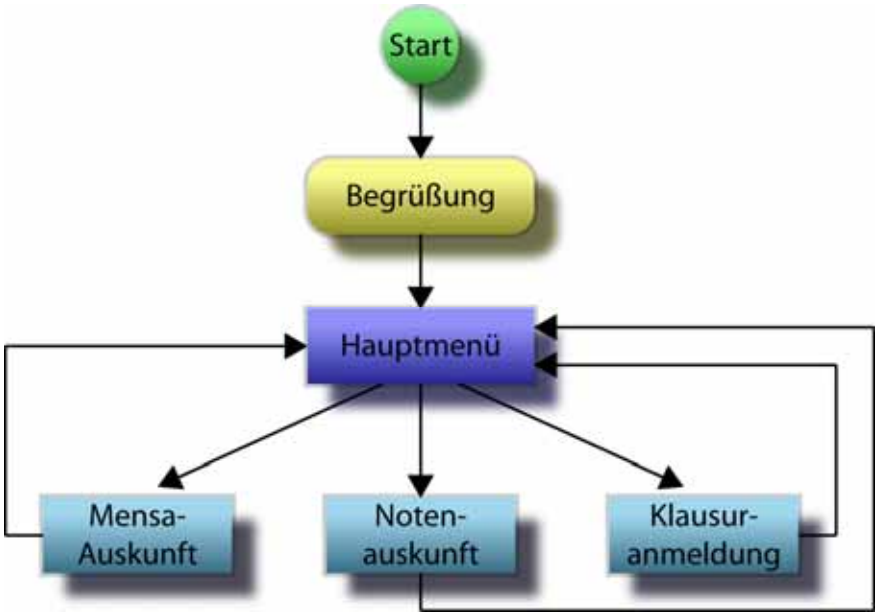


Abbildung 54: Aufbau des Beispieldialogs

Nach dem Verbindungsaufbau wird der Benutzer zunächst begrüßt und gelangt dann in das Hauptmenü. Hier kann er aus den einzelnen Diensten auswählen. Hat er sich für einen Dienst entschieden, dann wird er in den entsprechenden Unterdialog weitergeleitet.

Die Unterdialoge bieten jeweils die Möglichkeit, wieder ins Hauptmenü zurückzukehren, damit der Benutzer auch einen der anderen Dienste verwenden kann.

3.6.3 Implementierung

3.6.3.1 Begrüßung und Hauptmenü

Bei der Begrüßung handelt es sich um eine einfache Sprachausgabe. Damit der Benutzer diese nur einmal hört, ist die Begrüßung separat vor dem Hauptmenü platziert. Sie ist auch gleichzeitig der Einstiegspunkt für den Dialog.

Aus der Datei „info_welcome.vxml“:

```
<form id="welcome">
      <block>
            <prompt>
                  Willkommen im Informationssystem der
                  Fachhochschule Giessen – Friedberg
            </prompt>
      </block>
      <goto next="info_main.vxml"/>
</form>
```

Die Sprachausgabe erfolgt mittels der Anweisung <prompt>. Anschließend wird durch die <goto>-Anweisung die Datei für das Hauptmenü aufgerufen.

Aus der Datei „info_main.vxml“:

```
<field name="selection">
      <prompt>Hauptmenue. Bitte waehlen Sie einen Bereich: Mensa,
      Notenauskunft, Klausuranmeldung.</prompt>
      <option>Mensa</option>
      <option>Notenauskunft</option>
      <option>Klausuranmeldung</option>
</field>
```

Durch das <field>-Element wird der Demonstrator angewiesen, eine Spracheingabe vom Benutzer anzufordern. Die möglichen Antworten sind durch die <option>-Felder vorgegeben. Nach der Erkennung wird die Variable „selection“ den Wert von einer der Optionen enthalten.

```
<if cond="selection == 'Mensa'">
      <goto next="info_mensa_welcome.vxml"/>
</if>
<if cond="selection == 'Notenauskunft'">
      <goto next="info_noten_welcome.vxml"/>
</if>
<if cond="selection == 'Klausuranmeldung'">
      <goto next="info_klausur_welcome.vxml"/>
</if>
```

Hier wird mit Hilfe des Bedingungsoperators <if> entschieden, welche Datei als nächstes geladen werden soll. Der Inhalt der Variable „selection“ wird nacheinander mit den vorgegebenen Antworten verglichen. Liefert der Vergleich den Wert true (wahr) zurück, so wird die entsprechende Datei mit der <goto>-Anweisung geladen.

3.6.3.2 Mensa-Auskunft

Jeder Unterdialog, so auch die Mensa-Auskunft, hat eine eigene Begrüßung, um den Benutzer darüber zu informieren, welchen Bereich er betreten hat.

Nach der Begrüßung gelangt der Benutzer in das Hauptmenü des jeweiligen Bereiches. Neben dem Aufruf der Hauptfunktionen des Bereichs kann der Benutzer hier auch wieder in das oberste Hauptmenü zurückkehren. In der Mensa-Auskunft stehen die Funktionen „Öffnungszeiten“ und „Speisekarte“ zur Verfügung.

Wählt der Benutzer „Öffnungszeiten“ so erhält er eine Sprachausgabe, die ihn über die Öffnungszeiten der Mensa aufklärt. Anschließend gelangt er automatisch in das Hauptmenü des Bereichs zurück.

Bei der Funktion „Speisekarte“ wird der Benutzer nach einem Wochentag gefragt, für den er die Speisekarte hören möchte. Nach der Sprachausgabe hat er die Möglichkeit, sich für noch einen anderen Tag die Speisekarte anzuhören, indem er die Frage nach einem weiteren Tag mit „Ja“ beantwortet. In diesem Fall kehrt er noch einmal in das Wochentags-Menü zurück, ansonsten gelangt er in das Hauptmenü des Bereichs.

Abbildung 55 veranschaulicht den Aufbau des Dialoges.

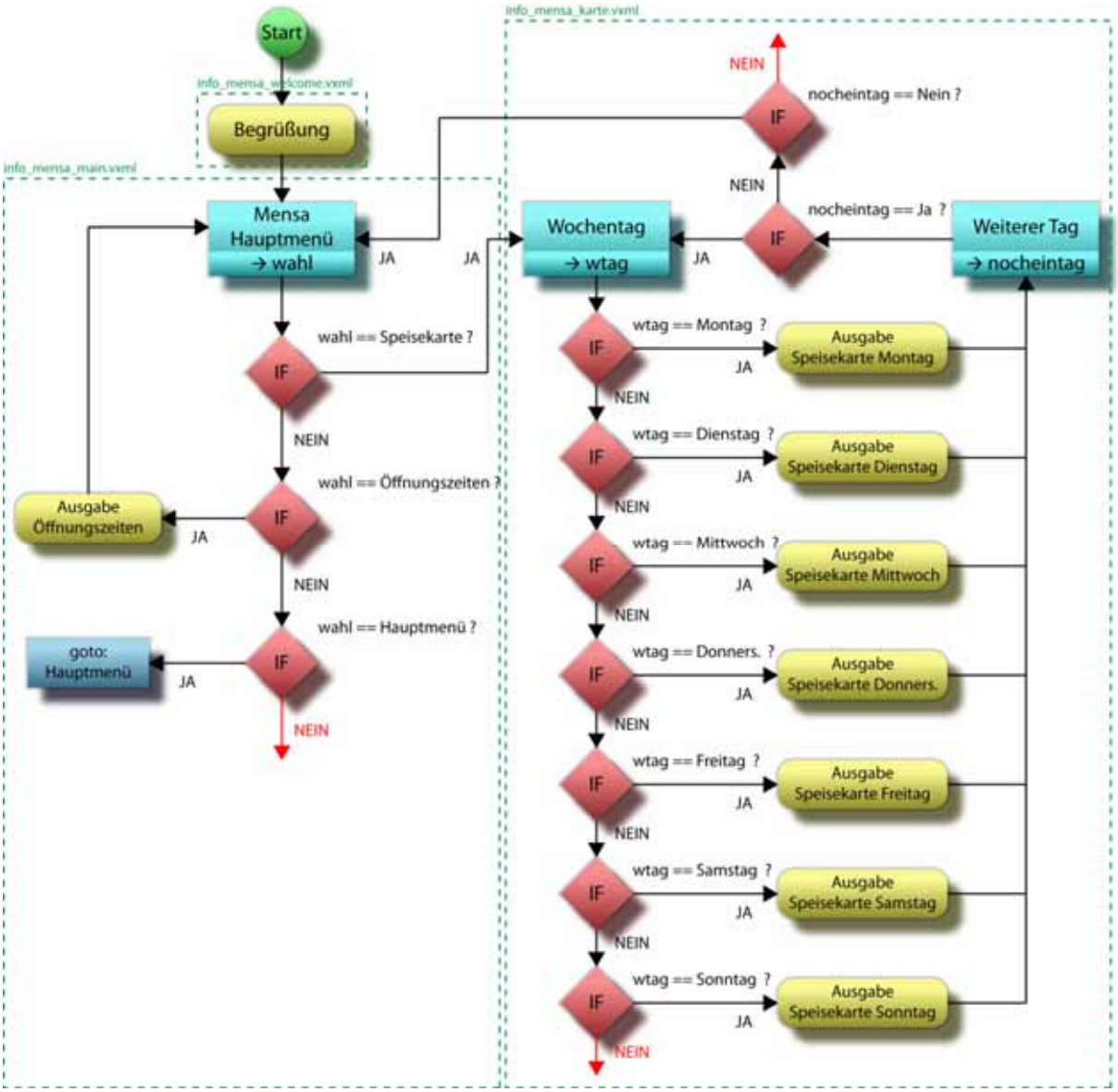


Abbildung 55: Schematischer Aufbau des Dialogs „Mensaauskunft“

Der Unterdialog „Mensaauskunft“ besteht aus insgesamt drei VoiceXML-Dateien: Begrüßung, Hauptmenü und Speisekarte.

Ein Dialog lässt sich mittels des <form>-Elements in mehrere Formulare unterteilen. Diese können mit der Anweisung <goto> direkt angesprungen werden. Es ist auch möglich mehrere Formulare in einer einzelnen Datei zu haben. Dies wird aber vom VoiceXML-Parser noch nicht vollständig unterstützt und so ist nur der Sprung in das erste Formular einer VoiceXML-Datei möglich. Damit die Navigation dennoch funktioniert, muss jedes Formular in eine eigene Datei gespeichert werden. Dieses Vorgehen ist für de Übersichtlichkeit der Dialoge jedoch sinnvoll.

3.6.3.3 Klausur-Anmeldung

Im Hauptmenü der Klausur-Anmeldung hat der Benutzer die Optionen, sich für eine Klausur anzumelden, sich von einer Klausur abzumelden, eine Liste aller angemeldeten Klausuren zu erhalten oder in das vorherige Menü zurückzukehren.

Die Funktion der einzelnen Operationen wurde in JavaScript implementiert. Die meisten JavaScript-Variablen und Funktionen wurden in der Begrüßungs-Datei „info_klausur_welcome.vxml“ abgelegt. Damit stehen sie allen später folgenden Dialogbereichen zur Verfügung.

Die Klausuren und der aktuelle Status (angemeldet/nicht angemeldet) werden in einem Array festgehalten:

```
var klausuren = [
[
     ["Informationsverarbeitung_Eins", true],
     ["Betriebs_und_Wirtschaftslehre", false],
     ["Gestaltungsgrundlagen", true],
     ["Mathematik_Eins", false],
     ["Privat_und_Arbeitsrecht", false]
],
[
     ["Informationsverarbeitung_Zwei", false],
     ["Akustik_Optik_Wellen", true],
     ["Mathematik_Zwei", true],
     ["Wirtschaftsenglisch", false]
],
[
     ["Informationsverarbeitung_Drei", false],
     ["Mediengestaltung_Eins", false],
     ["Digitaltechnik_Eins", false],
     ["Mathematik_Drei", true],
     ["Nachrichtentechnik_Eins", false]
]
];
```

Dabei handelt es sich um ein dreidimensionales Array. Die erste Dimension legt das Semester fest, die zweite Dimension besteht aus den einzelnen Klausuren und in der dritten Dimension sind die Namen und der Zustand der Klausuren gespeichert. Für den Beispieldialog wurden nur die ersten drei Semester angelegt, das Array lässt sich aber beliebig fortsetzen.

Mit der Funktion `listAngemeldet()` kann man alle Klausuren ausgeben, die den Zustand `true` für „angemeldet“ tragen:

```
function listAngemeldet ()
{
      var result = ""

      for (var i=0; i < klausuren.length; ++i)
      {
            for (var j=0; j < klausuren[i].length; ++j)
            {
                  if (klausuren[i][j][1] == true)
                        result += klausuren[i][j][0] + ".\n";
            }
      }
      return result;
}
```

Wählt der Benutzer im Menü der Klausuranmeldung den Punkt „Liste“ so wird diese Funktion aufgerufen und der Benutzer erhält das Ergebnis als Sprachausgabe. Der Aufruf erfolgt innerhalb der <prompt>-Anweisung, umgeben vom <value>-Tag:

```
<prompt>
      Sie haben sich angemeldet fuer:
      <value expr="listAngemeldet()"/>
</prompt>
```

Möchte der Benutzer eine Klausur anmelden, so wird er zunächst nach dem zugehörigen Semester gefragt. Er hat hier außerdem die Möglichkeit, mit der Antwort „abbrechen“ die Anmeldung zu beenden und ins Hauptmenü der Klausur-Anmeldung zurückzukehren.

Nachdem er ein Semester genannt hat, bekommt er eine Liste aller Klausuren aus diesem Semester, für die noch keine Anmeldungen vorliegen. Dies geschieht mit der Funktion `listRemain()`:

```
function listRemain (semester, status)
{
      var result = ""
      semester -= 1;

      for (var j=0; j < klausuren[semester].length; ++j)
      {
            if (klausuren[semester][j][1] == status)
                  result += klausuren[semester][j][0] + ".\n";
      }
      return result;
}
```

Diese Funktion ist `listAngemeldet()` sehr ähnlich, jedoch gibt sie nur die Klausuren für ein spezifisches Semester aus und auch nur dann wenn der angegebene Status übereinstimmt. Der Parameter `semester` ist hier eine Zahl. Um Zahlwörter wie „Eins“ oder „erstes“ in Nummern umzuwandeln, steht die selbst erstellte Funktion `word2num()` zur Verfügung. Der Aufruf lautet also folgendermaßen:

```
<value expr="listRemain(word2num(semester), false)"/>
```

Jetzt kann der Benutzer per Spracheingabe bestimmen, welche Klausur angemeldet werden soll. Zur Sicherheit muss die Anmeldung noch mit der Antwort „Ja“ auf die Frage, ob auch die richtige Klausur erkannt wurde, bestätigt werden. Ist dies nicht der Fall, so kann der Benutzer die Angabe von Semester und Klausur nochmal wiederholen. Andernfalls wird die Anmeldung mit dem Aufruf der Funktion `klausurAnmelden()` durchgeführt:

```
function klausurAnmelden (klausur)
{
  var result = ""

  for (var i=0; i < klausuren.length; ++i)
  {
    for (var j=0; j < klausuren[i].length; ++j)
    {
      if (klausuren[i][j][0].equals(klausur))
      {
        if (klausuren[i][j][1] == true)
          return "Sie sind bereits für die Klausur " + klausur +
          " angemeldet.";
        else
        {
          klausuren[i][j][1] = true;
          return "Sie wurden erfolgreich für die Klausur " +
          klausur + " angemeldet.";
        }
      }
    }
  }
  return "Angegebene Klausur wurde nicht gefunden.";
}
```

Zunächst wird die übergebene Klausur in allen Semestern gesucht. Wurde die Klausur gefunden, so wird überprüft, ob nicht bereits eine Anmeldung vorliegt. Falls ja, so wird ein entsprechender Hinweis ausgegeben und die Anmeldung abgebrochen. Ansonsten wird die Klausur als angemeldet gekennzeichnet und eine Bestätigung ausgegeben. Nach erfolgreicher Anmeldung kehrt der Benutzer ins Menü der Klausuranmeldung zurück.

Der Dialog für die Klausurabmeldung ist nahezu identisch mit dem für die Anmeldung. Es mussten lediglich die Sprachausgaben und Funktionsaufrufe angepasst werden. Die Ausgabe der angemeldeten Klausuren anstatt der nicht angemeldeten geschieht mit Hilfe des zweiten Parameters der Funktion `listRemain()`:

```
<value expr="listRemain(word2num(semester), true)"/>
```

Schließlich wird die Abmeldung mit der Funktion `klausurAbmelden()` durchgeführt. Bis auf die Ausgaben und die Änderung des Status „angemeldet“ auf `false` anstatt `true` ist die Funktion identisch mit `klausurAnmelden()`.

Abbildung 56 zeigt eine schematische Darstellung des Dialogs Klausuranmeldung.

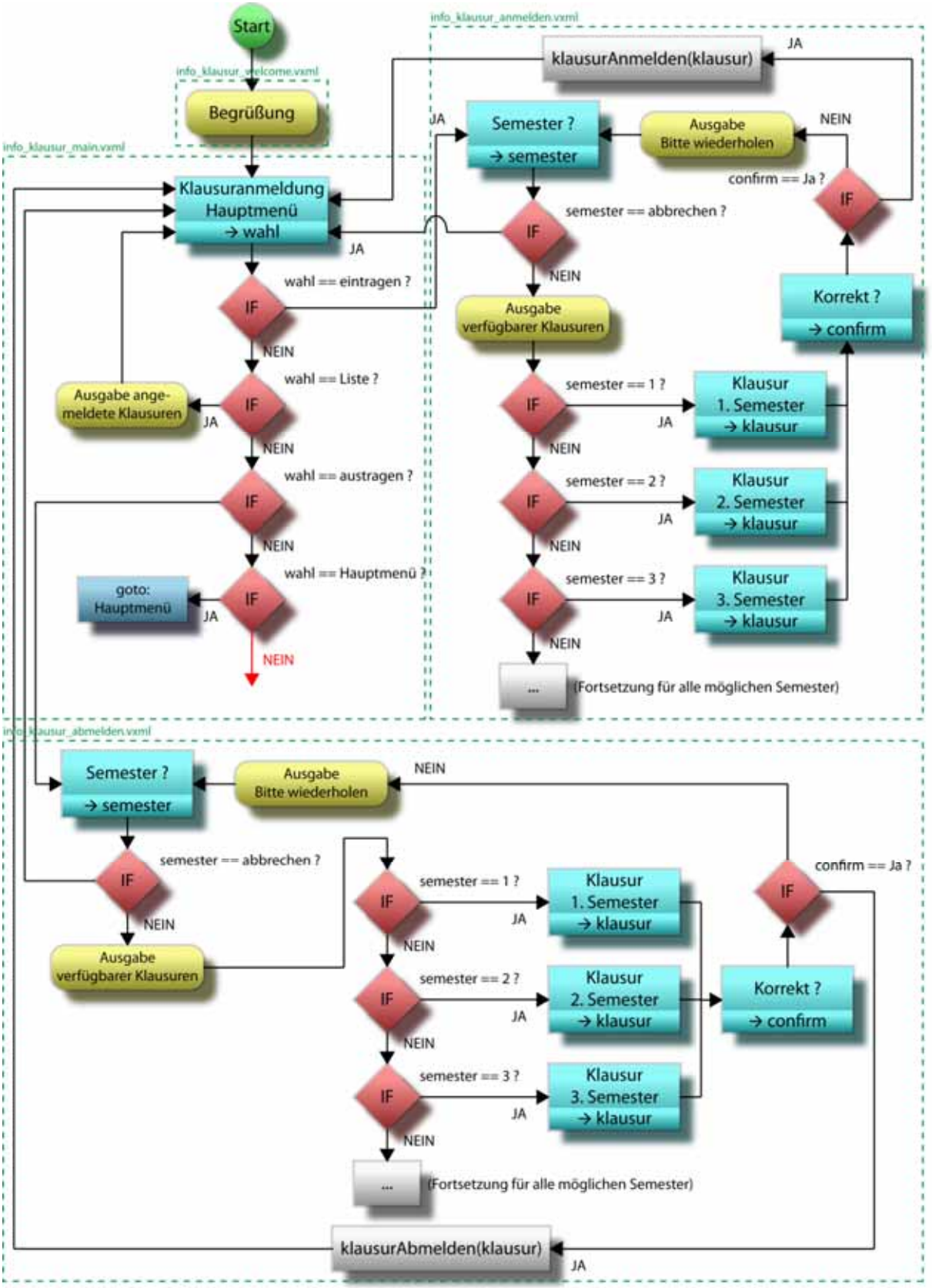


Abbildung 56: Schematischer Aufbau des Dialogs „Klausuranmeldung“

3.6.3.4 Notenauskunft

In der Notenauskunft kann sich der Benutzer über seine erbrachten Leistungen informieren. Dabei hat er die Wahl, sich alle bisherigen Leistungen aufzulisten zu lassen, die Zensuren für ein ganzes Semester zu erhalten oder nur die Note für einen bestimmtes Fach.

Die Funktionen wurden wieder mit JavaScript umgesetzt. Die Noten werden in einem Array gespeichert, das in seinem Aufbau sehr ähnlich mit dem der Klausuranmeldungen ist. An der Stelle mit dem boolschen Wert für den Status „angemeldet“ steht nun die Note.

```
var noten = [
[
      ["Informationsverarbeitung_Eins", 1.7],
      ["Betriebs_und_Wirtschaftslehre", 3.0],
      ["Gestaltungsgrundlagen", 2.3],
      ["Gestaltungsgrundlagen_Labor", -1],

[…]
```

Einige Werte haben eine besondere Bedeutung: -1 steht für ein bestandenes Testat, -2 steht für ein nicht bestandenes Testat. Der Wert 0 bedeutet, dass keine Note vorliegt, z. B. weil die Klausur noch nicht geschrieben wurde oder noch nicht bewertet worden ist.

Ausgabe aller Noten

Möchte sich der Benutzer alle Noten ausgeben, wird die Funktion `listNotenAll()`verwendet:

```
function listNotenAll ()
{
  var result = ""
  var temp = ""

  for (var i=0; i < noten.length; ++i)
  {
    temp = "";
    for (var j=0; j < noten[i].length; ++j)
    {
      if (noten[i][j][1] != 0)
        temp += noten[i][j][0] + ". " + note2word(noten[i][j][1])
        + ".\n";
    }
    if (temp.length > 0)
    {
      result += (i+1) + ". Semester:\n" + temp;
    }
  }
  if (result.length > 0)
    return result;
  else
    return "Es liegen noch keine Noten vor.";
}
```

In zwei Schleifen werden jeweils alle Semester und Fächer ausgegeben, für die eine Note vorliegt (das ist der Fall wenn die Note ungleich Null ist).

Damit die Note verständlich von der Sprachsynthese ausgegeben wird, muss sie vorher in Worte gewandelt werden. Diese Aufgabe wird von der Funktion `note2word()` erledigt:

```
function note2word (note)
{
      var result = "";

      if (note == -1)
            return "Testat bestanden";
      else if (note == -2)
            return "Testat nicht bestanden";

      var vorKomma = Math.floor(note);
      var nachKomma = Math.round((note - vorKomma) * 10.0);

      result = zahlen[vorKomma] + " Komma " + zahlen[nachKomma];

      return result;
}
```

Das Array `zahlen` enthält die Worte von Null bis Neun. Damit können einzelne Ziffern in Worte umgewandelt werden. Das Ergebnis der Funktion ist eine Wortfolge, z.B. ergibt der Wert 1.7 die Wortfolge „Eins Komma Sieben“.

Ausgabe der Noten eines Semesters

Bei der Ausgabe der Noten für ein einzelnes Semester wird die Funktion `listNotenSemester()` aufgerufen. Anstatt einer Schleife, die alle Semester durchläuft, wird hier nur das vom Benutzer vorher angegebene Semester verwendet. Ansonsten ist der Ablauf identisch mit dem für die Ausgabe aller Noten.

Ausgabe der Note für ein bestimmtes Fach

Möchte der Benutzer nur die Note für ein bestimmtes Fach erfahren, so wird er zuerst nach dem Semester gefragt. Dies ist zwar theoretisch unnötig, da jedes Fach nur einmal vorkommt, jedoch müsste der Spracherkenner in diesem Fall aus zu vielen Optionen gleichzeitig eine Wahl treffen und einige der Optionen wären sich sehr ähnlich, wie z.B. Mediengestaltung I und Mediengestaltung II. Aus diesem Grund wird die Auswahl vorher durch das Semester begrenzt.

Das anschließend erkannte Fach wird als Parameter der Funktion `listNotenKurs()` übergeben. Dort werden alle Namen der Fächer aus dem Array Noten mit dem übergebenen Namen verglichen. Bei einer Übereinstimmung wird die zugehörige Note ausgegeben.

Der Dialogablauf ist in Abbildung 57 dargestellt.

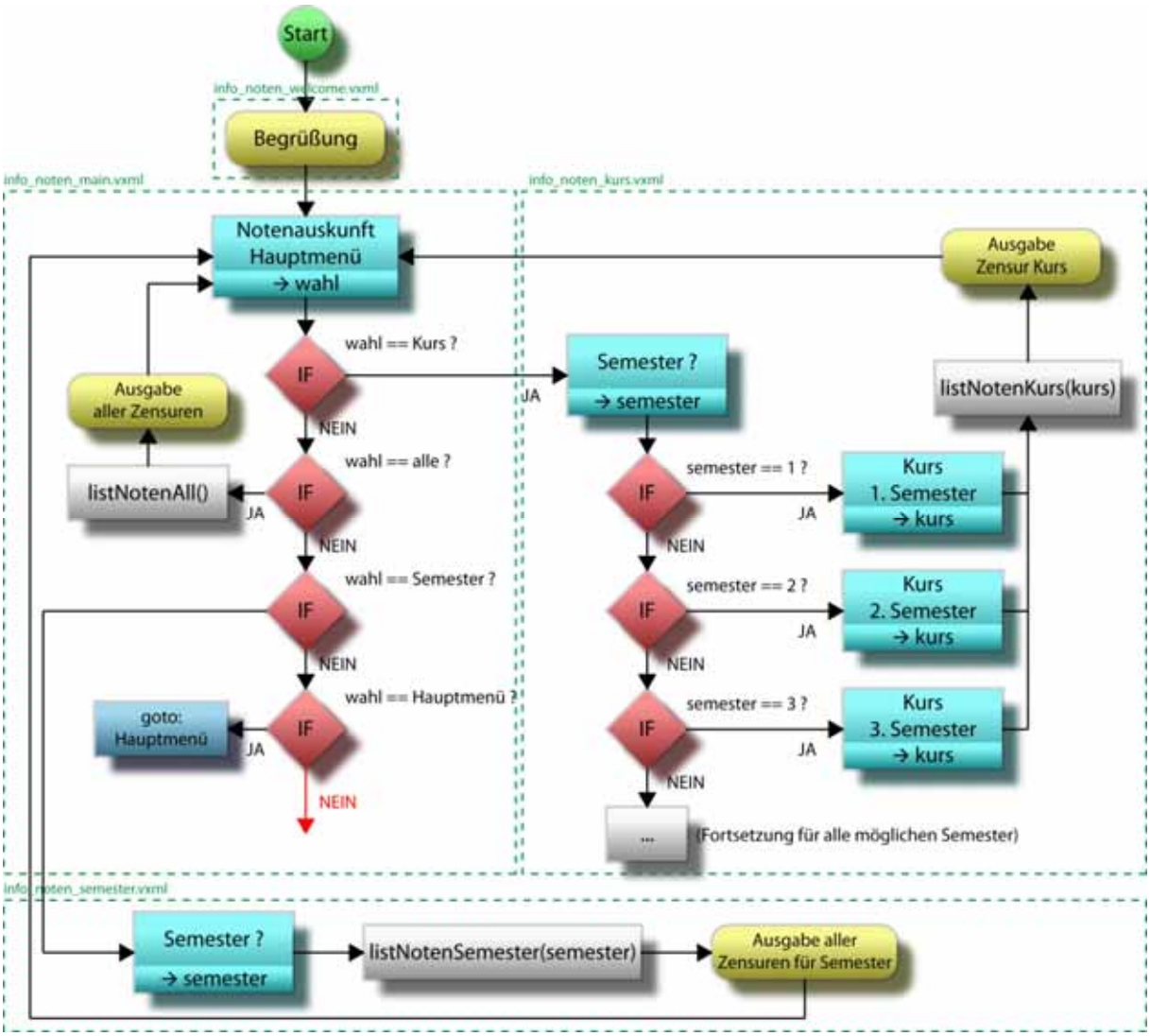


Abbildung 57: Schematischer Aufbau des Dialogs „Notenauskunft“

3.6.3.5 Authentifizierung

Für die korrekte Funktion des Informationsservers muss die Identität des Benutzers bekannt sein. Sie wird benötigt um den richtigen Datensatz für den aktuellen Benutzer, wie z. B. die Zensuren, auszuwählen. Weiterhin muss sichergestellt werden, dass ein Benutzer nur Zugriff auf seine eigenen Daten erhält um Missbrauch zu vermeiden.

Um die Identität eines Benutzers festzustellen ist eine Anmeldung erforderlich. Dabei werden eine Benutzeridentifikation und auch eine Benutzerverifikation durchgeführt.

Zeitpunkt der Anmeldung

Prinzipiell gibt es zwei Möglichkeiten für den Zeitpunkt der Anmeldung:

1. Sofort bei Beginn des Dialoges
2. Wenn die Identität benötigt wird

Die erste Möglichkeit, die Anmeldung sofort bei Beginn durchzuführen, hat den Vorteil, dass der Dialog eine gewisse Uniformität aufweist und der Benutzer sich bei mehrmaliger Verwendung besser auf den Dialogverlauf einstellen kann. Ein weiterer Vorteil ist, dass Benutzer, die eine Freisprecheinrichtung verwenden, den Hörer nach der Anmeldung mittels DTMF-Eingabe schnell wieder aus der Hand legen können. Der entscheidende Nachteil bei dieser Methode ist, dass der Benutzer sich immer Anmelden muss, auch wenn die Anmeldung eigentlich überflüssig ist, z.B. wenn der Benutzer nur die Mensaauskunft verwenden möchte.

Die zweite Möglichkeit besteht darin, die Anmeldung erst bei Bedarf durchzuführen. Dazu werden Punkte im Dialog festgelegt, an denen eine Anmeldung benötigt wird, z.B. beim Betreten der Noten-Auskunft. Hat sich der Benutzer bisher noch nicht identifiziert, so wird die Anmeldung jetzt durchgeführt. Ist die Identität des Benutzers bereits bekannt, so kann die Anmeldung übersprungen werden. Der Vorteil dabei ist, dass die Anmeldung nur dann durchgeführt wird, wenn sie wirklich benötigt wird. Der Nachteil ist, dass je nachdem wie der Benutzer durch den Dialog navigiert, die Anmeldung bei verschiedenen Sitzungen an anderen Stellen auftauchen kann. Dies kann unter Umständen zur Verwirrung des Benutzers führen.

Für den Testdialog wurde die Anmeldung bei Bedarf gewählt, da sich diese insgesamt als vorteilhafter erweist.

Realisierung der Anmeldung

Für die Art und Weise, wie die Anmeldung durchgeführt werden kann, gibt es auch mehrere Möglichkeiten. Die wichtigsten davon sind in Tabelle 15 aufgeführt.

Methode	Beschreibung	Vorteil	Nachteil
Benutzerkennung und PIN	Der Benutzer gibt eine Kennung (hier Matrikel-Nummer) an und bestätigt anschließend durch die Eingabe einer PIN-Nummer.	funktioniert zuverlässig, sicher	lange Eingabe nötig, zeitintensiv
Sprechererkennung und PIN	Das System ermittelt aus einer Äußerung des Benutzers automatisch die Kennung, der Benutzer bestätigt mit der Eingabe einer PIN-Nummer	schnell, sicher	nur bei geringem Nutzerkreis möglich, nahezu unmöglich bei ca. 8000 Studenten
Benutzerkennung und Sprecherverifikation	Der Benutzer gibt seine Kennung an, das System ermittelt aus einer Äußerung automatisch, ob es sich tatsächlich um diesen Benutzer handelt.	etwas schneller/ einfacher als mit PIN	Sicherheitsrisiko: Sprecherverifikation nicht ausreichend zuverlässig
Anrufer-ID und PIN	Das System ermittelt aus der Anrufer-Kennung (Telefonnummer) automatisch den Benutzer, dieser bestätigt mit Eingabe einer PIN	schnellste Methode, sicher	funktioniert nicht immer (z.B. bei analogen Anschlüssen, Rufnummer-Unterdrückung), Problematisch bei geteilten Anschlüssen

Tabelle 15: Verschiedene Arten der Anmeldung

Für den Beispieldialog wird eine Anmeldung mit Benutzerkennung und PIN verwendet. Eine Erweiterung des Demonstrators ist dazu nicht nötig. Sprechererkennung und Sprecherverifikation werden dagegen bisher noch nicht unterstützt. Eine Anmeldung durch Anrufer-ID und PIN ist für den späteren Einsatz aber auch denkbar. Die benötigten Änderungen am Demonstrator sollten nicht allzu schwer zu bewerkstelligen sein, denn die Nummer des Anrufers wird bereits über SIP übertragen und muss nur ausgewertet werden.

Der Sprachdialog kann bei der Anmeldung mit Benutzerkennung und PIN hinsichtlich der Übertragung der Nummern unterschiedlich aufgebaut sein. Einige der Möglichkeiten sind in Tabelle 16 aufgeführt.

Methode	Beispiel (Matrikel-Nr. 123456)	Bewertung
Ein Zahlwort	Hundertdreiundzwanzig-tausendvierhundertund-sechsundfünzig	Unrealistisch. Zu kompliziert für Erkenner. Umwandlung zu schwer für Benutzer.
Ziffern mit Pausen	1 2 3 4 5 6	Einfach für Erkenner. Unkompliziert und schnell für Benutzer. Nachteil: Benutzer hat keine Rückmeldung, wann Erkenner bereit. Bei Fehler muss Eingabe komplett wiederholt werden.
Ziffern mit Bestätigung	1 – Sie sagten eins. – 2 – Sie sagten zwei. - 3 …	Einfach für Erkenner. Benutzer bekommt Rückmeldung und kann falsche Ziffern korrigieren. Nachteil: zeitaufwändig.
Einzelabfrage	1 – Sie sagten eins, ist das korrekt? - Ja/Nein	Einfach für Erkenner. Fehler können leicht korrigiert werden. Nachteil: sehr zeitaufwändig

Tabelle 16: Mögliche Verfahren zur Eingabe von Zahlen

Für den Beispieldialog wurde die Methode „Ziffern mit Bestätigung“ gewählt, da der Demonstrator gelegentlich die Ziffern falsch erkennt und somit die Fehler schnell korrigiert werden können.

Der Dialog

Zu Beginn des Dialoges wird der Benutzer darauf hingewiesen, dass eine Anmeldung erforderlich sei und er nun bitte die Matrikel-Nr. angeben solle.

Dazu hat er neben der Angabe der Ziffern von Null bis Neun auch die Optionen „falsch“ und „fertig“. Sagt der Benutzer das Wort „falsch“, so wird die letzte erkannte Ziffer wieder gelöscht und er kann die Angabe wiederholen. Sagt er das Wort „fertig“ so wird die Eingabe der Matrikel-Nr. beendet. Dadurch ist es möglich, Matrikel-Nummern von unbestimmter Länge anzugeben.

Nach der Matrikel-Eingabe wird die vollständige Nummer nochmal ausgegeben und der Benutzer wird anschließend gefragt, ob das Ergebnis richtig sei. Antwortet er mit „nein“, so wird die Eingabe der Matrikel-Nr. wiederholt, andernfalls wird der Anmeldungsdialog mit der PIN-Abfrage fortgesetzt.

Die PIN-Abfrage ist in ihrer Struktur sehr ähnlich der Matrikel- Eingabe. Jedoch stehen hier die Optionen „falsch“ und „fertig“ nicht zur Verfügung. Die PIN-Nummer ist mit einer festen Länge von vier Stellen relativ kurz, somit bietet die Korrektur einzelner Ziffern nicht wirklich einen Vorteil gegenüber der kompletten Neueingabe und da die Stellenanzahl festgelegt ist, muss die Eingabe nicht vom Benutzer beendet werden. Sie endet automatisch, sobald vier Stellen erkannt wurden.

Hat der Benutzer die Eingabe der PIN-Nummer anschließend mit „ja“ bestätigt, so wird die Anmeldung durchgeführt. Falls die Matrikel-Nr. und die PIN-Nummer übereinstimmen, so wird der Benutzer auf die erfolgreiche Anmeldung hingewiesen und zu dem Bereich weitergeleitet, den er ursprünglich betreten wollte.

Schlägt die Anmeldung jedoch fehl, weil die angegebene Matrikel-Nr. nicht registriert ist oder die PIN nicht korrekt ist, so hat der Benutzer die Möglichkeit, die Anmeldung nochmal zu versuchen. Beantwortet er die Frage nach einem weiteren Versuch mit „ja“, so beginnt der Dialog erneut mit der Matrikel-Abfrage. Falls er mit „nein“ antwortet, so gelangt er zurück in das Hauptmenü.

Die Funktionalität der Anmeldung wurde wieder mit JavaScript umgesetzt. Die folgende Tabelle listet alle verwendeten Funktionen mit einer kurzen Erklärung auf:

Name der Funktion	Beschreibung
add (ziffer)	Fügt der Matrikel-Nr. eine Stelle hinzu.
redo ()	Entfernt die letzte Stelle der Matrikel-Nr.
getMatrikel ()	Liefert die vollständige Matrikel-Nr. zurück.
restart ()	Löscht die bisher eingegebene Matrikel-Nr.
addPin (ziffer)	Fügt der PIN-Nummer eine Stelle hinzu.
pinComplete ()	Überprüft, ob die PIN-Nummer vollständig ist. Beträgt die Stellenanzahl mindestens vier, so liefert die Funktion den Wert „true“ zurück, andernfalls „false“.
getPin ()	Liefert die vollständige PIN-Nummer zurück.
restartPin ()	Löscht die bisher eingegebene PIN-Nummer.
login()	Schlägt die angegebene Matrikel-Nr. im Benutzer-Array nach und vergleicht die angegebene PIN mit der dort hinterlegten. Bei Übereinstimmung wird der Wert „true“ zurückgeliefert. Stimmen die PINs nicht überein oder wurde die Matrikel -Nr. nicht gefunden, so lautet der zurückgegebene Wert „false“.

Tabelle 17: JavaScript-Funktionen des Anmelde-Dialogs

Die Anmeldung wurde in einer einzelnen VoiceXML-Datei untergebracht. Dies ist möglich, da durch den Aufruf

```
<goto nextitem="name"/>
```

es möglich ist, zu vorherigen <field>-Elementen zurückzukehren. Weiterhin hat der Benutzer bei diesem Dialog nicht die Möglichkeit, Bereiche zu überspringen, weshalb der lineare Ablauf ausreichend ist.

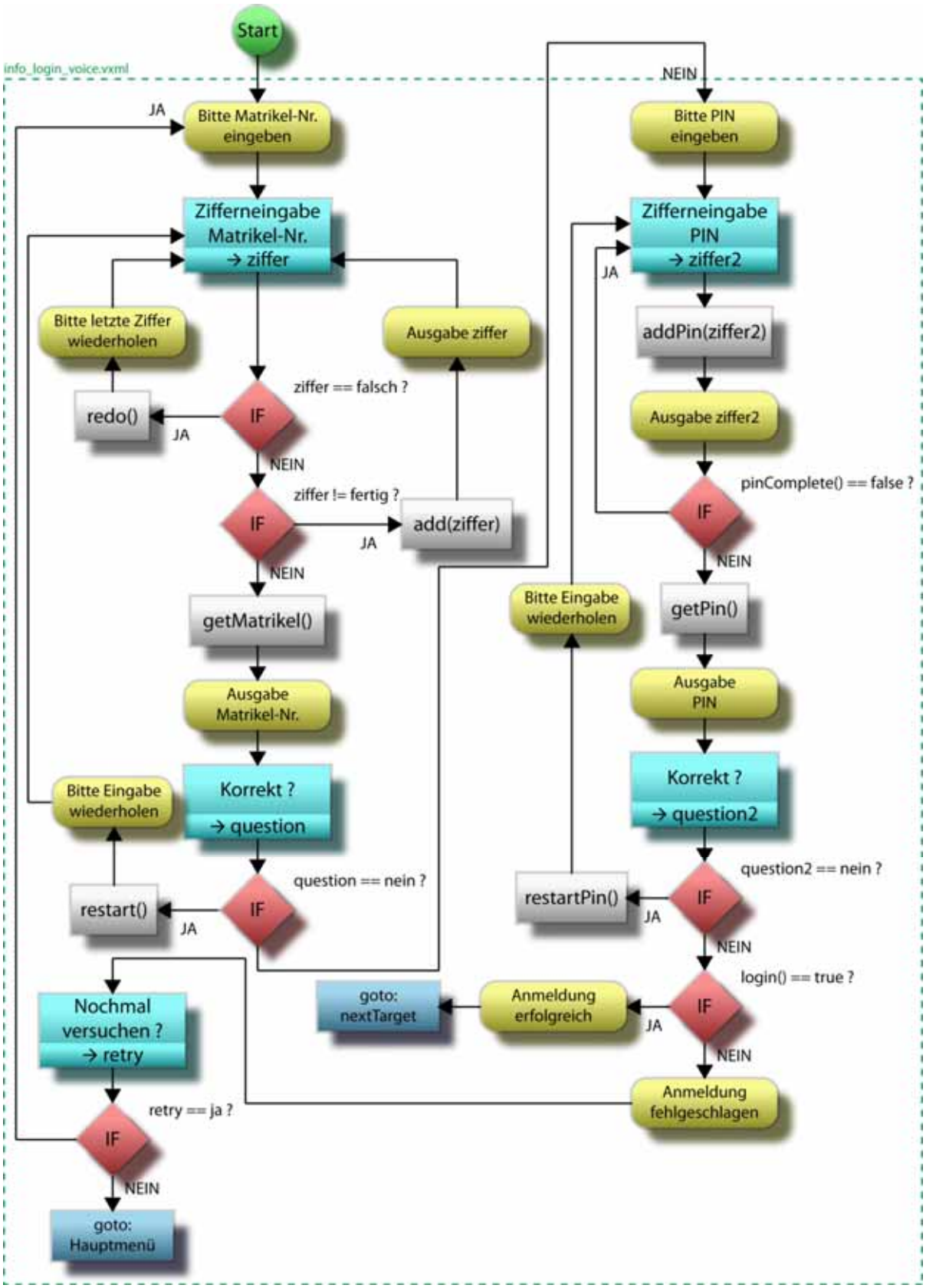


Abbildung 58: Schematischer Aufbau des Anmelde-Dialogs

3.6.3.6 Rückkehr zum Ausgangszustand

Damit der Dialog auch öfters verwendet werden kann, muss er nach einer Sitzung wieder in den Ausgangszustand zurückversetzt werden, d.h. der nächste Benutzer bekommt wieder die Begrüßung zu hören und befindet sich anschließend im Hauptmenü.

Die Funktion zum Zurücksetzen ist bereits implementiert: `fbsv.startThread(null)`. Sie wird aufgerufen, wenn ein lokaler Benutzer die „play"-Taste in der Benutzeroberfläche des Sprachservers betätigt. Durch den Aufruf wird der aktuelle Thread des VoiceXML-Parsers beendet und ein neuer gestartet. Dabei wird die vxml-Startdatei geladen, welche in der Konfigurationsdatei „fbsv.ini" angegeben ist.

Es gibt zwei Ereignisse, bei denen der Dialog des Demonstrators zurückgesetzt werden kann: Entweder bei einem eingehenden Anruf oder beim Beenden einer Sitzung, d.h. wenn der Benutzer aufgelegt hat.

In diesem Fall wurde der eingehende Anruf als Auslöser für die Zurücksetzung des Dialoges gewählt. Sinnvoller Weise ist der Demonstrator so konfiguriert, dass er eingehende Anrufe automatisch akzeptiert. Dabei wird die Funktion `runAutomaticAccept()` aus der Klasse `UserAgent` aufgerufen. Dort muss also nur noch die Funktion zum Zurücksetzen aufgerufen werden:

```
private void runAutomaticAccept(int delay_time)
{
      try {
            if (delay_time>0)
                  Thread.sleep(delay_time*1000);

            if (call!=null)
            {
                  printLog("AUTOMATIC-ANSWER");
                  fbsv.startThread(null);
                  accept();
            }
      }
      catch (Exception e) { e.printStackTrace(); }
}
```

Für den späteren realen Einsatz ist es außerdem wichtig, dass alle temporären Benutzerdaten gelöscht werden, wie etwa Matrikel-Nr. des letzten Benutzers, und dass geöffnete Datensätze, z.B. die Klausuranmeldungen, geschlossen werden.

Im Beispieldialog werden alle Daten in JavaScript-Variablen festgehalten. Durch den Aufruf von `fbsv.startThread(null)` werden diese bereits neu initialisiert, somit sind in diesem Fall keine weiteren Aktionen nötig.

4 Zusammenfassung

An dieser Stelle soll die Aufgabenstellung noch einmal ins Gedächtnis zurückgerufen werden: Die Aufgabe war es, die vorhandene VoiceXML-Plattform, genannt Friedberger Sprachserver (FBSV), um eine Voice-over-IP (VoIP)-Komponente zu erweitern, so dass der Sprachserver auch als automatisierter Telefon-Sprachdienst genutzt werden kann. Ferner sollte ein Dialog erstellt werden, mit der die Funktion des Systems demonstriert werden kann.

Bevor mit der Arbeit begonnen wurde, stand noch nicht fest, ob sich das gewünschte Ziel auf diese Weise und mit vertretbarem Aufwand umsetzen ließe. Nach einer anfänglichen Phase des Ausprobierens stellte sich dann aber rasch der erste Erfolg ein. Die über das Netzwerk eingehenden Sprachdaten konnten erfolgreich an den Spracherkenner umgeleitet werden. Zu diesem Zeitpunkt war klar, dass die Umsetzung des Konzeptes möglich ist. Da die Umleitung der ausgehenden Sprachdaten einen dazu analogen Prozess darstellt, war mit keinen neuen Problemen zu rechnen.

Der Umfang des für die VoIP-Anbindung benötigten Programmcodes ist relativ gering. Der aufwändigere Teil war die Analyse des vorhandenen Quellcodes, um den internen Fluss der Sprachdaten zu verstehen und eine geeignete Stelle zu finden, wo die Daten eingespeist bzw. abgegriffen werden konnten.

Nachdem die Verbindung zwischen Sprachserver und VoIP-Client stand, musste das Format der Audiodaten angepasst werden. Dazu mussten nur ein paar Parameter geändert werden.

Durch die Verwendung des Codecs G.711 bei der Audioübertragung entsteht kaum ein Qualitätsverlust. Die Daten werden zwar etwas reduziert, aber es entstehen dabei keine Kompressionsartefakte, die sich negative auf die Erkennung auswirken könnten.

Probleme bereitete die Umstellung der Abtastrate von 16 kHz auf 8 kHz. Die Hidden Markov Modelle mussten nochmal mit Eingangsdaten in 8 kHz erstellt und trainiert werden. Die Konfiguration des Erkenners musste entsprechend angepasst werden. Durch die Umstellung auf 8 kHz sank die Erkennungsrate um wenige Prozent (etwa um 2 – 3 %).

Die Erkennung funktionierte soweit auch über eine Telefonverbindung und es konnte mit der Entwicklung eines Beispieldialoges begonnen werden. Als Szenario für den Beispieldialog wurde ein FH Informationsdienst gewählt. Der Dialog wurde dabei in VoiceXML unter Zuhilfenahme eines einfachen Texteditors erstellt. Die speziellen Funktionen des Dienstes wurden mit JavaScript-Funktionen simuliert, z.B. wird bei der Notenauskunft eine Datenbankabfrage simuliert.

Für den Dialog war es nötig, den Erkenner von Wortbasierter Erkennung auf Phonembasierte Erkennung umzustellen, da sonst nur die wenigen Worte zu erkennen gewesen wären, für die auch Modelle erstellt und trainiert worden wären. Jetzt konnten alle Worte erkannt werden, deren Aussprache im Wörterbuch hinterlegt war. Die Aussprache der einzelnen Worte wurde aus dem bestehenden Wörterbuch Hadi-Bomp entnommen.

Durch die Umstellung auf Phonembasierte Erkennung zeigte sich dann der größte Einbruch in der Erkennungsrate. In einigen Spracheingabe-Menüs wurden bestimmte Optionen bevorzugt, während andere Optionen kaum erkannt wurden.

Die Ursache dafür ist, dass beim Training der Phonemmodelle nur eine geringe Menge an Trainingsmaterial verwendet wurde. Für gute Erkennungsergebnisse muss das Training entweder mit umfangreichem allgemeinem Material durchgeführt werden oder es muss Material verwendet werden, welches dem späteren Einsatzgebiet entspricht.

Besonders die Durchführung des Anmeldedialogs gestaltete sich mit der mäßigen Erkennungsrate schwierig. Für Abhilfe sollte hier ein DTMF-Erkenner sorgen. Dual Tone Multiple Frequency (DTMF) ist ein Verfahren zur Übertragung der gewählten Rufnummer oder auch der gedrückten Tasten. Der Benutzer hat also die Möglichkeit, Ziffernfolgen wie Matrikel-Nr. und PIN direkt über die Tastatur einzugeben. Die DTMF-Erkennung kann aber auch für normale Dialogmenüs verwendet werden, indem das Drücken einer Taste einer bestimmten Option zugewiesen wird.

Die Implementierung des DTMF-Erkenners gestaltete sich einfach. Für die eigentliche Signalanalyse wurde ein externes Programm hinzugezogen. Der Sprachserver hat die aufgezeichnete Audiodatei an das Programm übergeben und anschließend das Ergebnis aus einer Datei eingelesen.

Abschließend kann man sagen, dass die VoiceXML-Plattform durch die VoIP-Anbindung zu einem vollwertigen Sprachdialogsystem wurde, das auch über das Telefon erreichbar ist. Die Verbindung kann sowohl online über VoIP, als auch über das Festnetz mittels SIP-Gateway hergestellt werden. Bevor das System in der Praxis eingesetzt werden kann, sollten aber noch bessere Phonemmodelle organisiert werden.

5 Verbesserungsmöglichkeiten

Bei der Durchführung der Arbeit sind ein paar Punkte aufgefallen, an denen der Sprachserver und der Beispieldialog noch verbessert werden könnten. Die wichtigsten Verbesserungsmöglichkeiten werden im Folgenden dargestellt.

Datenbankabfrage
Im Beispieldialog wurden die Informationen wie Klausuranmeldungen und Noten in JavaScript-Feldern hinterlegt. Für den praktischen Einsatz ist diese Lösung nicht geeignet. Die benötigten Informationen sollten aus einer Datenbank z. B. mit MySQL abgefragt werden. Dazu ist vorher aber eine dynamische Generierung der VoiceXML-Dateien nötig.

Dynamischer Dialog
Bisher wird der Dialogablauf fest in den VoiceXML-Dateien definiert. So müssen z. B. auch alle möglichen Antworten im Dialog für die Klausuranmeldung hinterlegt werden. Das hat den Nachteil, dass auch bereits angemeldete Klausuren erkannt werden können. In diesem Fall wäre es besser, wenn die Auswahlmöglichkeit auf die noch nicht angemeldeten Klausuren begrenzt wäre. „Dynamisch“ bedeutet, dass die VoiceXML-Dateien erst zur Laufzeit generiert werden und so dem aktuellen Zustand angepasst werden. Die Generierung der VoiceXML-Dateien könnte dabei von einem Webserver mittels PHP übernommen werden.

Sprachausgabe überspringen
Viele Sprachdialogsysteme bieten dem Benutzer die Möglichkeit an, Sprachausgaben zu überspringen. Dies kommt vor allem Benutzern zu Gute, die den Dialog schon mehrmals verwendet haben. Durch Überspringen schon bekannter Sprachausgaben kann der Ablauf des Dialoges beschleunigt werden. Dabei gibt es die Möglichkeit, die Sprachausgabe bei einer beliebigen Äußerung des Benutzers oder bei Nennung eines bestimmten Schlüsselwortes abzubrechen. Man spricht dabei von einem „barge in“ (engl. für hereinplatzen, sich einmischen). Die Möglichkeit des Überspringens kann bei VoiceXML im <prompt>-Element angegeben werden: <prompt bargein=“true“>. Der Sprachserver unterstützt dies bisher jedoch nicht.

Sprachaufzeichnung
In VoiceXML ist auch die Möglichkeit zur Sprachaufzeichnung vorgesehen. Durch die Angabe von <record> wird das System angewiesen, eine Äußerung des Benutzers aufzuzeichnen. Diese Fähigkeit wäre sehr nützlich für das „digitale Schwarze Brett“, welches im Kapitel Beispieldialog als möglicher Dienst für den FH Informationsserver erwähnt wurde. Zudem könnten Benutzer des Systems Sprachnachrichten, sogenannte Voicemails, für andere Benutzer hinterlassen.

Neuabfrage bei nicht eindeutigem Ergebnis
Der Sprachserver sucht sich bisher immer eine der angegebenen Antwortmöglichkeiten aus, auch wenn die Äußerung des Benutzers auf keine der Optionen zutrifft. Eine Verbesserung wäre hier, den Sprachserver in zwei Fällen die Spracheingabe wiederholen zu lassen: Erstens, die Wahrscheinlichkeit für alle Optionen ist sehr gering und zweitens, die Wahrscheinlichkeiten der besten und zweitbesten Alternative liegen sehr nahe beieinander.

Mehrbenutzer-Betrieb

Der Demonstrator ist zurzeit so konzipiert, dass nur ein Benutzer gleichzeitig mit dem System verbunden sein kann. Versucht ein weiterer Benutzer das System anzuwählen, so erhält er ein Besetztzeichen. Theoretisch ist es möglich, über VoIP mehrere Verbindungen laufen zu lassen, allerdings sind dazu im Demonstrator weitgreifende Änderungen nötig.

6 Literaturverzeichnis

[Wik082] **Wikipedia**[23]**.** Codec. [Online] 3. Juli 2008. [Zitat vom: 11. Juli 2008.] http://de.wikipedia.org/wiki/Codec.

[Wik089] —. Entwurfsmuster. [Online] 12. Juli 2008. [Zitat vom: 24. Juli 2007.] http://de.wikipedia.org/wiki/Entwurfsmuster.

[Wik084] —. G.711. [Online] 4. Mai 2008. [Zitat vom: 11. Juli 2008.] http://de.wikipedia.org/wiki/G.711.

[Wik07] —. G.729. [Online] 19. Juni 2007. [Zitat vom: 11. Juli 2008.] http://de.wikipedia.org/wiki/G.729.

[Lin08] **Linguatec.** Grundlagen der Spracherkennung. *Linguatec Sprachtechnologien.* [Online] 22. Januar 2008. [Zitat vom: 29. Juni 2008.] http://www.linguatec.net/products/stt/information/basics.

[Wik081] **Wikipedia.** Hidden Markov Model. [Online] 19. Juni 2008. [Zitat vom: 1. Juli 2008.] http://de.wikipedia.org/wiki/Verborgenes_Markow-Modell.

[OzV08] **OzVoIP.com.** Introduction to Codecs. *OzVoIP.com.* [Online] [Zitat vom: 11. Juli 2008.] http://www.ozvoip.com/codecs.php.

[Wik071] **Wikipedia.** IntServ. [Online] 5. September 2007. [Zitat vom: 12. Juli 2008.] http://de.wikipedia.org/wiki/IntServ.

[IPT08] —. IP-Telefonie. [Online] 12. Juni 2008. [Zitat vom: 29. Juni 2008.] http://de.wikipedia.org/wiki/IP-Telefonie.

[Wik085] —. Jitter. [Online] 21. Mai 2008. [Zitat vom: 12. Juli 2008.] http://de.wikipedia.org/wiki/Jitter.

[Wik083] —. Mean Opinion Score. [Online] 4. Juli 2008. [Zitat vom: 11. Juli 2008.] http://de.wikipedia.org/wiki/Mean_Opinion_Score.

[Wik08] —. Mehrfrequenzwahlverfahren. [Online] 3. Mai 2008. [Zitat vom: 29. Juni 2008.] http://de.wikipedia.org/wiki/Mehrfrequenzwahlverfahren.

[Wik086] —. Quality of Service. [Online] 2. Juli 2008. [Zitat vom: 12. Juli 2008.] http://de.wikipedia.org/wiki/Qos.

[Ses08] —. Session Initiation Protocol. [Online] 27. Juni 2008. [Zitat vom: 29. Juni 2008.] http://de.wikipedia.org/wiki/Session_Initiation_Protocol.

[Wik088] —. Singleton (Entwurfsmuster). [Online] 20. Juli 2008. [Zitat vom: 24. Juli 2008.] http://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster).

[23] steht stellvertretend für die an einem Artikel der freien Enzyklopädie Wikipedia beteiligten Autoren. Die Liste der Autoren kann auf der jeweiligen Seite eingesehen werden.

[Spr08] —. Spracherkennung. [Online] 27. Juni 2008. [Zitat vom: 29. Juni 2008.] http://de.wikipedia.org/wiki/Spracherkennung.

[Spr081] —. Sprachsynthese. [Online] 24. Juni 2008. [Zitat vom: 29. Juni 2008.] http://de.wikipedia.org/wiki/Sprachsynthese.

[Wik087] —. Traffic-Shaping. [Online] 25. Mai 2008. [Zitat vom: 12. Juli 2008.] http://de.wikipedia.org/wiki/Traffic_Shaping.

[Voi08] —. VoiceXML. [Online] 27. Juni 2008. [Zitat vom: 29. Juni 2008.] http://de.wikipedia.org/wiki/VoiceXML.

[Vox08] **Voxeo Corporation.** VoiceXML Development Guide - Version 2.1. *Vxml.org.* [Online] 2008. [Zitat vom: 29. Juni 2008.] http://www.vxml.org/.

[3CX08] **3CX.** Was bedeutet SIP? *3CX VOIP Telefonanlage für Windows.* [Online] [Zitat vom: 29. Juni 2008.] http://www.3cx.de/voip-sip/sip.php.

[Bur08] **Burkhardt, Felix.** Deutsche Sprachsynthese. *SyntheticSpeech.de.* [Online] 14. April 2008. [Zitat vom: 29. Juni 2008.] http://ttssamples.syntheticspeech.de/deutsch/index.html.

[Gur05] **Gurow, Lars.** Skype vs. SIP - Internet-Telefone unter sich. *Netzwelt.* [Online] 29. April 2005. [Zitat vom: 29. Juni 2008.] http://www.netzwelt.de/news/71004_2-skype-vs-sip-internettelefone.html.

[Haa03] **Haas, Felicitas.** MiLCA - Sprachsynthese. *Universität Bonn - Abteilung Sprache und Kommunikation.* [Online] Oktober 2003. [Zitat vom: 29. Juni 2008.] http://www.ikp.uni-bonn.de/dt/lehre/Milca/mmk/index_mmk.html.

[Hab98] **Haberland, Nils, et al.** Wie funktioniert computerbasierende Spracherkennung? *c't - Computer und Technik.* 05 1998, S. 120 ff.

[Hen01] **Henne, Sebastian.** Grundlagen der Spracherkennung. *FH Wedel - University of Applied Sciences.* [Online] 12. September 2001. [Zitat vom: 29. Juni 2008.] http://www.fh-wedel.de/~si/seminare/ss01/Ausarbeitung/a.sprache/gdlgsprerk00.htm.

[Krö08] **Kröner, Tim.** SIP (Session Initiation Protocol). *Voip-Information.de.* [Online] [Zitat vom: 29. Juni 2008.] http://www.voip-information.de/sip/sip.php.

[Krö081] —. Was ist VoIP? *Voip-Information.de.* [Online] [Zitat vom: 29. Juni 2008.] http://www.voip-information.de/voip-voice-over-ip.html.

[Mei07] **Meiert, Jens.** Einführung in VoiceXML 2.0. *Meiert.com.* [Online] 20. Juli 2007. [Zitat vom: 29. Juni 2008.] http://meiert.com/de/w3/Voice/Guide/.

[Meß07] **Meßner, Florian.** Einführung in VoIP: Grundlagen und Praxis-Tipps für Privatanwender zum Einstieg in die Internet-Telefonie mit Hinweisen für Businessanwender. [Online] 15. März 2007. [Zitat vom: 29. Juni 2008.] http://www.florianmessner.com/support/themen/voip/index.htm.

[Möb02] **Möbius, Bernd.** Sprachsynthese I. *Universität Stuttgart: Institut für Maschinelle Sprachverarbeitung.* [Online] 21. Februar 2002. [Zitat vom: 29. Juni 2008.] http://www.ims.uni-stuttgart.de/lehre/teaching/2001-WS/Synthese1/.

[Nöl06] **Nölle, Jochen.** Voice over IP (VoIP) - Eine Einführung. *VoIP-Info.de.* [Online] 2006. [Zitat vom: 29. Juni 2008.] http://www.voip-info.de/wissen/_Artikel_Allgemein_32.php.

[Pre08] **Prelle, Stefan.** Grundlagen der IP-Telefonie. *WIPTEL.* [Online] [Zitat vom: 29. Juni 2008.] http://www.wiptel.de/docs/basics.html.

[Aud08] **Schnabel, Patrick.** Audio-Codecs zur Sprachdigitalisierung (Voice over IP). *Elektronik Kompendium.* [Online] [Zitat vom: 29. Juni 2008.] http://www.elektronik-kompendium.de/sites/net/0905121.htm.

[VoI08] —. VoIP - Voice over IP. *Elektronik Kompendium.* [Online] [Zitat vom: 12. Juli 2008.] http://www.elektronik-kompendium.de/sites/net/0503131.htm.

[Sch08] **Schulzrinne, Henning.** Session Initiation Protocol (SIP). *Columbia University Department of Computer Science.* [Online] 28. März 2008. [Zitat vom: 29. Juni 2008.] http://www.cs.columbia.edu/sip/.

[Tra97] **Traunmüller, Hartmut.** Geschichte der Sprachsynthese. *Universität Stockholm.* [Online] August 1997. [Zitat vom: 29. Juni 2008.] http://www.ling.su.se/staff/hartmut/kempln.htm.

[Wor04] **W3C.** Voice Extensible Markup Language (VoiceXML) Version 2.0. *W3C - World Wide Web Consortium.* [Online] 16. März 2004. [Zitat vom: 29. Juni 2008.] http://www.w3.org/TR/voicexml20/.

[War99] **Warth, Dora.** Künstliche Intelligenz: Spracherkennung und Sprachverstehen. *Uni Mainz: Fachbereich Angewandte Sprach- und Kulturwissenschaft.* [Online] 19. August 1999. [Zitat vom: 29. Juni 2008.] http://www.fask.uni-mainz.de/user/warth/Ki.html.

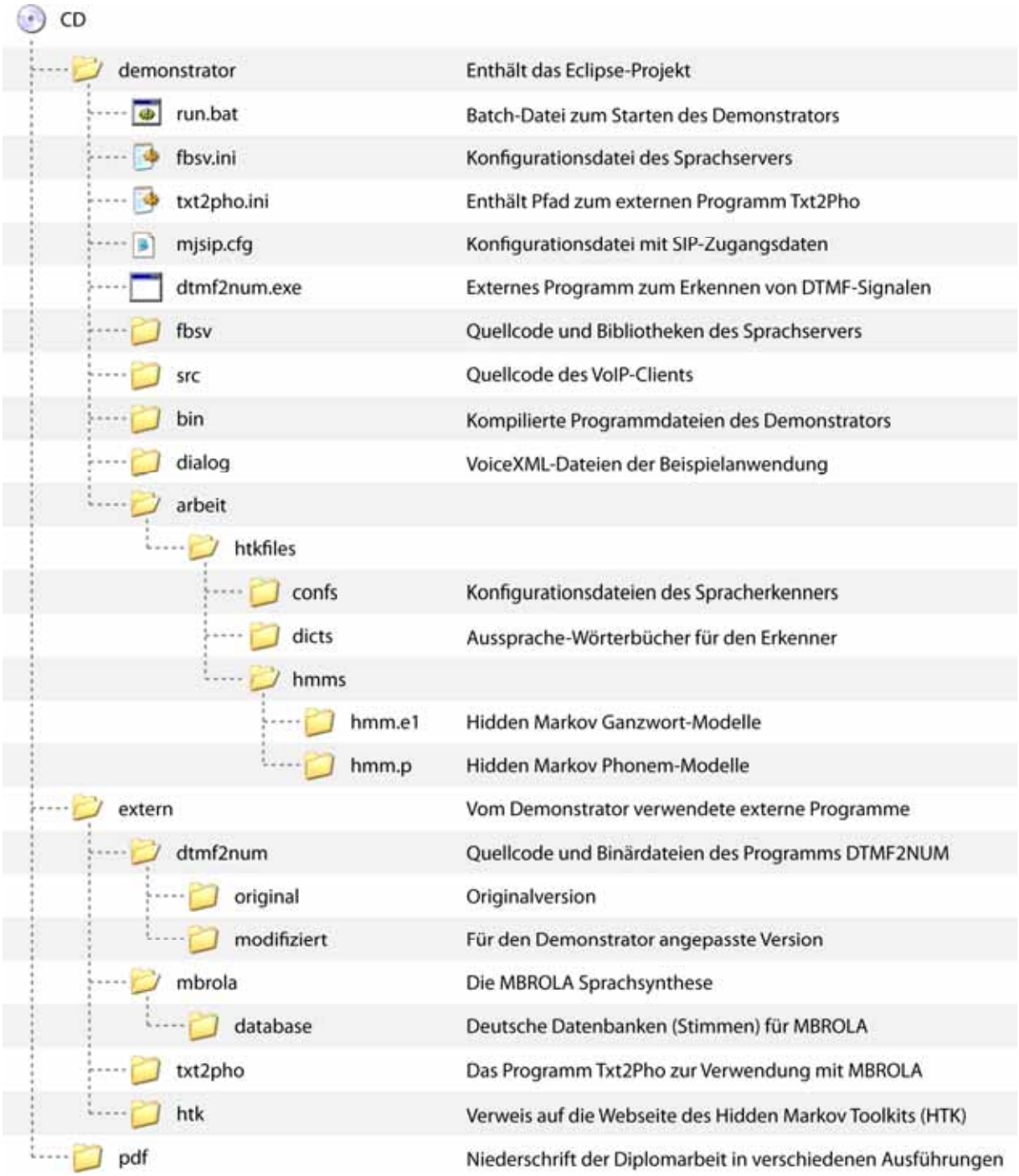
7 Abbildungsverzeichnis

Alle Abbildungen ohne Angabe der Herkunft wurden von mir persönlich für diese Diplomarbeit erstellt.

Anhang

A Inhalt der CD

Die nachfolgende Grafik gibt einen kurzen Überblick über die auf der CD befindlichen Dateien. Es sind nur die wichtigsten Dateien und Ordner dargestellt.



B Hinweise zum Ausführen des Demonstrators

Der Demonstrator kann durch einen Doppelklick auf die Datei „run.bat“ im Ordner „demonstrator“ gestartet werden. Für ein korrektes Arbeiten des Demonstrators sind jedoch einige Punkte zu beachten:

- Es muss eine JAVA Laufzeitumgebung installiert sein. Eine aktuelle Version kann unter http://www.java.com bezogen werden.
- Die externen Programme müssen, wie in Punkt 3.3.2 und 3.3.3 dargestellt, installiert sein.
- Da zur Laufzeit einige temporäre Dateien angelegt werden, muss der Demonstrator von einem Medium mit Schreibzugriff gestartet werden. Dazu muss der Ordner „demonstrator“ von der CD z. B. auf eine Festplatte oder einen USB-Stick kopiert werden.
- Es sollten einige Megabyte an freiem Speicherplatz für die temporären Dateien vorhanden sein.
- Für die VoIP-Funktionalität werden ein Internetanschluss sowie ein SIP-Konto benötigt. Die SIP-Zugangsdaten müssen in der Datei „mjsip.cfg“ im Ordner „demonstrator“ eingetragen werden.