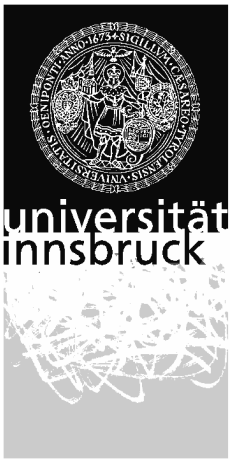


Leopold-Franzens-Universität
Innsbruck

Institut für Informatik
**Networking**

# VoIP für Extremsituationen

**Bakkalaureatsarbeit**

eingereicht bei
Dr. Dipl.-Ing. Michael Welzl

eingereicht von
Elmar Weiskopf
0215215
csae1328@uibk.ac.at

Innsbruck, 10.11.2006

# Inhaltsverzeichnis

# 0. Vorbetrachtung

## 0.1 Einleitung

VoIP (Voice over IP), also Telefonieren über IP-Netzwerke, führt zu einer Veränderung unseres klassischen Bildes der Telefonie. Mit der zunehmenden Anzahl von Breitbandverbindungen und populären Applikationen konnte sich diese Form der Kommunikation in den letzten Jahren vom Nischenprodukt zu einer ausgereiften Technologie neben der klassischen Telefonkommunikation etablieren. Daneben macht auch die Tatsache, dass für Gespräche innerhalb eines IP-Netzes - außer den üblichen Kosten für die Internetverbindung - keine zusätzlichen Gesprächsgebühren anfallen, VoIP so populär. Auch die durchwegs gute Gesprächsqualität, auch über große Distanzen, trägt zu seiner Beliebtheit bei.

Gerade der 11. September 2001 hat jedoch offenbart, dass auch in Zeiten globaler und lokaler Vernetzung Telekommunikationsstrukturen nicht unverwundbar sind und zumindest teilweise ausfallen können. Aufgrund der dramatischen Ereignisse brach damals in New York kurzerhand das gesamte Telekommunikationssystem zusammen, wodurch immens wichtige Informationsflüsse (z.B. Notrufe) gestört wurden bzw. überhaupt nicht zustande kamen. Dieser hochtechnisierten Stadt wurden dabei die Grenzen ihres ausgezeichnet entwickelten Kommunikationsnetzes aufgezeigt, wodurch eine effektivere Ausnützung vorhandener Bandbreite neu diskutiert wurde.

## 0.2 Aufgabenstellung

Wie der Name dieser Arbeit bereits verrät, sollte eine Anwendung entwickelt werden, die neben einer traditionellen, funktionsfähigen Sprachübermittlung auch einen alternativen, sehr bandbreitenschonenden Ansatz verfolgt: Einfach gesagt, sollte mit Hilfe einer Spracherkennung menschliche Sprache in normalen ASCII-Text umgewandelt, dieser über ein IP-Datennetz (Internet) übertragen und beim Gegenüber wieder in menschenähnliche Sprache transformiert werden.

Aufgrund der viel geringeren Beanspruchung eines Datennetzes durch Textübertragung, könnte eine solche Anwendung in einigen Situationen ihren Dienst verrichten. Bei Netzausfällen könnten beispielsweise wichtige Informationen nur bruchstückhaft oder gar nicht übermittelt werden. Mit dem Konzept purer Textübermittlung ließe sich bereits mit nur einem Kilobyte Nutzdaten mehrere vollständige Sätze übertragen, wohingegen diese Datenmenge bei herkömmlicher Sprachübertragung für sinnvolle Informationen nicht ausreicht. In Verbindung mit einer Spracherkennung und einer Sprachsynthese könnte dieses Übertragungskonzept quasi einen Notfallersatz einer normalen Telefonanlage darstellen. Dem Nachteil ungenauer Spracherkennung bzw. unnatürlicher

maschinengenerierter Spracherzeugung steht ein äußerst geringes Datenaufkommen gegenüber.

Diese Arbeit gliedert sich in mehrere Teilbereiche:

Als erstes wird die Entwicklungsgeschichte und die grundsätzliche Funktionsweise von VoIP erläutert.

Im nächsten Kapitel behandelt die Arbeit den Aufbau eines Systems mit Textübermittlung und Spracherkennung/Sprachsynthese. Überdies werden Teilkomponenten beschrieben, die in diesem Projekt verwendeten wurden.

Die konkrete Realisierung des Programmes „VoIP4Text“ mit Klassenbeschreibungen und Diagrammen bzw. Screenshots wird im 3. Kapitel behandelt.

Das 4. Kapitel beschreibt den durchgeführten Funktionstest mit Illustration der Netzwerkstrecke.

Schließlich bietet nach projektspezifischen Informationen und einer kurzen Zusammenfassung das 6. und letzte Kapitel eine Einführung bzw. Hilfestellung für das Programm „VoIP4Text“ in Form einer Readme-Datei.

## 0.3 Danksagung

Meinen Dank aussprechen möchte ich Michael Welzl, der bei Problemen helfend zur Seite stand und zu jeder Zeit als Ansprechpartner verfügbar war.

Weiters danke ich Nathalie Steinmetz für die Bereitstellung der IBM - Java Bibliotheken.

Vielen Dank!

# 1. VoIP

## 1.1 Geschichte der Telefonie

Seit den ersten Versuchen von Graham Bell 1876 bis zur heutigen fortschrittlichen VoIP Technik hat sich so manches an den technischen Grundlagen der Telefonie geändert. Anfangs bestand nur ein fixes Kabel als Verbindung zwischen 2 Geräten, es war noch keine Rede von freier Auswahl zwischen Gesprächsteilnehmern und effektiver Verkabelung. Als erste weiterführende Maßnahme wurde ein eigener Telefonoperator beschäftigt, der ankommende Gespräche von einem zentralen Standort aus manuell mit einer anderen Leitung verband. Diese Arbeit übernahm später ein elektronisches Gerät, der Switch. Die damals noch analoge Übertragungsform war sehr anfällig für „Noise“, also für Störungen des Originalsignals durch externe Quellen bzw. aufgrund schlecht abgeschirmter Verkabelung. Ab 1980 wurde das Telefonnetz, das allgemein auch als PSTN (Public Switched Telephone Network) bezeichnet wird, nach und nach digitalisiert, sodass auch andere Dienste (wie später das Internet) diese Leitungen nutzen konnten. Mit der PCM-Technologie (Pulse Code Modulation) als digitale Repräsentation analoger Sprachsignale konnte die Übertragungsqualität gegenüber der reinen analogen Übermittlung eindeutig verbessert werden. Aktuelle PCM-Standards sind μ-law (Amerika, Japan) und a-law (Europa).

## 1.2 Entstehung von VoIP

Noch vor der Erstellung des Internet Protokolls[1] (IP) wurde 1973 das Network Voice Protocol (NVP) verfasst, das bereits auf paketbasierenden Netzwerken aufbaut. Diese Technologie testete man damals im ARPANET, das als eine der ersten Formen des heutigen Internets verstanden werden kann. Das Protokoll unterschied zwischen Daten- und Kontrollnachrichten (Ruf-, Annahme- und Auflegesignale). Im Jahre 1980 wurde der UDP-Standard[2] (User Datagram Protocol) verabschiedet und 1981 die Entwicklung des Internet-Protokolls abgeschlossen. Beide Konzepte wurden als Standards für verbindungslosen, paketorientierten Datenverkehr anerkannt. Mit Einführung der ISDN-Technologie (Integrated Services Digital Network) Mitte der 80er Jahre fanden auch die von der ITU[3] (International Telecommunication Union) standardisierten Konzepte μ-law und a-law erste Verwendung. Im Jahr 1989 entwickelte Tim Berners-Lee im CERN[4] (Europäische Organisation für Kernforschung) das World Wide Web. 1996 wurde mit RTP[5] (Real-Time Transport Protocol) die Grundlage der heutigen VoIP-Technologie geschaffen. Es dient dazu, Multimediainhalte wie Audio und Video zu kodieren und über ein IP-Netzwerk zu senden. 1998 wurde die allgemein gültige Übertragungsempfehlung von

[1] http://www.ietf.org/rfc/rfc791.txt, http://www.ietf.org/rfc/rfc2460.txt (Version 6)
[2] http://www.ietf.org/rfc/rfc768.txt
[3] http://www.itu.int/
[4] http://www.cern.ch/
[5] http://www.ietf.org/rfc/rfc3550.txt

Multimediainhalten über IP-basierte Netzwerke H.323 und 1 Jahr später das SIP[6] (Session Initiation Protocol) verabschiedet, das sich um die grundlegende Kommunikation zwischen mehreren Teilnehmern kümmert. Seit damals führt die zunehmende Verfügbarkeit von Breitbandanschlüssen zu einer stetig wachsenden Verwendung der VoIP-Technologie.

## 1.3 Ablauf von VoIP Gesprächen

Im grundlegenden Ablauf unterscheidet sich VoIP nicht von normaler Telefonie. Auch hier erfolgt zuerst ein Verbindungsaufbau, dann die eigentliche Kommunikation und schließlich der Verbindungsabbau. Anders als bei der ursprünglichen Form gibt es jedoch keine feste Adressierung der Teilnehmer (zum Beispiel anhand der IP-Adresse). Dies erledigt ein Server, der die aktuellen IP-Adressen der teilnehmenden Geräte speichert. Mit Hilfe vom SIP, das jedem Teilnehmer einen URI[7] (Uniform Resource Identifier), ähnlich einer E-Mail Adresse zuordnet, werden Basisinformationen über die zu errichtende Verbindung ausgetauscht. Das Problem der Verbindung von Geräten mit nicht statischer IP-Adresse wird derzeit (2006) diskutiert und es bestehen verschiedene Lösungsansätze:

- Verwendung gleich bleibender SIP-Adressen
- Verwendung des ENUM-Verfahrens: Umkehrung einer bestehenden Rufnummer mit Einsetzen von Punkten zwischen den Nummern und anschließendem Anhängen der Domain „e164.arpa“. Zum Beispiel wird aus der Telefonnummer +43 512 112233 3.3.2.2.1.1.2.1.5.3.4.e164.arpa.
- Verwendung herkömmlicher Ortsrufnummern

Bei einem laufenden Gespräch wird das von einem Mikrofon kommende analoge Signal digitalisiert und meistens durch Kompression in der Datenmenge verkleinert. Mit RTP, das auf RTCP Befehle (Real-Time Control Protocol) reagiert, werden UDP Pakete zum Gegenüber gesendet, wo die Dekomprimierung und eine Umwandlung digitaler in hörbare Signale stattfindet.

## 1.4 Anfallende Datenmengen

Für die Sprachübertragung mit VoIP können verschiedene Komprimierungsstandards verwendet werden, wobei teilnehmende Gesprächspartner die gleiche Technologie einsetzen müssen. Bei der Komprimierung werden für das menschliche Gehör nicht wahrnehmbare Frequenzen entfernt. Der ausgewählte Komprimierungsgrad bestimmt demnach die Sprachqualität und die anfallende Upload- bzw. Download-Datenmenge.

[6] http://www.ietf.org/rfc/rfc3261.txt
[7] http://www.ietf.org/rfc/rfc3986.txt

| Codec | Bitrate Codec | Bitrate Ethernet |
|---|---|---|
| G.711 | 64 kBit/s | 88 kBit/s |
| G.722 | 64 kBit/s | 88 kBit/s |
| G.723 | 6,4 kBit/s | 22 kBit/s |
| G.726-24 | 24 kBit/s | 47 kBit/s |
| G.726-32 | 32 kBit/s | 55 kBit/s |
| G.726-40 | 40 kBit/s | 64 kBit/s |
| G.728 | 16 kBit/s | 32 kBit/s |
| G.729 | 8 kBit/s | 32 kBit/s |
| iLBC | 13,3 kBit/s | 27 kBit/s |
| GSM | 13 kBit/s | 35 kBit/s |
| Speex | 2 bis 44 kBit/s | variabel |

*Tabelle 1: Gängige Audiocodecs für die Verwendung von VoIP*[8]
*(Bitrate Ethernet: inklusive aller benötigten Header-Daten)*

Demnach fallen bei guter Sprachqualität (G.711) über 1 Megabyte/Minute an Datenvolumen sowohl beim Download als auch beim Upload an. In bestimmten Situationen (Stau, generelle Netzüberanspruchung, Netzausfall) kann die einwandfreie Übertragung dieser Datenmenge aber nicht garantiert werden; Sprachaussetzer, Jitter (digitale Übertragungsschwankungen) und Delays (Verzögerungen) sind die Folge.

## 1.5 Vor- und Nachteile von VoIP

Vorteile:

- Einheitliche, kostengünstige Infrastruktur
- Mehrere Routen zum Empfänger möglich (Ausfallsicherheit)
- Verschiedenste Endgeräte verfügbar (z.B. PDA)
- Weltweiter Zugang/Erreichbarkeit
- Kostenlose Gespräche innerhalb des Internets (keine Gesprächsgebühren)

Nachteile:

- Schwankungen in der Sprachqualität
- Nur mit Breitband-Internet effizient nutzbar
- Zukünftig Probleme mit SPIT (Spam over Internet Telephony)
- Keine ortsmäßige Auswertung eintreffender Notrufe

[8] Entnommen aus [1]

# 2 VoIP für Extremsituationen

## 2.1 Aufbau eines Systems mit Spracherkennung und -ausgabe

Menschliche Sprache wird vom Recognizer (Spracherkennung) in der Applikation („VoIP4Text“) in Text umgewandelt und mit TCP oder UDP zum Gesprächspartner gesendet (Datenübertragung), wo die eintreffenden Textnachrichten dem Synthesizer (Sprachausgabe) übergeben werden.

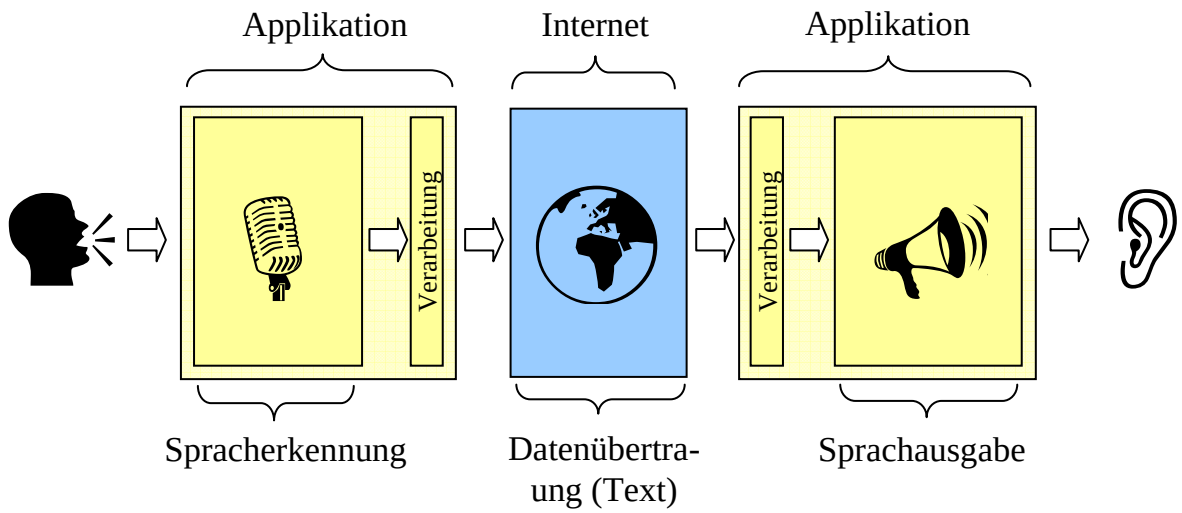


*Abbildung 1: Grundsätzliches Ablaufschema einer Sitzung mit Spracherkennung und Sprachausgabe*

In den folgenden Abschnitten werden die Komponenten dieses Systems (Spracherkennung, Sprachausgabe, Applikation) näher beschrieben.

## 2.2 Spracherkennung

### *Überblick*

IBM war der Pionier auf dem Gebiet der automatischen Spracherkennung durch Computer. Die Forschungen begannen in den 60er Jahren, mit zunehmendem Erfolg, bis 1984 ein System mit einem Wortschatz von 5000 Wörtern vorgestellt wurde. 1991 hatte sich das Vokabular bereits auf 20.000 bis 30.000 Wörter vergrößert, bevor 1994 der PC-Massenmarkt mit dem „IBM Personal Dictation System“ und später mit „IBM ViaVoice“[9] bedient wurde. Weitere Anbieter von Spracherkennungstechnologie und -produkten sind unter anderem Nuance mit „Dragon Naturally Speaking“[10] oder die Carnegie Mellon University mit „Sphinx“[11].

### *Grundlegende Funktionsweise*

Die ins Mikrofon eingehenden Audiosignale werden digitalisiert und in die kleinsten unterscheidbaren Sprachelemente, den sogenannten Phonemen, aufgesplittet. Menschliche Sprache umfasst eine Phonemenanzahl von 10 bis 80, die deutsche Sprache verwendet ca. 40. Weiters werden Vektoren definiert, die die akustischen Eigenschaften dieser Phoneme charakterisieren. Ein Akustikmodell führt dann Vergleiche dieser Vektoren mit bereits bekannten Vektoren (aufbauend auf Hidden Markov Modellen - HMM[12]) durch. Mit dieser Vorgangsweise kommt man auf eine gute Trefferrate. Ein Wörterbuch listet alle dem System bekannten Wörter und deren Zusammensetzung aus Phonemen auf. Zusätzlich zu einem optionalen Sprachmodell, das Grammatiken für die Spracheingabe definiert, sind damit diese sogenannten HMM-Spracherkennungssysteme mit der heutigen Rechenleistung moderner PCs in der Lage, kontinuierliches Sprechen zu erfassen und auszuwerten.

[9] http://www-306.ibm.com/software/voice/viavoice/
[10] http://www.nuance.com/naturallyspeaking/
[11] http://www.speech.cs.cmu.edu/
[12] http://de.wikipedia.org/wiki/Hidden_Markov_Model

## 2.3 Sprachsynthese

### *Überblick*

Die ersten ernstzunehmenden Versuche auf diesem Gebiet führten in den 30er Jahren die Bell Laboratories durch, die ein Gerät zur Sprachanalyse und Sprachsynthese, den sogenannten „Vocoder“, entwickelten. Der „Voder“ wurde Ende des Jahrzehnts bei der New Yorker Welt-Handelsmesse vorgestellt. Obwohl der mit einem Fußpedal und einer kleinen Tastatur gesteuerte Sprachsynthesizer noch sehr weit von qualitativ guter Sprachausgabe entfernt war, führte er doch zu einem gesteigerten Interesse an elektronischen Sprachausgabesystemen. Zur praktischen Anwendung kam eine abgewandelte Form, welche zur Kommunikationsverschlüsselung während des 2. Weltkriegs diente.

George Rosen vom Massachusetts Institute of Technology (MIT)[13] war Hauptentwickler des DAVO (Dynamic Analog of the VOcal tract) Systems. Später flossen die ersten Erfahrungen von Linear Predictive Coding (LPC) in die Entwicklung von Sprachausgabesystemen mit ein. 1968 entstand in den Electrotechnical Laboratories in Japan das erste vollständige Text-To-Speech (TTS) System. Nachdem Mitte der 70er Jahre die Verbindung OCR (Optical Character Recognition) mit anschließender Sprachausgabe (für blinde Menschen) erfolgreich getestet wurde, konnten in den 80er und 90er Jahren laufend neue und leistungsfähigere Engines entwickelt werden.

### *Grundlegende Funktionsweise*

Ein Sprachsynthese-System besteht aus zwei Teilen, dem Front-End und dem Back-End. Das Front-End führt zunächst eine Satz- und Wortanalyse durch, aus der dann Informationen über die Satz- bzw. Wortmelodie extrahiert werden. Zusätzlich werden Abkürzungen und Zahlen als vollständige Wörter behandelt. Danach wird aus dem bearbeiteten Text eine symbolische Sprachform ähnlich einer Lautschrift generiert, wobei noch eine weitere Aufteilung einzelner Wörter in unabhängige Phrasen (Text-To-Phoneme) stattfindet. Dieser Input wird vom Back-End verwendet, um dann die tatsächliche Spracherzeugung durchzuführen. Je nachdem welchen Umfang die bei der Synthese verwendete Datenbank hinsichtlich Informationen über verwendete Sprachsegmente hat, kann die Qualität der Sprachausgabe unterschiedlich ausfallen.

[13] http://web.mit.edu/

## 2.4 Applikationskomponenten

### *2.4.1 Java*

Als Programmiersprache wurde Java verwendet. Vor allem wegen der Lauffähigkeit auf unterschiedlichen Betriebssystemen und der Verfügbarkeit mehrerer Open-Source Komponenten (Recognizer, Synthesizer) war Java die erste Wahl.

### *2.4.2 JSAPI, JSML, JSGF*

Im Oktober 1998 wurde von Sun eine eigene Spezifikation für die Entwicklung von Spracherkennungs- und Sprachausgabesystemen veröffentlicht: das JSAPI[14] (Java Speech Application Interface). Dieses, unter anderem von IBM, Dragon Systems und Philips mitentwickelte Interface, besteht aus ca. 70 Klassen und ist in verschiedenen Software-Systemen realisiert. Zum JSAPI gesellen sich noch 2 weitere Spezifikationen hinzu, die Java Speech API Markup Language[15] (JSML) und das Java Speech API Grammar Format[16] (JSGF). Die XML-Anwendung JSML ermöglicht, die sprachliche Ausgabe normalen Textes durch spezielle Strukturierungen natürlicher und flüssiger zu gestalten. JSGF beschreibt Grammatiken (in Backus-Naur-Form), nach denen sich der Benutzer bei seiner Spracheingabe richten muss. JSAPI, JSML und JSGF sind in den verwendeten Komponenten in unterschiedlichem Umfang implementiert.

### *2.4.3 Spracherkennungssysteme*

Folgende Lösungen standen für die Realisierung von sprachgesteuerten Java-Anwendungen zur Verfügung:

- Sphinx-4[17]

  Hier handelt es sich um eine vollständig in Java geschriebene flexible Spracherkennungssoftware, die an der Carnegie Mellon University (CMU) gemeinsam mit anderen Technologien (Sphinx 1 - 3) entwickelt wird. Die JSAPI-Spezifikation wird nicht vollständig genutzt, es wird jedoch ein eigenes, auf niedrigerem Level angesiedeltes API verwendet. Sphinx-4 ist, wie alle übrigen Sphinx Produkte, vollständig Open-Source (derzeitige Version: 4.1.0 Beta).

[14] http://java.sun.com/products/java-media/speech/
[15] http://java.sun.com/products/java-media/speech/forDevelopers/JSML/
[16] http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/
[17] http://cmusphinx.sourceforge.net/sphinx4/

- IBM Java For Speech[18]

Diese Implementierung setzt auf der leistungsfähigen ViaVoice-Engine auf. Leider wurde das gesamte "Java For Speech"-Projekt mittlerweile eingestellt. Für ein ordnungsgemäßes Funktionieren wird neben einiger Java-Klassen außerdem noch eine ViaVoice-Programminstallation verlangt, eine „Out-Of-The-Box“-Benutzung der damit entwickelten Programme ist daher leider nicht möglich.

### *2.4.4 Sprachsynthesesysteme*

Bei der Sprachausgabe wurde die Auswahl auf folgende Produkte eingeschränkt:

- FreeTTS[19]

Vollständig in Java realisiert, baut es auf den Technologien „Festival“[20] (Sprachsynthese) und „FestVox“[21] (Tool für Stimmengenerierung) auf. Abgesehen von einigen wenigen Ausnahmen implementiert FreeTTS die JSAPI-Spezifikation in der Synthesizer-Sektion. Die Java-Klassen sind Open-Source und befinden sich aktuell in der Version 1.2.1.

- The Cloud Garden[22]

Hier handelt es sich um eine zu Microsofts SAPI 5 kompatible Synthesizer-Engine, die JSAPI vollständig nutzt, jedoch ist sie nicht kostenlos bzw. Open-Source.

### *2.4.5 Auswahl Recognizer*

#### 2.4.5.1 Sphinx-4

Die Vorteile von Sphinx-4 liegen einerseits in der Unabhängigkeit von proprietärer Software, andererseits in der Plattformunabhängigkeit. Das gesamte Paket besteht aus mehreren Jar-Dateien. Hinzu kommen noch unterschiedlich große obligatorische Akustik- und Sprachmodelle. Grundsätzlich ist die Architektur 3-geteilt:

*FrontEnd*

Die digitalisierten Audiodaten werden vom FrontEnd des Sphinx-Systems in Empfang genommen und in eine vom Decoder erkenn- und verarbeitbare Form gebracht (Feature).

[18] http://www.alphaworks.ibm.com/tech/speech
[19] http://freetts.sourceforge.net/
[20] http://www.speech.cs.cmu.edu/festival/
[21] http://festvox.org/
[22] http://www.cloudgarden.com/

*Linguist*

Der Linguist generiert für den Decoder einen Suchgraphen, der an das verwendete Sprach- bzw. Akustikmodell angepasst wird. Das Wörterbuch (Dictionary) enthält alle verwendbaren Wörter, inklusive deren Zusammensetzung aus Phonemen. Akustikmodell, Sprachmodell und Wörterbuch bilden zusammen die Linguist-Sektion.

*Decoder*

Der Decoderbereich besteht wiederum aus dem SearchManager, der mit Hilfe des Pruners die abgewandelten Audio-Frames zerlegt. Weiters versucht der Scorer, mit dem vom Linguisten erstellten Suchgraphen, die Erkennung zu starten. Das Resultat (evt. auch mehrere Resultate) werden dann an die aufrufende Applikation weitergegeben.

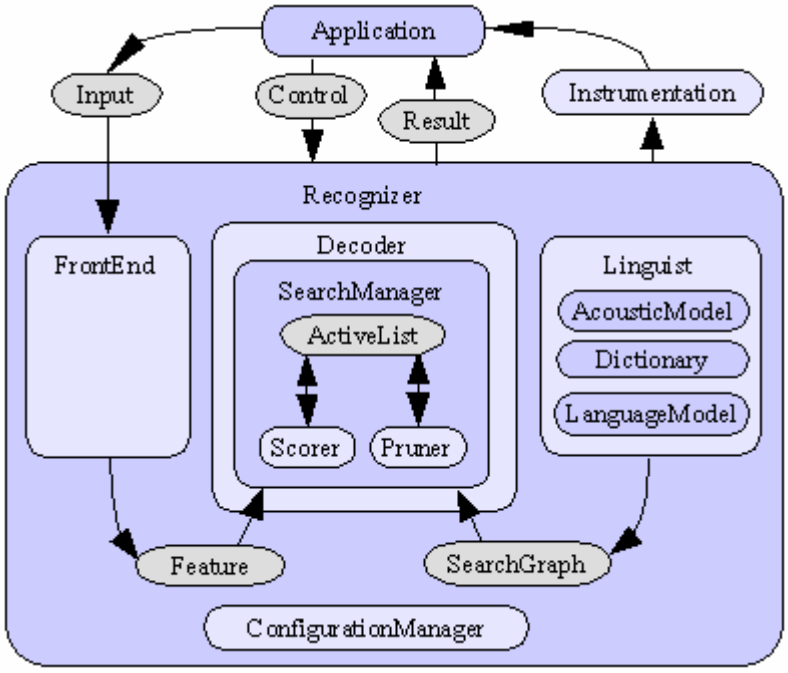


*Abbildung 2: Sphinx-4 Architektur*[23]

Alle Eigenschaften der beteiligten Sphinx-4 Komponenten werden in einer XML-Datei gespeichert. Diese wird vom ConfigurationManager eingelesen, der die Einstellungen verwaltet und gegebenenfalls auch während des laufenden Betriebs ändert. Ein typischer Ausschnitt einer solchen XML-Datei wird im folgenden abgebildet:

[23] Entnommen aus [2]

```
    <!-- ******************************************************** -->
    <!-- The Dictionary configuration                             -->
    <!-- ******************************************************** -->
    <component name="dictionary"
        type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
        <property name="dictionaryPath"
            value="resource:/edu.cmu.sphinx.model.acoustic.RM1_8gau_13dCep_16k_4
            0mel_130Hz_6800Hz.Model!/edu/cmu/sphinx/model/acoustic/RM1_8gau_13dC
            ep_16k_40mel_130Hz_6800Hz/dict/cmudict.0.6d+rm1.txt.fixed"/>
        <property name="fillerPath"
            value="resource:/edu.cmu.sphinx.model.acoustic.WSJ_8gau_13dCep_16k_4
            0mel_130Hz_6800Hz.Model!/edu/cmu/sphinx/model/acoustic/RM1_8gau_13dC
            ep_16k_40mel_130Hz_6800Hz/dict/fillerdict"/>
        <property name="addSilEndingPronunciation" value="false"/>
        <property name="wordReplacement" value="&lt;sil&gt;"/>
        <property name="unitManager" value="unitManager"/>

    </component>
```

Die Einbindung in die Entwicklungsumgebung Eclipse ging rasch vonstatten. Mit den enthaltenen Demo-Programmen ließen sich zunächst nur einzelne Fähigkeiten der Engine testen (Zahlen, einzelne Wörter), da die meisten dieser Programme nur auf einfachen Grammatiken aufbauen. Sie sind für Anwendungen geschaffen, in denen nur wenig gesprochen werden soll, aber eine hohe Trefferwahrscheinlichkeit und eine schnelle Verarbeitung erfordern. Beispiele von Grammatiken dieser Art sind das bereits genannte JSGF oder SimpleWordList (nimmt eine normale Wortliste als mögliche Spracheingabe her). Für den Zweck der zu entwickelnden Applikation sollte allerdings eine diktierartige Eingabe erfolgen, die zwar nicht immer sofort vom System erkannt werden muss, aber dennoch ein umfangreiches Vokabular zulässt.

Akustikmodelle enthalten unter anderem die Struktur aller Phoneme einer Sprache. Neben den bereits enthaltenen, standen auf der Sourceforge-Seite von Sphinx-4 weitere Modelle zum Download bereit, wobei folgende Pakete getestet wurden:

- WSJ (in Sphinx-4 enthalten)
- Hub4
- RM1

Leider ließen alle 3 Akustikmodelle keine Änderung der darin enthaltenen Daten zu. Zwar wurde Sphinx-4 wegen seiner nativen Java Herkunft bevorzugt, aber es fiel doch aufgrund der schlechteren Performance im Diktiermodus im Vergleich zum IBM System zurück.

#### 2.4.5.2 IBM Java For Speech

Da IBMs Ansatz auf einem kommerziellen Benutzerprodukt basiert, konnte auf Einbau externer Sprach- und Akustikmodelle vollends verzichtet werden.

Für die Einbindung in Java ist eine bestehende Installation von ViaVoice notwendig. Da aktuelle Versionen von ViaVoice möglicherweise keine Interaktion mit den Java-Klassen mehr zulassen, wurde eine ältere Version verwendet. In dieser Arbeit wurde auf die Millennium Edition aus dem Jahre 1999 zurückgegriffen, die problemlos auch in Windows XP installierbar und benutzbar ist. In einem speziellen Wizard kann mit einem ca. 15-minütigen Sprachtraining (je nach Intensität und Umfang des Trainings) die Erkennungsfähigkeit deutlich gesteigert werden. Weiters kann man Profile (und damit bereits absolvierte Sprachtrainings) speichern und laden. So können die eigenen Einstellungen leicht auf andere Arbeitsplätze übertragen werden.

Für die Integration in Eclipse genügte eine Einbindung des Archivs `ibmjs.jar`. Außerdem musste noch aufgrund diverser DLL-Dateien, die die eigentliche Verbindung zum ViaVoice System bildeten, die `PATH` Variable in Windows entsprechend angepasst werden. Um Spracherkennungs- bzw. Sprachsynthese-Engines im System zu registrieren, steht laut JSAPI-Spezifikation die Datei `speech.properties` im `lib` Verzeichnis des Java Runtime Environments zur Verfügung. Die Registrierung konnte man manuell oder mit Ausführung einer Stapelverarbeitungsdatei vornehmen.

### *2.4.6 Auswahl Synthesizer*

Die Auswahl fiel hier auf FreeTTS, da die gute Sprachausgabe und die problemlose und verhältnismäßig rasche Einbindung in das Projekt sehr von Vorteil waren.

### *2.4.7 JMF*

Für die Realisierung von Sprachübermittlung (eigentlichem VoIP) gibt es eigene Open-Source Lösungen, wie zum Beispiel Mobicents[24]. Da aber fast alle dieser Middleware-Systeme für dieses Projekt überdimensioniert waren, wurde auf die Multimediafähigkeiten des Java Media Frameworks (JMF)[25] zurückgegriffen. Mit diesem Package lässt sich Sprachübermittlung mit UDP und RTP mit unterschiedlichen Audiokompressionen effizient durchführen.

[24] https://mobicents.dev.java.net/
[25] http://java.sun.com/products/java-media/jmf/

# 3 VoIP4Text

## 3.1 Charakteristische Klassen

Um die Funktionsweise relevanter Programmteile näher zu erläutern, werden im folgenden charakteristische Klassen näher betrachtet. Um den unterschiedlichen Funktionalitäten der Klassen Rechnung zu tragen, wurden 3 Java-Packages erstellt. Das erste beinhaltet alle GUI-Klassen (`gui`), die nur der Informations-Repräsentation bzw. der Interaktivität mit dem Benutzer dienen. Das `core`-Package enthält Klassen für den eigentlichen Programmablauf und ist mit dem dritten Package lose verbunden, das die eigentlichen Spracherkennungs, -synthese und -übermittlungsklassen beinhaltet (`engine`).

Alle Funktionen werden innerhalb von Thread-Klassen ausgeführt, da während der (oft blockierenden) Ausführung dieser Tätigkeiten der Programmfluss nicht unterbrochen oder aufgehalten werden darf. Alle im `engine`-Package enthaltenen Klassen implementieren daher einheitlich das `Runnable`-Interface; für die Interaktion mit der Programmlogik realisieren sie außerdem das `Observer`-Pattern.

### *3.1.1 RecognizerThread.java (engine)*

Die Hauptaufgabe dieser Klasse besteht darin, nach Initialisierung der Engine, Erkennungsprozesse zu behandeln und die beobachtenden Klassen mit Resultaten zu versorgen. Außerdem bietet sie auch Möglichkeiten an, Änderungen an der (laufenden) Engine zuzulassen. Da diese JSAPI Funktionen von IBM nicht berücksichtigt wurden, bleibt ihr Aufruf aber ohne Wirkung.

Wie auch der `SynthesizerThread` benutzt der `RecognizerThread` Klassen aus `javax.speech.*`, speziell aus `javax.speech.recognition`. Das wichtigste Objekt im `RecognizerThread` ist vom Typ `Recognizer`, das die notwendigen Spracherkennungsfunktionen anbietet. Seine Eigenschaften werden in einem Objekt der Klasse `RecognizerProperties` abgelegt.

Mit Hilfe der statischen Methode `availableRecognizers(...)` werden je nach übergebenen Parametern die zur Verfügung stehenden `Recognizer` in einer dafür vorgesehenen `EngineList` gespeichert. Durch spezielle Übergabeparameter könnte zum Beispiel eine länderspezifische Auswahl erfolgen; mit dem leeren Übergabeparameter `null` wird keine Einschränkung vorgenommen. Die `Recognizer`-Instanz wird mit dem ersten (und in diesem Fall einzigen) Element einer `EngineList` initialisiert und seine aktuellen Parameter gespeichert.

```
EngineList engList = Central.availableRecognizers(null);
recRecognizer = Central.createRecognizer((EngineModeDesc) engList.get(0));
recProperties = recRecognizer.getRecognizerProperties();
```

Die `allocate()`-Methode stellt dem `Recognizer` alle benötigten Ressourcen zur Verfügung. Dieser Aufruf dauerte bei dem in der Entwicklung eingesetzten System 1 bis 3 Sekunden.

```
recRecognizer.allocate();
recRecognizer.waitEngineState(Engine.ALLOCATED);
```

Ein `RecognizerAudioListener` erhält Events bei Änderungen der Lautstärke des Mikrofonkanals. Diese Änderungen werden in der GUI mittels einer Pegelanzeige grafisch angezeigt. Daneben wird noch ein `EngineListener` (für spezielle Engine-Ereignisse) und ein `DictationGrammar`-Objekt angelegt. Der `ResultListener` erhält und verarbeitet die erkannten Wörter.

```
recRecognizer.getAudioManager().addAudioListener(mainAudioListener);
recRecognizer.addEngineListener(mainEngineListener);
grmDictation = recRecognizer.getDictationGrammar(null);
grmDictation.addResultListener(mainResultListener);
```

Die `resultAccepted()`-Methode erhält vom `Recognizer` ein `ResultEvent`, das alle Wörter mit der höchsten Trefferquote beinhaltet. Nachdem daraus ein Satz gebildet worden ist, wird die Klasse `MainControl` über diese erkannte Wortfolge benachrichtigt, die dann dort weiter bearbeitet wird.

```
public void resultAccepted(ResultEvent resEvent){
      Result result = (Result) resEvent.getSource();
      String strResult = "";
      int i = 0;

      for (int j=0; j<i; j++)
            strResult += result.getBestToken(j).getSpokenText() + " ";
      ...
}
```

Eine Anpassung der grafischen Pegelanzeige für die Mikrofonlautstärke wird mit folgendem Methodenaufruf signalisiert (`getAudioLevel()` liefert Float-Zahlen von 0.0 bis 1.0).

```
mainControl.updateVolumeLevel((int) (audioEvent.getAudioLevel() * 10));
```

### *3.1.2 SynthesizerThread.java (engine)*

Alle Sprachausgabefunktionen werden in der Klasse `SynthesizerThread` behandelt, wobei speziell die Klassen aus `javax.speech.synthesis` benutzt werden.

```
SynthesizerModeDesc synModeDesc = new SynthesizerModeDesc(null, "general",
                                        Locale.US, null, null);
```

```
synSynthesizer = Central.createSynthesizer(synModeDesc);
```

Wie bei der Klasse `Recognizer` kann mit `createSynthesizer(...)` eine neue `Synthesizer`-Instanz geschaffen werden. Vorher werden ihre Eigenschaften näher spezifiziert.

```
synModeDesc = (SynthesizerModeDesc) synSynthesizer.getEngineModeDesc();
Voice[] vocAllVoices = synModeDesc.getVoices();
```

Jeder `Synthesizer` enthält eine bestimmte Anzahl von Stimmen. Manche sind nur für eine eingeschränkte Benutzung gedacht (z.B. Ansage der Uhrzeit), erzeugen dadurch aber eine sehr akkurate Ausgabe. Andere sind für alle Arten von Text bestimmt und haben durch diese generische Bestimmung eine etwas schlechtere Ausgabequalität. Die Hauptstimme für normalen Text nennt sich ‚`kevin16`' und wird als aktive Stimme in den Eigenschaften des `Synthesizers` gespeichert:

```
synSynthesizer.getSynthesizerProperties().setVoice(vocCurrent);
```

### *3.1.3 AudioReceive.java (engine)*

Der Erstellung einer `RTPManager`-Instanz, die sich um die Verwaltung einer RTP Session kümmert, folgt eine Registrierung bestimmter Listener. Der `SessionListener` behandelt neu hinzukommende Teilnehmer der Session; der `ReceiveStreamListener` verwaltet deren eintreffende `ReceiveStreamEvents`:

```
rtpManager = RTPManager.newInstance();
rtpManager.addSessionListener(this);
rtpManager.addReceiveStreamListener(this);
```

Mit den Anweisungen

```
rtpManager.initialize(new SessionAddress(InetAddress.getLocalHost(),
intLocalPort));
rtpManager.addTarget(new SessionAddress(adrPartner, intRemotePort));
```

wird der RTP Sitzung Sender und Empfänger hinzugefügt. Danach wartet der Thread in der `update-`Funktion des `ReceiverStreamListeners` in einer Schleife eine bestimmte Zeitdauer (benutzerdefiniert) auf ein `NewReceiveStreamEvent`.

```
stmReceive = ((NewReceiveStreamEvent) evt).getReceiveStream();
DataSource datSource = stmReceive.getDataSource();
```

```
...

Player plyMainPlayer = javax.media.Manager.createPlayer(datSource);
plyMainPlayer.realize();
```

Mit diesem Event lässt sich eine Instanz von `DataSource` (Datenquelle) kreieren, die wiederum als Argument eines `Player`-Objektes dient, welches für die tatsächliche Ausgabe der Audiodaten verantwortlich ist.

Der `ReceiveStreamListener` horcht ebenso auf ein ankommendes `ByeEvent`. Dieses signalisiert die Beendigung des Streams seitens des Gesprächspartners.

### *3.1.4 AudioTransmit.java (engine)*

Der zur Laufzeit des Programms auswählbare `MediaLocator` (`DirectSoundCapture` oder `JavaSound audio capture)` dient als Argument für die Erstellung der Datenquelle der Übermittlung `(DataSource).`

```
DataSource datMainSource = Manager.createDataSource(locAudio);
```

Ankommende Streams werden von einem `Processor`-Objekt verwaltet, außerdem muss der Typ der gesendeten Daten explizit gesetzt werden `(RAW_RTP)`:

```
Processor prcProcessor = javax.media.Manager.createProcessor(datMainSource);
prcProcessor.setContentDescriptor(new
ContentDescriptor(ContentDescriptor.RAW_RTP));
```

Eine `TrackControl`-Instanz kümmert sich um den Stream (im Programm wird nur 1 Audiostream behandelt), aus dieser werden die unterstützten Audiocodecs entnommen, von denen einer durch Benutzerauswahl selektiert wird.

```
TrackControl trcControl = prcProcessor.getTrackControls()[0];
Format forAvailableFormats [] = trcControl.getSupportedFormats();
...
trcControl.setFormat(forAvailableFormats[forDialog.getSelectedIndex()]);
```

Gleich wie bei der Klasse `AudioReceive` verwaltet ein `RTPManager` die RTP Sitzung, wo wiederum der lokale Computer und die IP-Adresse des Gesprächspartners registriert werden.

```
rtpMgrs[i] = RTPManager.newInstance();
```

```
rtpMgrs[i].initialize(new SessionAddress(InetAddress.getLocalHost(),
intLocalPort));
rtpMgrs[i].addTarget(new SessionAddress(adrPartner, intRemotePort));
```

Schließlich wird die Audioübermittlung mit Ausführung der `start`-Methode eines vom `RTPManager` kreierten `SendStream`-Objektes initiiert.

```
 (rtpMgrs[i].createSendStream(datOutput, i)).start();
```

### *3.1.5 UDPCallListener.java & TCPCallListener.java (core)*

Die Thread-Klassen `UDPCallListener` und `TCPCallListener` horchen in einer Endlosschleife an bestimmten Ports auf eintreffende TCP und UDP Nachrichten. Der `UDPCallListener` wartet in der `receive`-Funktion so lange, bis eine Nachricht eintrifft, und gibt sie mittels der `receiveUDPMessage`-Funktion an die `MainControl`-Instanz weiter, die die Nachricht weiter bearbeitet.

```
DatagramPacket pakUDP = new DatagramPacket(...)
socUDP.receive(pakUDP);
mainControl.receiveUDPMessage(pakUDP.getAddress(), pakUDP.getPort(),
      (new String(pakUDP.getData())).substring(0, pakUDP.getLength()));
```

Eine ähnliche Prozedur beinhaltet der `TCPCallListener`, der zusammen mit der Willkommensnachricht auch noch den `Socket` an `MainControl` weitergibt.

```
Socket socClient = ssoClient.accept();
mainControl.receiveTCPMessage(socClient, (new BufferedReader(new
InputStreamReader(socClient.getInputStream()))).readLine());
```

### *3.1.6 Call.java (core)*

Die Klasse `Call` repräsentiert den eigentlichen Anruf. Der Zustand eines Calls ist eindeutig bestimmt:

`Initialized` bedeutet, dass der Anruf von einer Seite getätigt, auf der anderen Seite jedoch noch nicht bestätigt/abgelehnt wurde (d.h. er befindet sich momentan im `Received` Zustand). Der aktive Anruf (höchstens einer) bekommt den `Running` Status, ein wartender Anruf (vorübergehend deaktiviert) den `Waiting` Status zugewiesen. Sobald ein Gespräch beendet wurde, ist es als `Cancelled` identifizierbar (bestimmte Funktionen wie zum Beispiel „Save Call“ sind noch verfügbar), bis man es komplett aus der Anrufsliste löscht.

```
public static int INITIALIZED = 0;
```

```
public static int RECEIVED = 1;
public static int RUNNING = 2;
public static int CANCELLED = 3;
public static int WAITING = 4;
```

Alle Nachrichten werden chronologisch jeweils als `LogMsgCon`-Instanz in einem Vektor gespeichert. Mit einem Thread innerhalb der Klasse `Call` wird der `Socket` auf ankommende TCP Messages abgehorcht. Beim UDP-Modus übernimmt diesen Part die Klasse `UDPCallListener`, da `TCPCallListener` sich nur auf neu ankommende Anrufe beschränkt und vorhandene aktive Sockets nicht behandelt.

```
private Vector <LogMsgCon> vecMessages;
private InMessagesThread thrInMessages;
```

## 3.2 Struktureller Klassenaufbau

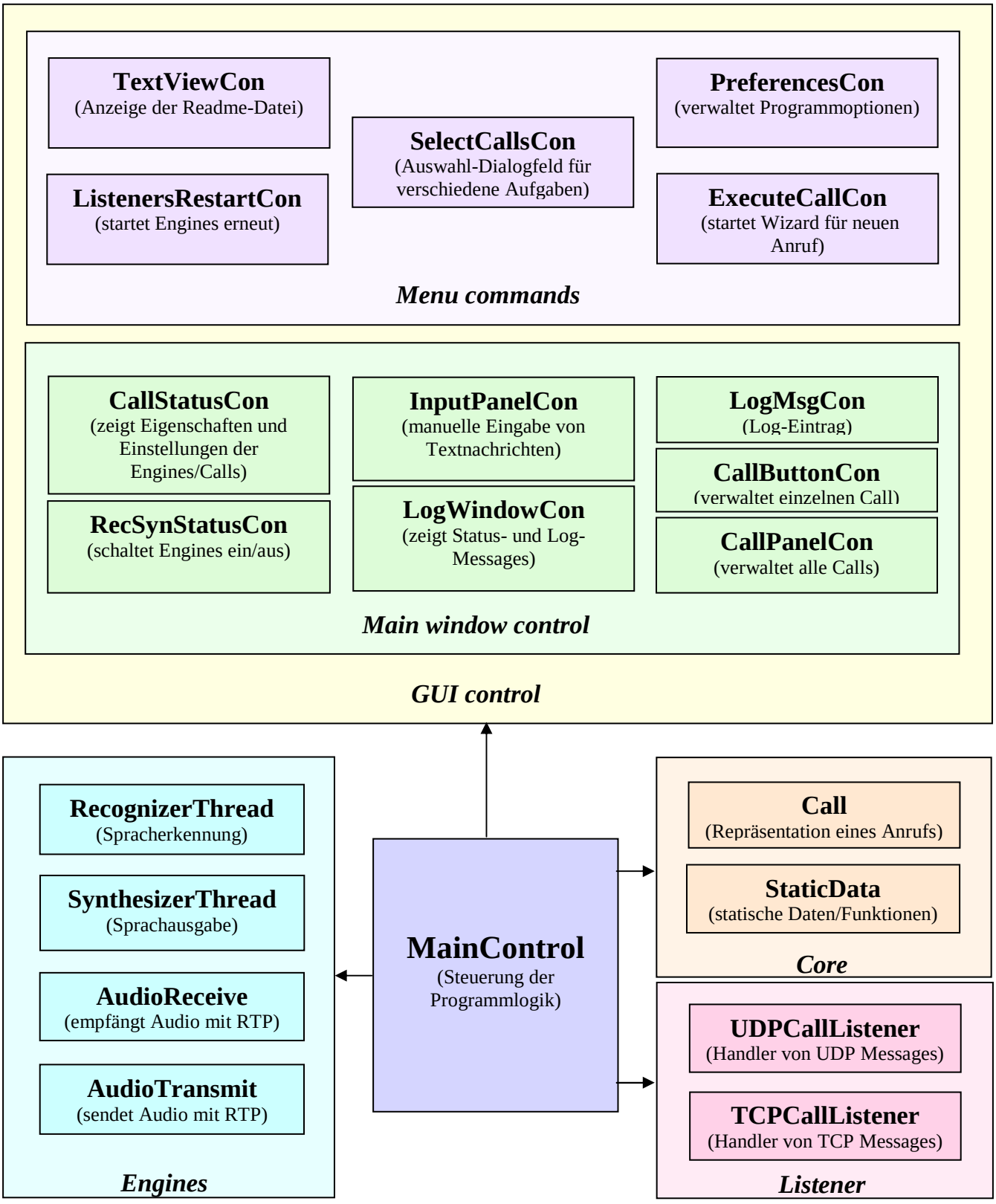


*Abbildung 3: Strukturelles Klassendiagramm von „VoIP4Text“*

## 3.3 Screenshots


*Abbildung 4: Hauptfenster von „VoIP4Text“ mit aktivem Anruf (TCP) und aktiviertem Recognizer und Synthesizer*

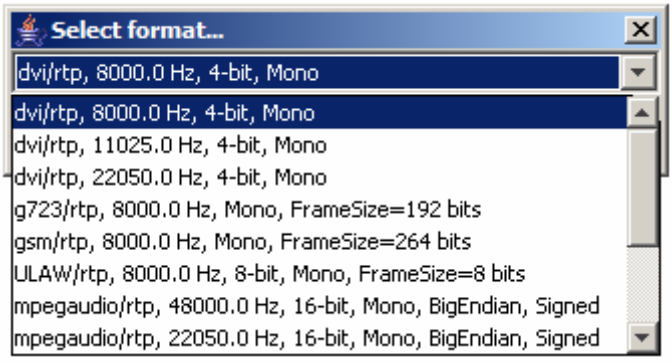


*Abbildung 5: Auswahldialog der Audiokompression*

# 4 Test

Nachdem erste Tests mit Sphinx-4 als Spracherkennungs-Engine eher enttäuschend verlaufen waren, wurde dieser Teil des Systems komplett auf die IBM-Lösung umgestellt. Anschließend konnten auch andere neue Funktionen (Sprachübermittlung mit RTP) getestet werden.

Für den Test wurde nur eine einfache Netzwerkumgebung, ohne künstlich erzeugten Stau oder Bandbreitenbeschränkungen verwendet.

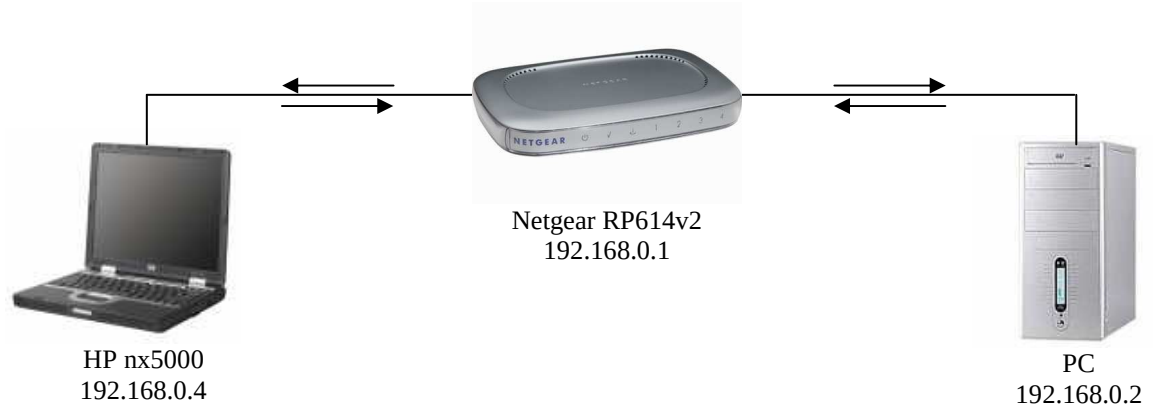


*Abbildung 6: Aufbau der einfachen Teststrecke*

Folgende Software- und Hardwarekomponenten wurden beim Test verwendet:

*Software (auf PC und Laptop):*

- Microsoft Windows XP Professional SP2
- JRE 1.5.0_08
- JMF 2.1.1e
- FreeTTS 1.2.1
- IBM ViaVoice Millennium

*Hardware:*

| **PC:** | **Laptop (HP nx5000):** | **Router (Netgear RP614v2):** |
|---|---|---|
| AMD Athlon 1600+<br>768 MB RAM<br>NVidia Geforce 3 Ti 200<br>Creative Audigy<br>Philips Mikrofon<br>(SBC MD100) | Intel Pentium M 1500<br>512 MB RAM<br>Intel 82855 GM<br>SoundMax Audio<br>Philips Mikrofon<br>(SBC MD150) | Firmware: 5.20_RC3 |

Da die Performance von ViaVoice sehr von einem absolvierten Sprachtraining abhängt, konnten bei der Durchführung der Tests unterschiedliche Erkennungsraten bei unterschiedlichen Sprechern beobachtet werden. Für den Betrieb der Software war eine leise Umgebung erforderlich, da die Spracherkennung auch auf leise Gespräche im Hintergrund bzw. die Sprachausgabe des Gegenübers reagierte. Nach Möglichkeit sollte daher ein Headset verwendet werden. Des Weiteren wurden längere Sätze erst bei einsetzender Stille am Satzende verarbeitet und übertragen, was durch (leider unveränderbare) Einstellungen im Spracherkennungssystem bedingt ist. Nur bei kürzeren Sätzen ist ein unmittelbarer telefonähnlicher Austausch möglich.

Sowohl die UDP- als auch die TCP-Übertragung funktionierten einwandfrei. Auswirkungen des Nagle-Algorithmus[26] konnten wegen fehlender Protokollierung des Datenverkehrs nicht beobachtet werden. Auch bei der Sprachübermittlung mit RTP konnten keine Probleme festgestellt werden.

[26] http://tools.ietf.org/html/rfc896

# 5 Zusammenfassung

## 5.1 Allgemeines

Die folgende Software wurde bei dieser Arbeit verwendet:

- Eclipse 3.2.0
- Java SE Development Kit 1.5.0
- Microsoft Word 2003 (Dokumentation der Bakkalaureatsarbeit)
- PDFCreator 0.9.0 (PDF Erstellung)

Folgende Ordnerstruktur befindet sich auf der Programm-CD:

- `source` (Quellcode)
- `bin` (ausführbare Programmdateien)
- `doc` (Dokumentation der Arbeit im Doc- und Pdf-Format sowie Javadoc-Projektdokumentation)

## 5.2 Fazit

Das Ziel dieser Arbeit war es, eine (zugegebenermaßen) experimentelle Anwendung zu schreiben, die Sprache erkennt, in Text umwandelt, diesen Text in einem Netzwerk einer anderen Instanz der Anwendung zusendet und dort wieder als gesprochene Nachricht über Lautsprecher ausgibt. Außerdem sollte man zwischen diesem und dem Modus herkömmlicher Sprachübermittlung während eines Gespräches auswählen können. Als verwendetes Sprachausgabesystem setzte sich ziemlich bald FreeTTS durch, bei der Auswahl des Spracherkennungssystems wurde zunächst mit Sphinx-4 experimentiert. Da diese Engine nicht die erwartete Leistung bot, wurde auf die IBM „Java For Speech“ Technologie gewechselt, die vor allem im Diktiermodus gute Resultate erzielte. Die traditionelle VoIP-Funktion wurde mit Hilfe des Java Media Frameworks bewerkstelligt. Im Test schwankte die Qualität der Spracherkennung je nach Sprecher. Durch Sprachtraining (Lesen von ca. 100 Sätzen) kann die Erkennungsgenauigkeit aber deutlich verbessert werden.

# 6 Anhang: Readme

```
============================================================
Program: VoIP4Text 1.0
Author: Elmar Weiskopf <csae1328@uibk.ac.at>
Date: 24.10.2006
License info: free to use
Supervised by: Michael Welzl
============================================================

Readme

1. Introduction to VoIP4Text
2. Hardware and software requirements
3. Download instructions
4. Installation instructions
    4.1 ViaVoice installation
    4.2 VoIP4Text prerequisites
    4.3 Engine registration
    4.4 Uninstallation
5. Using VoIP4Text
    5.1 GUI description
    5.2 Menu bar
    5.3 Engine buttons
    5.4 Status panel
    5.5 Input text field
    5.6 Log window
    5.7 Call area
        5.7.1 Calls
        5.7.2 Call management
        5.7.2 Call panel
6. Tutorial


1. Introduction to VoIP4Text
------------------------------------------------------------

VoIP4Text is an application which translates human speech
into normal text, transmits this text to another participant
and transforms it again to human language with voice
synthesizing. Also it provides a routine for ordinary VoIP
operation and a chat function too.

As many VoIP applications are available with growing
popularity today, one may ask why developing a program with
characteristics mentioned above. Consider situations with
limited or hardly any bandwidth available, such as disasters
or blackouts. The infrastructure of many information systems
will collapse due to heavy use of telephone and computer
networks. Although more and more modern audio compression
```

algorithms are available today, there is nothing more efficient in terms of low data throughput than transmitting pure text. One can transfer a couple of sentences with just one kilobyte. There is of course a significant drop in speech quality on the receiver's side and some spoken words cannot be recognized for the very first time on the other side too, but with careful and systematic use of recognition facilities, it could be a useful tool somewhere, sometime.

VoIP4Text uses FreeTTS as synthesizing and IBM Java-For-Speech as recognizing engine. FreeTTS libraries are already included in VoIP4Text but also can be downloaded freely from http://sourceforge.net/projects/freetts as they are open-source. IBM Java-For-Speech provides powerful recognizing capabilities for the application. Unfortunately, the Java-For-Speech project has been shut down, so the IBM Java classes aren't available for download anymore, but have been included in VoIP4Text.

## 2. Hardware and software requirements

Hardware:

* Standard PC
* Soundcard (for speech input/output)
* Microphone (analog, USB)

Software:

* Windows (XP, 2000, 98, NT)
* Java Runtime Environment 1.5.0
  www.java.com
* Java Media Framework 2.2.1e
  http://java.sun.com/products/java-media/jmf/
* IBM ViaVoice (possibly older versions)
  http://www.nuance.de/viavoice

## 3. Download instructions

Please download VoIP4Text from www.welzl.at. This location provides additional information on this project too. Uncompress the downloaded zip file to a destination of your choice.

## 4. Installation instructions

### 4.1 ViaVoice installation

As said before, ViaVoice has to be installed, just follow the

installation wizard. In the final step, the wizard will ask you to train your voice by speaking a number of sentences, so that the recognition engine can adapt to your voice and speaking style. The ViaVoice options offer a profile saving feature, which creates an approximately 2 megabyte zip file. So, if you have a saved profile and use ViaVoice on another computer, you just have to import your profile. Finally you should be in the position to use the ViaVoice recognition engine.

## 4.2 VoIP4Text prerequisites

Please note that the latest Java Runtime Environment (JRE) is required, otherwise the application won't start. Furthermore, Java Media Framework (JMF) is required to enable transmission of speech via RTP. Download JRE and JMF and install them. Both install routines should alter CLASSPATH and PATH settings appropriately.

## 4.3 Engine registration

To enable JSAPI with FreeTTS enter the ‘lib/freetts’ directory of the VoIP4Text folder and execute ‘jsapi.exe’. Click on ‘I Agree’ and press ‘Close’ to complete this part.

Set the Windows environment variables by double-clicking the ‘System’ icon in your control panel. Choose the ‘Advanced’ tab and then click on the button ‘Environment variables’.

In the ‘System variables’ section create or edit the variable ‘PATH’ and append the following string:

%VoIP4Text%\lib\ibmjs;

where %VoIP4Text% is the VoIP4Text installation folder. For example if you installed VoIP4Text in ‘C:\Program Files\VoIP4Text’, the PATH entry would be

C:\Program Files\VoIP4Text\lib\ibmjs;

After you did these steps restart your system to apply the changes. Finally, enter the ‘lib’ directory of the installed JRE and open the ‘speech.properties’ file with a text editor. Append the following 2 lines (perhaps to already existing text) to register both engines.

```
FreeTTSSynthEngineCentral=com.sun.speech.freetts.jsapi.FreeTT
SEngineCentral
com.ibm.speech.recognition.EngineCentral=com.ibm.speech.recog
nition.IBMEngineCentral
```

## 4.4 Uninstallation

If you don't want the application on your PC anymore, remove the VoIP4Text folder. Additionally remove the PATH entries and the 2 added lines in the ‘speech.properties’ file. This doesn’t uninstall the IBM ViaVoice application, you also have to remove it for a totally cleaned system.

# 5. Using VoIP4Text

---

## 5.1 GUI Description

Start VoIP4Text by double-clicking VoIP4Text.jar or by typing

java –jar VoIP4Text.jar

in a console window. This brings up the main window of VoIP4Text, which consists of the following parts:

* Menu bar
* Buttons to switch engines/speech transmission on/off
* Status panel with information on the current call/engines
* Text field to send text messages
* Log window to display call and system messages
* Call area with all existing calls

## 5.2 Menu bar

In the following section all menu items are described:

File

Restart Listeners (Ctrl-G):
UDP- and TCP-Listeners can be restarted by selecting the items, press ‘OK’ to confirm.

Exit VoIP4Text (Alt-F4):
Exits the application and ends all running threads (all calls are terminated on both sides)

Call

Start (Ctrl-T):
Starts a new wizard for starting a new call. You can click ‘Next’ or ‘Back’ to get to the next/previous page. Enter the IP-address and select the transmission mode (UDP or TCP) on the first screen. You can also enable Nagle’s algorithm in addition to TCP mode, which has positive effects on the efficiency of data transmission. On the next wizard page enter a unique ID for the call. A call is identified with this ID externally and internally, mostly the ID will consist of the call partner’s name. ‘Output messages to

synthesizer' and 'Display log messages' enable speech output and log window output, respectively. Both items are enabled by default, but you can switch them off later in the popup menu of the call button. Also a representing call color can be selected. If not, a default color will be used. At the last page you can enter a message, which is displayed on your partner's side when he clicks on the incoming call. Press 'Finish' to quit the wizard and start a call with the entered properties.

Hang Up Call(s) (Ctrl-U):
Shows a list where you can select between suitable calls to hang up. By clicking 'Select All' all calls are checked. Finally press 'OK' to hang up selected calls. Hung up calls are still shown in the call panel, so their log messages can still be saved.

Remove Call(s) (Ctrl-E):
Shows a list where you can select hung up calls to remove them permanently. Press 'OK' to confirm your choice.

Switch Call (Ctrl-W):
Shows a list of inactive calls, where you can select one to switch to. The currently running call will be forced to wait.

Save Call(s) (Ctrl-S):
Shows a list of all existing calls. You can save the log messages of selected calls in a text file (one file per call) with date, time, speaker and spoken message. Therefore, a file chooser window will appear for each selected call.

Options
Preferences (Ctrl-P):
The following ports can be changed within this dialog window:
* UDP/TCP port
* RTP local port receiver/remote port sender

IMPORTANT: The application cannot receive and send data from only one port, so you have to adjust the RTP local/remote port settings to your call partner. Your call partner's local receiver port should be equal to your remote sender port and vice versa. Otherwise RTP audio transmission will not work.

You can also switch on/off the possibility to use the recognizer without a running call by selecting the according checkbox. The 'RTP timeout' value determines

the time (in seconds) the application will wait for incoming RTP streams when you start RTP speech transmission. Enter the name you wish to be recognized in the log window messages in the appropriate box (‘User name’). Press ‘Apply’ or ‘OK’ to save settings.

Help

Help (Ctrl-H):

Shows this readme file.

About (Ctrl-B):

Shows some information about VoIP4Text.

## 5.3 Engine buttons

There are 3 big buttons with icons in the upper left area of the main window. Every button switches an engine on or off. The current status of an engine can be seen by the actual state of the button: if an engine is running, the button appears pushed and the icon on the button is colored; when deactivating the look of the button will change to its initial look. Also messages in the log window will inform you about the current engine states. The engines can be turned on/off independently (except for some combinations, e.g. RTP speech transmission and a recognizer cannot run at the same time). Therefore, you can also use the program if your PC isn’t equipped with a soundcard or a microphone, as you can type in text messages with your keyboard.

The first click on the synthesizer button will start the engine, following clicks will only pause/resume it. This is much faster as starting the engine completely new. This also applies to the recognizing engine. Additionally, a graphical view of the microphone volume is shown below the buttons on activation of the recognizer. This can possibly be helpful if you have problems with the microphone. These 2 engines can be enabled without starting a new call. The RTP speech transmission can only be activated if there exists a running call.

When clicking on the ‘Speech Transmission’ button you will have the choice between different audio formats to use for this session, each with a different data rate. You can adapt your choice to your existing bandwidth, the call partners can also use 2 different audio formats (their formats are not bound together). Press ‘OK’ to confirm your choice.

## 5.4 Status panel

The status panel includes information on the current running

call (ID, IP-address, port, etc.). Furthermore, several adjustments can be done for all relevant engines (synthesizer, recognizer and speech transmission). Synthesizer and recognizer properties can only be changed if the respective engine is running. However, the ‘Speech Transmission’ tab can only be accessed before starting that mode. The tabs ‘Synthesizer’, ‘Recognizer’ and ‘Speech Transmission’ will be enabled/disabled according to the current states of the engines, whereas the tab ‘General’ is always accessible. For all synthesizer and recognizer adjustments there is a default button too, which sets the according value to its default. You can adjust the values by dragging the sliders.

Synthesizer properties:

Volume: adjusts speech output volume, ranges from 0 (muted) to 10 (maximum); default is 10

Words/min: adjusts the speed words and sentences are spoken, ranges from 0 to 400; default is 200

Pitch: adjusts the pitch of the voice, ranges from 50 (very low) to 200 (very high); default is 100

Range of voice: adjusts the pitch range of the voice, ranges from 0 (very monotonous) to 50 (excessively lively); default is 10

Recognizer properties:

The following properties aren’t implemented (yet) in the IBM engine and dragging the sliders doesn’t have any effects.

Complete timeout: determines the time of silence (in seconds) before the engine stops the current recognition process and publishes the result; ranges from 0 to 10 seconds; default is 1

Sensitivity: a higher value makes the recognizer more sensitive to (background) noise, a lower value requires a louder speaking voice, but most of the background noise will be ignored; ranges from 0 to 10; default is 5

Speed/Accuracy: sets the ratio between fast and accurate recognition; decreasing the value minimizes recognition time, increasing leads to an improved, but very slow recognition process; ranges from 0 to 10; default is 5

Speech Transmission properties:

Audio capture engine: In Windows you can select between the DirectSound capture and the JavaSound capture engine. Both use the same audio compressions for transmission.

## 5.5 Input text field

The long input text field provides a way to send text directly to the partner of the current running call, which gives the program chat capabilities. Type the text into the text field and press ‘Go’ to send it.

## 5.6 Log window

The log window captures all incoming and outcoming messages of all calls (except of the manually excluded ones). Also system messages (synthesizer on/off,...) are shown. All call messages have the same format: Date->Time->Speaker->Message. Below the log window you have 3 buttons: ‘Save active call’ saves the current running call to a text file and ‘Save log window’ saves all messages to a text file.

## 5.7 Call area

The call panel is the area on the bottom of the main window where all existing calls are shown.

### 5.7.1 Calls

A call is displayed as a button with a telephone icon in a certain color. Next to the telephone there is the unique ID of the call. You have the possibility of managing various calls. A call is always in some state:

RUNNING: When a call is in the RUNNING state, it is active and all inputs (speech and text) are sent to the IP-address of the call’s opponent. Only one call can be active at a time. If you set another call into the RUNNING state, the previous active call will be transferred into the WAITING state. A running call has a green telephone icon within the button.

WAITING: When a Call is in the WAITING state, it is temporarily suspended (no input will be sent to the counterpart), but incoming messages will be received continuously. There is no limitation for waiting calls. Calls in this state are visualized by a blue telephone icon.

INITIALIZED: When you start a call and your opponent hasn’t responded to it yet, it is in the INITIALIZED state. You cannot send text or speech to the call partner, until he accepts your call. INITIALIZED calls have a light yellow telephone icon.

RECEIVED: An incoming call is in the RECEIVED state, until you accept that call (set it to RUNNING or WAITING state). If a new call has arrived you will hear a telephone ring and the call button will flash. Also received calls are indicated by a light yellow telephone icon.

CANCELLED: A cancelled call is a hung up call, so it can’t take input anymore. However, as long as they aren’t removed permanently, you are still able to save the their log messages. These calls will have a gray telephone icon.

### 5.7.2 Call management

As said before, only one call can be in the RUNNING state at a certain time, all other states don’t have such restrictions. If there is no active call, some options are grayed out (Speech transmission button, text input) and a running recognizer puts the recognized words only in the log window. If your opponent switches from your call to another call, you won’t notice it. You can still send messages, your counterpart only receives them in the log window. If a participant quits the call (or closes the application) the call will be hung up immediately on both sides.

### 5.7.3 Call panel

Call buttons are grouped in the call panel and represent calls. You can click on such a button, regardless of the current state of the call. If you click on a call button which representing call is in the RECEIVED state, a dialog window will pop up, showing you a welcome message, a text field to enter the ID for this call and some buttons to change the state (activate, set waiting, hang up) and the main color of the call.

On all other call states, a popup menu will be shown, with some call information and several menu items. Active menu items depend on the current state of the call. You will have the possibility to activate a call, set it to the WAITING state, hang up and remove the call permanently. What is more, you can save the call, change the color and decide, whether log messages of this call are shown and speech output from the synthesizer is enabled.

## 6. Tutorial

---

To make it easier for you to use VoIP4Text, the following little tutorial has been written.

After downloading and installing all required components according to paragraphs 3 and 4 of this readme file, start

VoIP4Text and the main window appears. Adapt the RTP ports to your call partner, according to the ‘Preferences’ section in paragraph 5.2. Attach your microphone to an empty USB-port or to your soundcard. In the windows mixer window select the microphone line to capture audio.

Starting a new call:

Select from the menu ‘Call’ the item ‘Start’ or press Ctrl-T and a little wizard appears. Enter the desired IP-address of your call partner in the first box and select a transmission mode from the next combobox. UDP data transfer is the default mode for a call, so just leave it. Press ‘Next’ to get to the next page, where you have to enter a unique ID for the call. Leave the checkboxes selected and choose a call color by clicking the ‘Color’ button. After you hit ‘Next’ enter a welcome message, which will be displayed on your opponent’s side when it accepts your call. Press ‘Finish’. If any of your inputs was faulty, you will be informed by that, otherwise a new button with a light yellow telephone icon will appear at the bottom section of the main screen. Now you have to wait until your call partner accepts the call. If it doesn’t, you can quit the call by clicking on the button and select ‘Hang up’, otherwise the light yellow telephone icon will turn into a green one and the call will be activated immediately (forcing the current active call to wait).

Accepting an incoming call:

If someone starts a call to your IP-address, you will get a flashing button in the call panel. Click on the button to show a dialog window. If you don’t want to accept the call, press ‘Hang up’. Otherwise type in a welcome message and click on ‘Activate’ to accept the call and make it active or click on ‘Set Waiting’ to suspend it (e.g. you have an important call currently running).

Enabling engines:

Simply click on the big ‘Synthesizer’ button to enable speech output, do the same with the button ‘Recognizer’ to activate speech recognition. System messages in the log window will inform you on success. To hold a traditional telephone conversation press the button ‘Speech Transmission’. This will open a dialog window where you can set an audio codec. Select one and press ‘OK’. These steps also have to be done on your call partner’s side at the same time. The application will wait a certain amount of time (can be adjusted in the preferences window) for the connection to be set up. The log window will inform you about a successful establishment. To disable engines press the buttons again.

Sending messages:

With an active call and enabled engines you should be able to send and receive messages, get the incoming text synthesized and have your speech recognized. You can also type in messages directly in the ‘Input directly’ labeled text field.

Exiting VoIP4Text:

To exit the application simply press Alt-F4, press the x in the upper right window corner or select ‘Exit VoIP4Text’ from the ‘File’ menu. You don’t have to quit all calls manually, they will be closed instantly.

**Linkverzeichnis:**

[1] ELKO – Das Elektronik-Kompendium
http://www.elektronik-kompendium.de

[2] Sphinx-4 - A speech recognizer written entirely in Java
http://cmusphinx.sourceforge.net/sphinx4/

[3] FreeTTS - A speech synthesizer written entirely in Java
http://freetts.sourceforge.net/docs/index.php

[4] Wikipedia: Spracherkennung
http://en.wikipedia.org/wiki/Speech_recognition

[5] Java Media Framework
http://java.sun.com/products/java-media/jmf/index.jsp

[6] Voice over IP Fundamentals –
A Systematic Approach to Understanding the Basics of VoIP
http://www.ciscopress.com

[7] History and Development of Speech Synthesis
http://www.acoustics.hut.fi/publications/files/theses/lemmetty_mst/chap2.html

[8] Patay, Helmut: Go To Java
Addison-Wesley Verlag, © 2003