Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Thomas Ahlhoff

Noteneingabe mit Hilfe von Spracherkennung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Thomas Ahlhoff

Noteneingabe mit Hilfe von Spracherkennung

Abschlussarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing. Andreas Meisel
Zweitgutachter : Prof. Dr. rer. nat. Wolfgang Fohl

Abgegeben am 26. September 2015

Thomas Ahlhoff

Thema der Bachelorarbeit

Noteneingabe mit Hilfe von Spracherkennung

Stichworte

Spracherkennung, Sphinx4, Lilypond

Kurzzusammenfassung

Im Rahmen dieser Arbeit soll eine Anwendung entwickelt werden, die es dem Benutzer mit einfachen Sprachbefehlen ermöglicht eine Textdatei zu erstellen und daran anschließend von dem textbasierenden Notensatzprogramm LilyPond genutzt wird, um daraus einen Notensatz zu erzeugen.

Thomas Ahlhoff

Title of the paper

music notation using speech recognition

Keywords

speech recognition, Sphinx4, LilyPond

Abstract

In this work, an application will be developed that allows the user with simple voice commands to create a text file. This file can be used from the text-based music notation program LilyPond to produce a music notation.

Inhaltsverzeichnis

1 Einleitung

1.1 Zielsetzung

Im Rahmen dieser Arbeit soll eine Anwendung entwickelt werden, die es dem Benutzer mit einfachen Sprachbefehlen ermöglicht eine Textdatei zu erstellen und daran anschließend von dem textbasierenden Notensatzprogramm LilyPond genutzt wird, um daraus einen Notensatz zu erzeugen. Die Befehle der zu entwickelnden Anwendung sollen durch Analyse des LilyPond-Syntax gewonnen und mit Hilfe einer einfachen Grammatik in eine Befehlssprache überführt werden. Ferner soll im Verlauf dieser Arbeit auch auf verschiedene Aspekte der Spracherkennung eingegangen werden.

2 Grundlagen

Dieses Kapitel soll als Einführung in die Problematik dieser Bachelorarbeit dienen und das Verständnis des Lesers diesbezüglich erweitern. Dazu wird nachfolgend das Thema der Spracherkennung näher beleuchtet und ein Überblick über die derzeit am meisten verbreiteten Spracherkennungstools gegeben. Der zweite Teil der Grundlagen widmet sich anschließend dem Notensatzprogramm Lilypond und dem einfach zu verwendenden Editor Frescobaldi.

2.1 Spracherkennung und Tools

Die automatische Spracherkennung stellt eine Schnittstelle der Mensch zu Maschine Kommunikation dar. Sie ermöglicht computerbasierende Systeme ohne physische Eingabegeräte (z.B. Tastatur oder Maus) zu bedienen. Die Aufgabe der Spracherkennung ist, die mündliche Aussage eines Benutzers maschinell zu erkennen und als konkreten Text wiederzugeben. Dessen Zeichenfolge kann anschließend von speziellen Programmen dazu verwendet werden, um zum Beispiel einen Dialog mit dem Benutzer zu führen oder eine beliebige Anwendung zu steuern.

Bevor der Spracherkennungsprozess erfolgen kann, muss die analoge Sprache zunächst vorverarbeitet werden. Die Aufgabe der Vorverarbeitung ist die Extraktion von Referenzvektoren aus dem Eingangssignal. Diese werden auch Merkmalsvektoren genannt und repräsentieren eine Art Muster der gesprochenen Äußerung, die der eigentlichen Spracherkennung später als Referenz dienen. Die Vorverarbeitung beginnt mit der Digitalisierung der Sprache. Dabei wird das analoge Signal abgetastet und in eine elektronisch weiter bearbeitbare Bitfolge umgewandelt. Der nächste wichtige Schritt ist die Entfernung von Störgeräuschen (z. B. Hintergrundrauschen) und wird Filterung genannt. Dieser soll sicherstellen, dass die Erstellung der Merkmalsvektoren möglichst nur auf den für die Spracherkennung relevanten Frequenzen (300Hz-5kHz) beruht. Die anschließende Transformation ist die Zerlegung des digitalen Signals in seine Frequenzanteile mittels FFT (fast Fourier transform). Aus all den dann vorliegenden Daten werden nun die Merkmalsvektoren erstellt und somit ist die Vorbearbeitung abgeschlossen.[Car12]

Bei der eigentlichen Spracherkennung geht es einfach ausgedrückt darum, die gesprochene Äußerung stückweise zu analysieren. Dabei werden zuerst die einzelnen Buchstaben erkannt, diese zu Wörtern zusammengesetzt und anschließend daraus ganze Sätze gebildet.

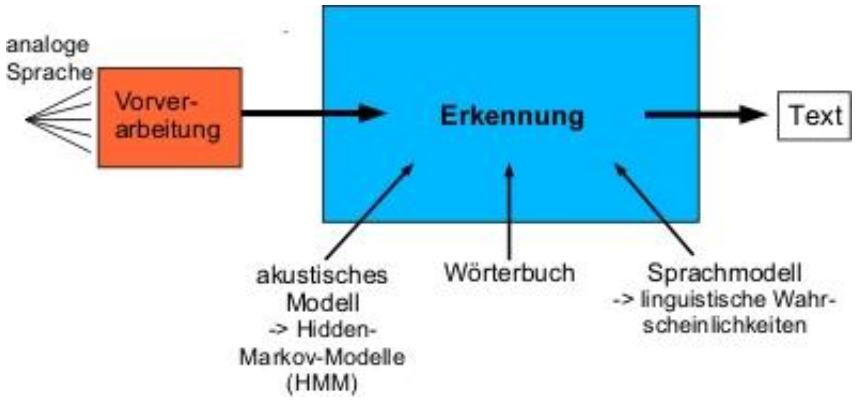

Abbildung 1 Vorgehensweise Spracherkennung [VoSp]

Um dies zu erreichen, muss der Spracherkennung ein akustisches Modell zur Verfügung stehen, das alle zu erkennenden Phoneme enthält. Ein Phonem stellt in der Linguistik die kleinste bedeutungsunterscheidende sprachliche Einheit (z. B. /a/ oder /o/) dar und wird mit einem Hidden-Markov-Modell (HMM) abgebildet. Die Größe des HMM richtet sich nach der Länge des modellierten Phonems, das innerhalb des Modells in Teilstücken (Anfang, beliebige Anzahl Mittelstücke, Schluss) zerlegt vorzufinden ist. Während der Erkennung werden die Eingangssignale (Merkmalsvektoren) mit den in den Modellen abgelegten Teilstücken verglichen. Danach wird mit Hilfe von geeigneten Algorithmen das wahrscheinlichste Phonem berechnet. Ein Wörterbuch, das jedes zu erkennende Wort mit den dazu passenden Phonemreihen beinhaltet, setzt anschließend die möglichen Wörter bzw. Sätze zusammen.[Wöl09]

Im letzten Teil des Erkennungsprozesses kommt ein Sprachmodell zum Einsatz. Für jede erkannte Wortkombination wird ein Wahrscheinlichkeitswert berechnet und anhand dieses Wertes versucht, die vom Sprecher gesagte Äußerung herauszufiltern. Als Sprachmodell kann entweder ein Grammatikmodell oder ein auf N-Grammen basierendes Sprachmodell zum Einsatz kommen. Die beiden Modelle werden In Kapitel 3.2.2. genauer beleuchtet.

2.1.1 HTK - Hidden Markov Model Toolkit

Das Hidden Markov Modell Toolkit (HTK) ist eine Sammlung von Werkzeugen zum Erstellen und Bearbeiten von Hidden Markov Modellen (HMM). Dieses Toolkit wird hauptsächlich für die Spracherkennungsforschung verwendet, aber auch in zahlreichen anderen Anwendungen, welche die Erforschung der Sprachsynthese, Zeichenerkennung und DNA-Sequenzierung mit einschließen. Die Bibliotheken und Tools des HTK stehen als C-Source-Code zur Verfügung und ermöglichen es dem Anwender über Konfigurationsdateien und Startparameter Sprachmodelle zu erzeugen, zu manipulieren und zu testen. Weiterhin beinhaltet das Paket eine umfangreiche Dokumentation und anschauliche Beispiele.

Ursprünglich wurde das HTK am Machine Intelligence Laboratory des Cambridge University Engineering Department (CUED) entwickelt und dazu genutzt, CUED's large vocabulary speech recognition System (HTK LVR) zu erstellen. Von 1993 - 1999 konnte das HTK nur kommerziell erworben werden, aber seit dem Jahr 2000 steht es wieder zur freien Verfügung und ist über die Website der Cambridge University zu beziehen. Aktuell ist HTK 3.4.1. eine als Download verfügbare stabile Version. [HTK]

2.1.2 Julius

Julius ist ein schnelles LVCSR (large vocabulary continuous speech recognition) Spracherkennungsprogramm, mit dem es möglich ist, die Erkennung, von einzelnen Wörtern oder ganzen Sätzen, in nahezu Echtzeit durchzuführen. Laut Entwickler ist dies mit den meisten aktuellen PCs (Personal Computer) und einem Vokabular, von bis zu sechzigtausend Wörtern realisierbar. Dabei benötigt Julius lediglich 32 Megabyte Arbeitsspeicher.

Die Entwicklung begann 1997 als Forschungssoftware für ein japanisches LVCSR und wird zurzeit vom Interactive Speech Technology Consortium (ISTC) fortgeführt. Julius wird, zusammen mit dem Source-Code, unter einer freien Lizenz vertrieben und ist hauptsächlich für Linux Plattformen ausgelegt. Jedoch bietet Julius keinerlei Tools zum erstellen eines Sprachmodells, welches für den Betrieb benötigt wird. Daher muss entweder auf ein vorhandenes Modell zurückgegriffen, oder ein Solches, mit einem externen Tool (z.B. HTK) erstellt werden. [LEE10]

2.1.3 CMU Sphinx

Ein weiteres sehr bekanntes Spracherkennungstoolkit ist CMU Sphinx, welches von der Carnegie Mellon University entwickelt wurde und wird. Es beinhaltet verschiedene Programme, die zur Erstellung verschiedenster Sprachanwendungen genutzt werden können.

Die im Toolkit enthaltene Spracherkennungssoftware Sphinx4 stellt die aktuelle Version dar und bietet eine, auf einem HMM basierende sprecherunabhängige Spracherkennung. Sphinx4 wurde vollständig in Java implementiert und benötigt deshalb zum Betrieb eine Java Virtual Machine mit 1 Gigabyte Arbeitsspeicher.

Ein weiteres Werkzeug aus dem Paket ist das Programm SphinxTrain. Dieses ermöglicht dem Anwender ein eigenes Sprachmodell zu erzeugen. [Wal04]

2.1.4 Simon Listen

Simon Listen ist eine Open-Source-Spracherkennungssoftware mit Client-Server-Architektur, die entwickelt wurde, um körperbehinderten Personen und Senioren die Bedienung eines Personal Computers zu erleichtern. Die Hauptbestandteile sind der Server simond, das Haupt-Frontend simon und das Werkzeug sam.

Die Serverkomponente des Simon Listen Projektes ist simond und für die eigentliche Erkennung bzw. für die Erzeugung des Sprachmodells zuständig. Alle dafür benötigten Dateien sowie die Daten vom Mikrofon werden von den simon Client(s) via Netzwerk bereitgestellt.

Der Client simon bietet eine grafische Benutzeroberfläche. Diese kann dazu benutzt werden, um eigene Sprachmodelle zu erzeugen, vorhandene zu bearbeiten und zu testen. Desweiteren können Szenarien erstellt werden, mit denen sich Programme (z. B. Firefox), rein über erkannte Kommandos steuern lassen. [Gra10]

2.2 LilyPond

Das Notensatzprogramm LilyPond ist mit dem Ziel entwickelt worden, das Erscheinungsbild von computererstellten Notendrucken zu verbessern und einen eleganten, leicht zu lesenden Notensatz zu erstellen. Die stilistischen Einstellungen, das Schriftartendesign und die Algorithmen orientieren sich an den besten handgestochenen Notenbeispielen um der Ausgabe das ausbalancierte und vornehme Aussehen einer klassischen Partitur zu verleihen. [Lily]

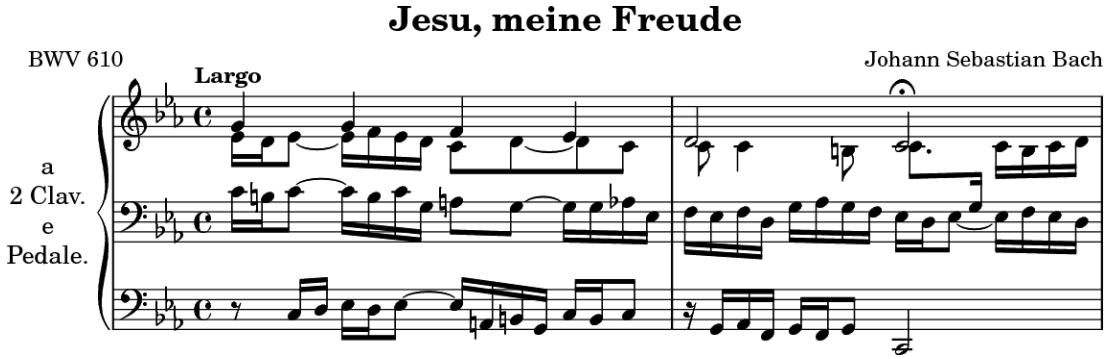

Abbildung 2: LilyPond Beispiel [LilyB]

Die Aufgabe der Formatierung, um ein dichtes und gleichmäßiges Notenbild zu erhalten, wird von LilyPond selbst übernommen. Dabei werden zum Beispiel die Platzaufteilung und die Zeilen- und Seitenumbrüche selbst errechnet. Zusammenstöße zwischen Gesangstext, Noten und Akkorden werden ebenfalls automatisch aufgelöst und Bögen (z. B. Bindebogen)

richtig gekrümmt. Sollte der Benutzer dennoch nicht mit dem Ergebnis zufrieden sein, so steht es ihm frei, alle Einstellungen an seinen persönlichen typografischen Geschmack anzupassen. Das Programm bietet außerdem die Möglichkeit, über verschiedene Plugins, Noten in LaTeX, HTML, OpenOffice.org-Dokumente, sowie Blogs und Wikis zu integrieren.

Die Anwendung LilyPond wird als freie Software unter einer GNU General Public License vertrieben und für alle gängigen Betriebssysteme auf der Webseite (www....) zum Download angeboten. Neben den Installationspaketen für Linux, MacOS X und Windows werden darüber hinaus auch der Quellcode, Handbücher sowie mehrere hunderte Beispieldateien kostenlos bereitgestellt. Die freie Lizenz erlaubt es jedem Benutzer den Quellcode zu verändern und somit zur Fehlerbehebung oder zur Erweiterung der Funktionalität beizutragen. [Lily]

Im Gegensatz zu anderen Notensatzprogrammen (z. B. FORTE), die über ein grafisches Benutzerinterface (engl. **G**raphical **U**ser **I**nterface) verfügen und bei denen der Nutzer die einzelnen Noten mit der Maus in einem dynamischen Notensystem platziert, verzichtet LilyPond auf eine solche GUI. Die Eingabe erfolgt vielmehr über eine einfache Textdatei, welche im ASCII-Format vorliegt und die Dateiendung .ly besitzt. Der Inhalt dieser Datei beschreibt in Textform das Aussehen des Notenblattes und kann mit einem beliebigen Texteditor bearbeitet werden. Mit dem Kommandozeilenaufruf "lilypond DATEINAME.ly" oder durch einen Doppelklick (unter Windows) auf die .ly-Datei beginnt LilyPond zu arbeiten. Der Text innerhalb der übergebenen Datei wird analysiert, auf Fehler überprüft und abschließend der Notensatz erstellt, der per default im PDF-Format (DATEINAME.pdf) gespeichert wird.

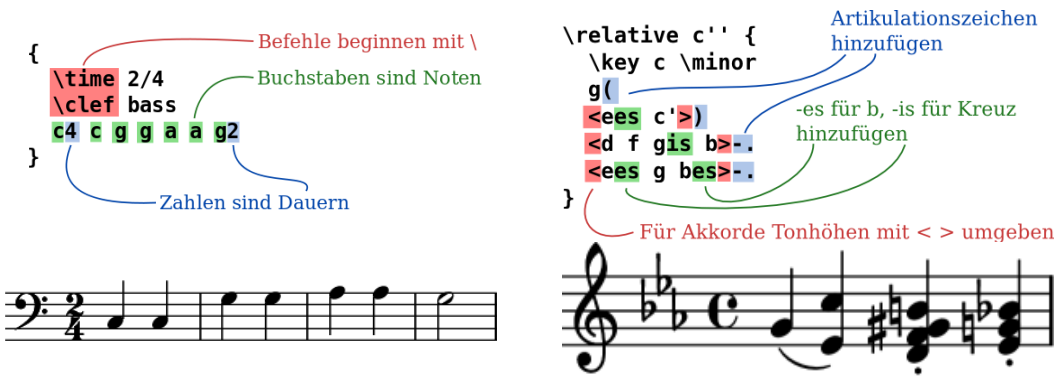

Abbildung 3: LilyPond Beispiele [LilyA]

Zwei sehr einfache Beispiele für den Inhalt der LilyPond-Datei und dem jeweils dazu gehörenden erzeugten Notensatz sind in der Abbildung 3 nebeneinander dargestellt. Da LilyPond eine Vielzahl von Befehlen und Funktionen besitzt, ähnelt es eher einer Programmiersprache als einem graphischen Notensatzprogramm. Neulinge sollten also die Handbücher lesen und eine gewisse Einarbeitungszeit in Kauf nehmen.

2.2.1 LilyPond - Dateistruktur

Der Aufbau der LilyPond-Dateistruktur lässt sich am besten als ein verschachteltes System darstellen. Vorzustellen wie bei einem handelsüblichen Buch (Buch -> Kapitel -> Seite -> Absatz -> Satz -> Wort) lässt sich die Dateistruktur in immer kleiner werdende Blöcke oder Ebenen unterteilen. Jeder Ausdruck bzw. Befehl ist dabei mindestens einer Ebene zugeordnet. Die von ihm definierten Eigenschaften gelten für diese und alle darunterliegenden Ebenen, bis sie von einem anderen Befehl verändert oder aufgehoben werden. Eine .ly-Datei, in der nur ein zusammengesetzter musikalischer Ausdruck, wie zum Beispiel: { c'4 d' e'2 }, gespeichert ist, wird von LilyPond vor der Interpretation automatisch in folgende Grundstruktur übersetzt.

```
\book {
  \score {
    \new Staff {
      \new Voice {
        { c'4 d' e'2 }
      }
    }
    \layout { }
  \header { }
}
```

Wie im obigen Beispiel zu erkennen, können die Ebenen als einzelne Umgebungen zusammengefasst werden. Diese Ebenen beginnen mit einem Ausdruck (z. B. \book), der die Funktion beschreibt, und betten die darunterliegenden Ebenen in geschweifte Klammern ein. Die \book-Umgebung ist auf der obersten Ebene angeordnet und stellt das Notenbuch dar. In diesem Buch können mehrere Partituren abgelegt werden, da sie jeweils eine eigene \score-Umgebung bilden. Als Partitur wird die Aufzeichnung mehrstimmiger Musik in Notenschrift bezeichnet. Dabei werden die einzelnen Stimmen, im allgemeinen Instrumente wie Flöte oder Violine, untereinander abgebildet und mit durchgezogenen Taktstrichen verbunden. Die nächste Ebene ist die \Staff-Umgebung. die die Eigenschaften der Notenzeile beschreibt und die auf dieser Zeile dargestellten Stimmen umfasst. Jede Stimme bzw. jedes Instrument erzeugt eine eigene \Voice-Umgebung und enthält neben den darzustellenden Noten (musikalischer Ausdruck) weitere Informationen, wie zum Beispiel den Namen des Instrumentes. Die Befehle \layout und \paper dienen der Ausgabedefinition und können genutzt werden, um die Standartformatierung seitens LilyPond abzuändern. Mit \layout lassen sich zum Beispiel der Zeilenabstand oder die Schriftgröße anpassen. Wogegen \paper für die Einrichtung der Seite (Hoch/Querformat) oder Anpassung der Blattgröße (z. B. DIN A5) genutzt wird. Einstellungen, welche den Titel oder den Komponisten beschreiben, werden in der \header-Umgebung definiert. Diese können auch für das ganze Buch auf der obersten Ebene vorgenommen werden. [Lily]

2.2.2 Frescobaldi

Der Editor Frescobaldi wurde mit dem Ziel entwickelt, die Erstellung und Bearbeitung der LilyPond-Dateien schneller und leichter zu gestallten. Die GUI des Programms ist in zwei Hälften aufgeteilt. Auf der linken Seite befinden sich das große Texteditorfenster und darunter die Lilypond-Protokollausgabe. Ein Komfortmerkmal des Editors ist, das Erzeugen des Notensatzes mit einem Mausklick durchführen zu lassen. Frescobaldi speichert den Inhalt des Texteingabefensters und übergibt diese Datei an LilyPond. Dieser Aufruf erfolgt im Hintergrund und alle für die Bearbeitung relevanten Informationen (z. B. Fehlerberichte oder Arbeitsschritte) werden in der LilyPond-Protokollausgabe ausgegeben. Sofern bei diesen Arbeitsschritten keine Fehler auftreten, wird das von LilyPond gerade erstellte Notenblatt auf der rechten Seite der GUI angezeigt. [Fres]

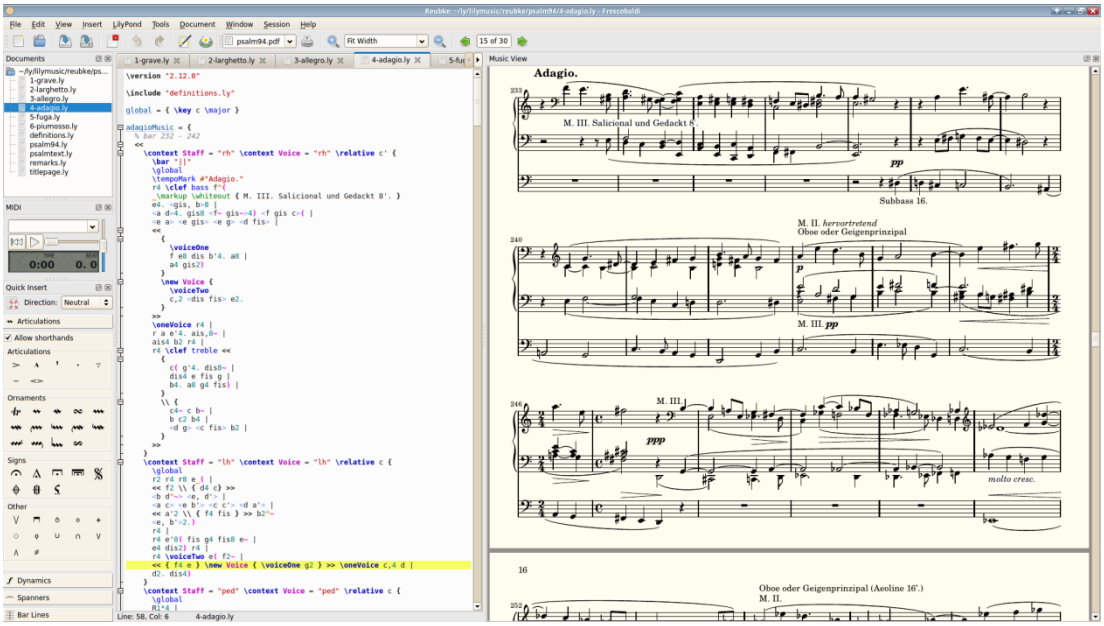

Abbildung 4 Frecobaldi Editoraufbau [FresA]

Frescobaldi bietet einen Partitur-Assistenten an, der den Benutzer bei der Erstellung neuer Partituren unterstützt. Dabei werden Textgrundgerüste erstellt, denen lediglich noch Noten und Gesang hinzugefügt werden müssen. Die dazu benötigten Informationen reichen von Titel oder Komponist über zu verwendende Instrumente oder Stimmen bis hin zu Tonart und Tempo. Eine farbliche Darstellung von bestimmten Befehlen oder Schlagwörtern sowie eine Autoformat-Funktion für den Quellcode (Text) sorgen für eine bessere Übersicht innerhalb des Texteditorfensters und runden das Gesamtpaket ab. Angeboten als freie Software ist dieses Programm kostenlos auf der Webseite von Frescobaldi zu haben. [Fres]

3 Anforderungsanalyse

3.1 Befehlsextraktion

Die vollständige Implementierung aller von LilyPond bereitgestellten Befehle und Funktionen sprengt den Rahmen dieser Arbeit. Daher ist es erforderlich eine Auswahl zu treffen, die die wichtigsten Befehle enthält um damit eine gewisse Grundfunktionalität des Programms zu gewährleisten. Als Grundlage für die Befehlsfindung dienen fünfundzwanzig LilyPond-Dateien, die Chorgesang darstellen (Beispieldatei "Die Nacht" auf CD), und die LilyPond-Benutzerhandbücher. Nach dem Einlesen in das "LilyPond - Learning Manual" wurden die zugrundeliegenden Dateien, unter Einsatz des Editors Frescobaldi, einzeln durchgearbeitet und miteinander verglichen. Das Ziel dieser Analyse war zum einen die Häufigkeit des Auftretens der verwendeten Befehle zu ermitteln und zum anderen ein Verständnis für die Syntax der LilyPond-Dateien zu entwickeln. Unter Verwendung der "LilyPond - Notationsreferenz" konnten die Funktionen der Ausdrücke verstanden und festgelegt werden, ob dieser Befehl in dem Programm implementiert werden sollte. Die Befehle sind zum besseren Verständnis in drei Gruppen eingeteilt.

Die erste Gruppe bilden die nachfolgend aufgeführten und kurz erläuterten LilyPond-Befehle, deren Hauptaufgabe darin besteht, die Struktur des zu erstellenden Notenblattes festzulegen.

`\new Book`	Anlegen eines neuen Notenbuches
`\new Score`	Anlegen einer neuen Partitur
`\new Staff`	Anlegen einer neuen Notenzeile
`\new Voice`	Anlegen einer neuen Stimme
`\time`	Festlegen des Musiktaktes
`\partial`	Festlegen eines Auftaktes
`\chord`	Notation von Akkorden
`\repeat`	Definition von Wiederholungen
`\unfoldRepeats`	Definition ausgeschriebener Wiederholungen
`\alternative`	Definition von Wiederholungsalternativen
`\break`	manueller Zeilenumbruch
`\tublet`	Erzeugung von Triolen
`\relative`	relative Oktavenbezeichnung

Alle anderen LilyPond-Ausdrücke lassen sich in einer zweiten Gruppe zusammenfassen. Diese Gruppe beinhaltet sämtliche zur Beschreibung der musikalischen Noten bzw. Binde- und Legatobögen wichtigen Textbausteine. Die einfache Notation der verschiedenen Töne (Musiknoten) im Quellcode erfolgt durch die Zusammensetzung von Buchstaben und Zahlen. So besteht ein einfacher Ton aus einem Buchstaben (c-d-e-f-g-a-h-c) für die Tonhöhe und einer Zahl (z. B. eine 2 für einen Halbton) für die Länge des Tones. Dieser Ton kann im Notensystem mit den Steuerzeichen Komma (,) und Hochkomma(') um jeweils eine Oktave nach unten bzw. nach oben versetzt werden. Zur Verlängerung der Tonlänge wird ein Punkt (.) genutzt und ein Bindebogen mit dem Zeichen Tilde (~) notiert. Legatobögen werden mit einfachen Klammern "(" ")" gebildet, in denen die betreffenden Noten von den Klammern umschlossen sind. [Lily]

In die letzte Gruppe gehören Befehle, die zur Steuerung des Programmes, zur Formatierung der Ausgabedatei und zum Einfügen benutzerspezifischer Inhalte genutzt werden können. Dabei handelt es sich nicht um LilyPond-Befehle, sondern um selbst definierte Anweisungen, mit denen die folgenden Anforderungen realisiert werden können. Der Anwender soll neben der Möglichkeit vom Programm falsch erkannte bzw. vom User falsch formulierte Äußerungen zu korrigieren auch in der Lage sein, das Programm mit einem Sprachbefehl zu beenden. Darüber hinaus soll die Lesbarkeit des erzeugten Quellcodes durch das Einfügen von Leerzeilen erhöht werden. Eine weitere Anforderung, die vom Programm erfüllt werden soll, stellt das Hinzufügen von Inhalten dar, die nicht mit den Sprachbefehlen abgedeckt werden können. In diese Kategorie fallen zum Beispiel die, in den analysierten Dateien vorzufindenden, Gesangstexte oder Variablendefinitionen, welche zur Zusammenfassung von musikalischen Ausdrücken genutzt werden.

Die folgenden Befehle stellen den Lösungsansatz zu diesen Problematiken dar.

`correction`	Korrektur des letzten Sprachbefehls
`end application`	Beenden des Programmes
`newline`	Einfügen einer Leerzeile
`variable`	Einfügen einer Textzeile
`file`	Einfügen eines Textabschnittes

Ein Teilziel dieser Arbeit ist es, aus den durch die Analyse gewonnenen Befehlen und Ausdrücken, eine für den Benutzer leicht zu erlernende Befehlssprache zu entwickeln, mit der das Programm bedient und der Notensatz erstellt werden kann.

3.2 Anforderungen an die Spracherkennung

3.2.1 Akustikmodell

Bei der Auswahl des später zu verwendenden Akustikmodells geht es weniger um dessen Aufbau (Abbildung der Phoneme im HMM) oder Inhalt (verwendete Sprache oder Anzahl der abgebildeten Wörter), sondern um den Einsatzzweck des Spracherkennungssystems. Derzeit kann die Spracherkennung in zwei Arten (sprecherabhängige und sprecherunabhängige) unterteilt werden.

Sprecherabhängige Systeme arbeiten auf Grundlage eines, an den späteren Benutzer, angepassten, Akustikmodells. Bevor eine Spracherkennung durchgeführt werden kann, muss das Modell vom Benutzer trainiert werden. In dieser Trainingsphase wird das Akustikmodell mit den Besonderheiten der Aussprache (z. B. Stimmlage oder ein verwendeter Dialekt) abgestimmt, was zur Erhöhung der Erkennungswahrscheinlichkeit beiträgt. Auf Grund dieser Abhängigkeit ist die Verwendung dieses Modells durch einen anderen Benutzer kaum möglich und sollte daher nur in Anwendungen (z. B. Diktat von Texten) mit festem Benutzer zum Einsatz kommen.

Im Gegensatz dazu sind sprecherunabhängige Spracherkenner in der Lage jeden Benutzer zu verstehen, solange sich die Aussprachemerkmale in einem gewissen Toleranzbereich bewegen. Die für diese Art der Spracherkennung verwendeten Akustikmodelle werden aus den Trainingsdaten einer sehr großen Sprechergruppe (mindestens hundert, besser über tausend) erzeugt und können meist über die Webseite des Spracherkennungsprogramms bezogen werden. Durch den enormen Trainingsaufwand kommen solche Systeme bei Anwendungen (z. B. automatisches Dialogsystem einer Telefonauskunft) mit beschränktem Wortschatz zum Einsatz.

Auf Grund des kleinen Wortschatzes und mit dem Ziel, die Benutzung des Programmes durch verschiedene Anwender zu ermöglichen, soll in dieser Arbeit ein sprecherunabhängiges Spracherkennungssystem zum Einsatz kommen.

3.2.2 Sprachmodell

Innerhalb eines Spracherkenners bildet das Sprachmodell die statistische Beschreibung der Sprache (z. B. Reihenfolge der Befehlsparameter oder Satzbau bei natürlicher Sprache) ab. Die Aufgabe eines solchen Sprachmodells liegt in der Bestimmung und Bewertung der Wahrscheinlichkeiten möglicher Wortkombinationen. Anschließend kann auf dieser Grundlage die wahrscheinlichste Wortkombination bestimmt werden. Bei Sprachmodellen wird zwischen Grammatiken und N-Gramm-Modellen unterschieden.

Grammatiken

Grammatikmodelle werden meist mit einer regulären Grammatik beschrieben und bieten dem Benutzer die Möglichkeit, das Sprachmodell leicht an seine Bedürfnisse anzupassen. Sie sind besonders gut dafür geeignet, kurze Wortfolgen zu beschreiben und lassen sich anschaulich in einem Zustandsdiagramm darstellen. Diese Diagramme werden mit wachsender Strukturgröße unübersichtlich.

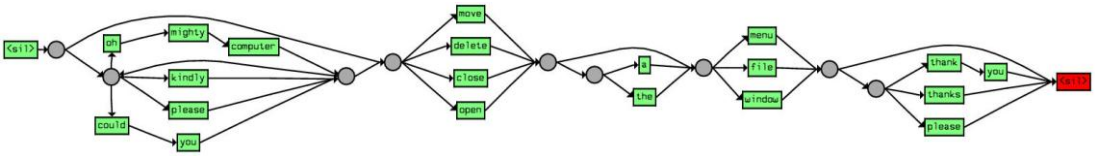

Abbildung 5 Darstellung einer Grammatik [JSGF]

Die folgende im Java Speech Grammar Format (JSGF) beschriebene Beispielgrammatik ist in Abbildung 5. als Zustandsdiagramm dargestellt. [JSGF]

```
#JSGF V1.0
public <basicCmd> = <startPolite> <command> <endPolite>;
<command> = <action> <object>;
<action> = /10/ open |/2/ close |/1/ delete |/1/ move;
<object> = [the | a] (window | file | menu);
<startPolite> = (please | kindly | could you
     | oh mighty    computer) *;
<endPolite> = [ please | thanks | thank you ];
```

Das Java Speech Grammar Format ist eine Plattform- und Herstellerunabhängige text-basierende Darstellungsform einer Grammatik, das von der Java Speech API (JSAPI) zur Spracherkennung genutzt wird. Das JSGF unterstützt die Merkmale der regulären Grammatik, wie die Beschreibung von Regeln für Sequenzen, Alternativen, Referenzen, deren Gruppierungen, unterschiedliche Gewichtungen und den Kleene- und Plus-Operator. Die genau Handhabung dieser Regeln ist in der von *Sun Microsystems* veröffentlichten Dokumentation (vgl. JAVA Sun, 2000) zu finden. [JSGF]

N-Gramm-Modelle

Im Gegensatz zu einer Grammatik wird bei N-Gramm-Modellen, die Wahrscheinlichkeit des Auftretens einer Wortgruppe nicht durch festgelegte Regeln errechnet, sondern durch statistische Erhebungen gewonnen. Ein N-Gramm ist das Ergebnis der Zerlegung eines Textes in Fragmente (z. B. Wortgruppen). Das "N" steht in diesem Fall für die Anzahl der im Fragment enthaltenen Wörter. Aus der Anzahl dieser, im Text vorhandenen, einzigartigen Fragmente wird eine Statistik über deren Auftrittswahrscheinlichkeiten, im Verhältnis zur Anzahl der Wörter im Text, erstellt.

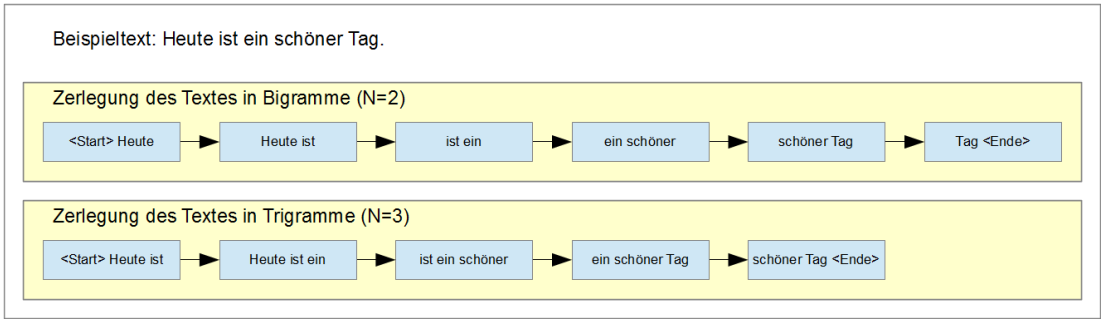

Abbildung 6 Zerlegung in Bi- und Trigramme

Eine Zerlegung des Satzes "Heute ist ein schöner Tag." in N-Gramme ist in Abbildung 6. veranschaulicht. Die aus dem Beispiel gewonnene Statistik ist wenig aussagekräftig, da jedes N-Gramm nur einmal vorhanden ist. Das Training einer aussagekräftigen Bigrammstatistik (N = 2) oder Trigrammstatistik (N = 3) sollte daher mit einem großen Textkorpus (mehrere Millionen Wörter), der die zulässigen Wortkombinationen in statistisch signifikanter Anzahl enthält, stattfinden.

Das in dieser Arbeit verwendete Sprachmodell soll durch eine im Java Speech Grammar Format beschriebenen Grammatik erzeugt werden, da diese für die Beschreibung der zu erstellenden Befehlssprache besser geeignet ist.

3.2.3 Das Homophonen-Problem

Als Homophone werden Wörter bezeichnet, die im Vergleich zu einem anderen Wort mit unterschiedlicher Bedeutung, die gleiche Aussprache (z. B. "fiel" und "viel") besitzen. Diese Homophonen stellen bei der Spracherkennung ein gewisses Problem dar, da die Erkennung anhand des eingehenden Sprachsignals durchgeführt wird. Das Akustikmodell kann keinerlei Aussage über die Sinnhaftigkeit des gesagten Wortes treffen und wird die Wahrscheinlichkeiten der Worte "fiel" und "viel" annähernd gleich bewerten. Somit kann der Satz "Das ist viel." als "Das ist fiel." verstanden werden. Dieses Problem wird im Normalfall von einem Sprachmodell behoben, da mit den Wahrscheinlichkeiten bestimmter Wortkombinationen unwahrscheinliche Hypothesen ausgeschlossen werden können und somit der korrekte Satz "Das ist viel." die beste Bewertung erhält.

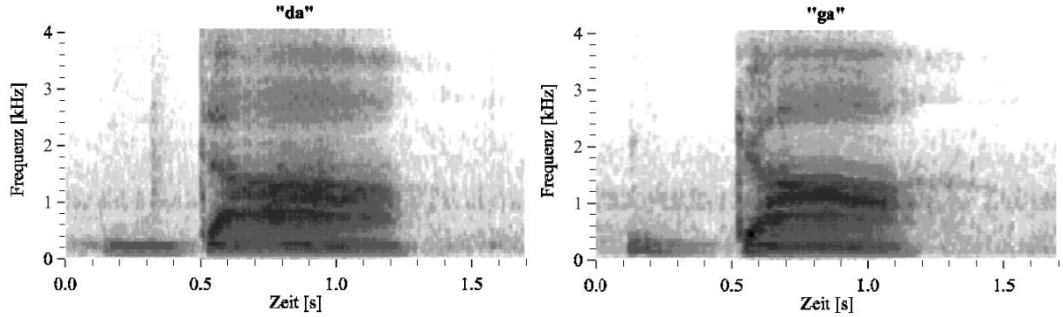

Abbildung 7 Frequenzspektrum der Silben da und ga [Spek]

Das oben beschriebene Problem kann, bedingt durch Störgeräusche oder Variationen in der Aussprache, auch bei kurzen Wörtern und vor allem bei Silben auftreten, die ähnlich klingen. Die Abbildung 7. zeigt die Frequenzspektren der Silben "da" und "ga" und soll nur zur Veranschaulichung der Ähnlichkeiten dienen. Bezogen auf diese Arbeit lassen sich diese Ähnlichkeiten bei der Notation der Tonhöhe durch die Buchstaben (c-d-e-f-g-a-h-c) wiederfinden. Besonders die Laute der Buchstaben "d" "e" und "g" erzeugen fasst das gleiche Klangbild. In diesem Fall kann das Problem nicht vom Sprachmodell gelöst werden, da jede Kombination (d2 oder e4) von Tonhöhe und Tonlänge, statistisch gesehen, mit gleicher Wahrscheinlichkeit auftreten kann.

Im Verlauf der Arbeit soll untersucht werden ob das gerade beschriebene Problem mit Hilfe einer Buchstabiertafel (z.B. NATO-Alphabet) umgangen bzw. gelöst werden kann.

4 Designentscheidungen

4.1 Sprachbefehle

Damit die in Kapitel 3.1 Befehlsextraktion durch die Analyse der LilyPond-Dateien gewonnenen Befehle von einem Spracherkennungsprogramm verstanden und korrekt interpretiert werden können, müssen diese vorher in eine natürliche Sprache überführt werden. Die beim Anfertigen dieser Arbeit angewendete Vorgehensweise soll beispielhaft mit dem Befehl \repeat verdeutlicht werden. Als Erstes wird der Syntax des Befehls untersucht.

```
\repeat Typ Wiederholungszähler { musikAusdr }
```

Der abgebildete Befehl besteht aus dem Befehlsbezeichner (\repeat), den Parametern Typ (volta, unfold, percent, tremolo), dem Wiederholungszähler (eine natürliche Zahl) und einem, in geschweiften Klammern notierten, musikalischen Ausdruck. Dem Befehlsbezeichner und jedem in den Parametern vorkommenden Ausdruck wird nun ein Wort aus dem Vokabular des Akustikmodells zugewiesen.

Bei der Auswahl und Zuweisung der Wörter wurde versucht, die LilyPond-Syntax weitestgehend beizubehalten, um erfahrenen Benutzern das Erlernen der natürlichen Befehlssprache zu vereinfachen. Im Vokabular nicht enthaltene Worte wurden durch ähnlich klingende (z. B. "volt" ersetzt "volta") bzw. Worte mit ähnlicher Bedeutung ("tremor" ersetzt "tremolo") ersetzt.

4.2 Erstellen der Grammatik

Die Beschreibung der Grammatik erfolgte im schon beschriebenen (Kapitel 3.2.2) *Java Speech Grammar Format* und baut auf die Untersuchung in Kapitel 4.1 auf. Die von den Parametern der LilyPond-Befehle erzeugten Alternativen werden in Referenzen zusammengefasst. Als Beispiel: Die Referenz <nato> stellt immer eine der, in Klammern zusammengefügten und dabei durch einen Balken (|)getrennten, Alternativen dar.

```
<nato> = ( alfa | bravo | charlie | delta | echo | florida
      | golf );
```

Mit Hilfe dieser Referenzen wurden anschließend die Regelsequenzen, welche die Sprachbefehle abbilden, erstellt und wenn möglich in Gruppen zusammengefasst.

```
public <tuplet-takt> = ( triplet | time ) <number> <number>;
```

Das obige Beispiel bildet beide unterschiedlichen LilyPond-Befehle *tublet* und *time* in einer Sequenz ab. Nachfolgend ist der Inhalt der Datei *speechtotone.gram* aufgeführt und definiert die zur Spracherkennung verwendete reguläre Grammatik. Die Beschreibung und Anwendung der Sprachbefehle ist in Kapitel 5.2.2. zu finden.

```
grammar speechtotone;
//Variablendefinition
<nato> = ( alfa | bravo | charlie | delta | echo | florida
      | golf );
<note> = ( c | d | e | f | g | a | b | pause | silent );
<number> = ( one | two | three | four | five | six | seven
      | eight | nine | null);
<halbton> = ( is | es | flat | sharp )*;
<oktave> = ( up | down )*;
<wert> = ( one | two | four | eight | sixteen | thirty two );
<punkt> = ( point )*;
<bogen> = ( slur )*;          //slur - Bindebogen

//eine einzelne Note
public <musik> =
( <note> | <nato> ) <halbton> <oktave> <wert> <punkt> <bogen>;

//LilyPond-Befehle
public <command> = ( break | correction );
public <relative> = (relative) (<note> | close)  (<oktave>);
public <wiederholung> = (repeat) (volt | unfold | percent
      | tremor | close) <number>;
public <tuplet-takt> = ( triplet | time ) <number> <number>;
public <auftakt> = (partial) <wert>;
public <achord> = (chord) <number> <wert>;
public <openCloseCommands> = ( book | score | staff | voice
      | legato | alternative | unfold ) ( open | close );

//Programmsteuerungsbefehle
public <vari> = (variable) <number>*;
public <datei> = (file) <number> <number>;
public <done> = (close) (application);
public <newline> = (new) (line);
```

4.3 Default-Datei

Die Default.txt ist eine einfache Textdatei, die es dem Anwender ermöglicht, mit Hilfe eines Sprachbefehls, eigene Ausdrücke in den LilyPond-Quellcode einzufügen. Sie kann zum Beispiel genutzt werden um Variablen zu definieren, Liedtexte einzufügen oder vom Programm nicht unterstützte Befehle zu verwenden. Die nach Benutzerwünschen erstellte Datei wird der Anwendung beim Programmstart als Parameter (Format: Dateiname.txt) übergeben.

Während der Programmausführung wird mit den Sprachbefehlen "*variable <0-9> <0-9>*" oder "*file <0-9> <0-9>*" der Zugriff auf die erstellten Inhalte ermöglicht. Aus diesem Grund ist der Inhalt der Datei in verschiedene Abschnitte unterteilt, deren Ende mit der Zeichenfolge "**NewTag**" beschrieben wird. Ein kurzer Auszug aus der Default.txt soll nachfolgend den Aufbau der Datei und die Funktionsweise der Sprachbefehle verdeutlichen.

```
\header {
  title = "Bitte Titel in der Default.txt eintragen."
  composer = "Bitte Komponist in der Default.txt eintragen."
}
global = {
  \key e \major
  \time 3/4
}
% ------------------------------------------------
*NewTag*    1
sopMusic =
altoMusic =
*NewTag*    2

...

*NewTag*    12
sopWordsA = \lyricmode {
  \repeat volta 2 {
    No-o no No-o no No-o no no, No-o no no no
    no no No-o no no no no.
  }
}
*NewTag*    13
```

Der oberste Abschnitt ist für den Header bzw. die globalen Einstellungen reserviert und wird immer in die erzeugte LilyPond-Datei übernommen. Somit können die Grundeinstellungen des Musikstückes vor dem Programmstart festgelegt werden. Der zeilenweise Zugriff auf die einstelligen "Tags" (1-9) kann mit dem variable-Befehl erfolgen. Die erste Zahl definiert dabei den Abschnitt und die Zweite die Zeile (0-9) des zu

verwendenden Ausdrucks. Im Gegensatz kann der file-Befehl dazu verwendet werden einen ganzen Abschnitt in den Quellcode einzufügen. Die beiden gesprochenen Zahlen werden deshalb als eine Zahl interpretiert und gewähren den Zugriff auf die "Tags" zehn bis neunundneunzig.

Beispiele:

Sprachbefehl: *variable one one*
fügt ein:

```
altoMusic =
```

Sprachbefehl: *file one two*
fügt ein:

```
sopWordsA = \lyricmode {
\repeat volta 2 {
  No-o no No-o no No-o no no, No-o no no no
  no no No-o no no no no.
  }
}
```

5 Implementierung und Anwendung

Im Rahmen dieser Arbeit wurde in Java ein Programm entwickelt und auf den Namen *SpeechToTone* getauft. Das Programm ermöglicht dem Benutzer durch Sprachsteuerung eine LilyPond-Quellcodedatei zu erzeugen. Aus dieser Datei (.ly) kann anschließend mit dem Notensatzprogramm LilyPond ein schön zu lesender Notensatz erstellt werden.

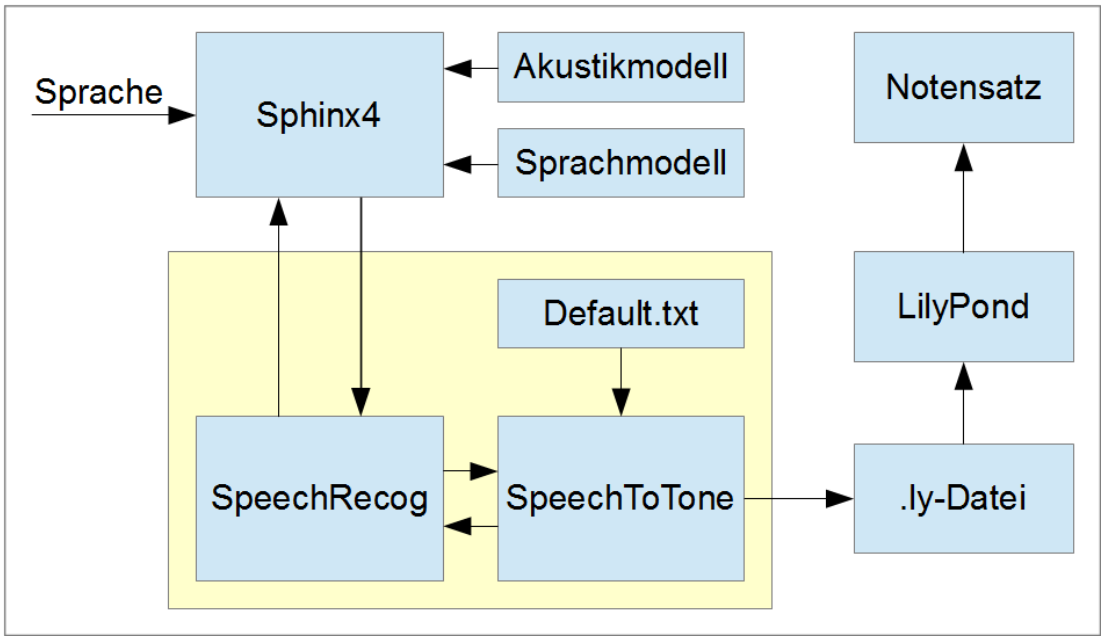

Abbildung 8: Darstellung der Projektstruktur

5.1 Komponenten des Programms

5.1.1 Sphinx4

Das Tool Sphinx4 wurde mit dem Gedanken entwickelt, die Forschung auf dem Gebiet der auf dem Hidden-Markov-Modell basierenden Spracherkennung zu fördern. Zwecks Plattformunabhängigkeit wurde Sphinx4 komplett in Java geschrieben und vereint die

überarbeitete Sphinx engine mit state-of-the-art Spracherkennungtechnologien. Der flexibel und modular gestaltete Aufbau des Frameworks (Abbildung 9) soll es Forschern ermöglichen, das Programm an die jeweiligen Gegebenheiten des Forschungsprojektes anzupassen.

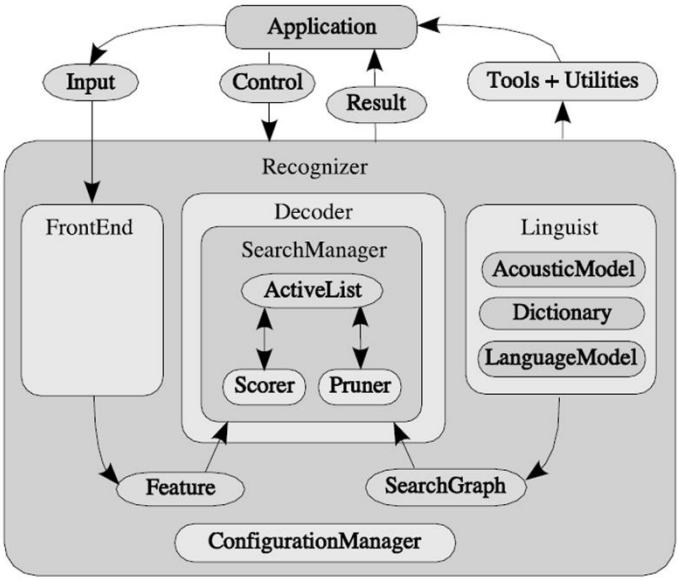

Abbildung 9 Sphinx4 Dekoder Framework [Wal04]

Der Grundaufbau des Sphinx4 Framework lässt sich wie folgt kurz beschreiben. Die eingehende Sprache (*Input*) wird im *FrontEnd* mit Hilfe von Signalverarbeitungsfiltern analysiert und in *Features* (Merkmalsvektoren) umgewandelt. Mit dem vorliegenden AkustikModell, dem Wörterbuch *(Dictionary*) und dem Sprachmodell erzeugt der *Linguist* einen *SearchGraph*. Dieser *SearchGraph* wird im *Decoder* dazu verwendet, die Features zu evaluieren und Hypothesen über die eingehende Sprache zu erstellen.

5.1.2 SpeechRecog.java

Dieser Abschnitt soll kurz erläutern, wie die Spracherkennung mit dem Sphinx4-System implementiert wurde. Dazu werden die verwendeten Klassen und Funktionen kurz beschrieben (detailliertere Informationen sind in der Sphinx4-Javadoc zu finden).

Als erstes wird durch das Einlesen einer Konfigurationsdatei die Initialisierung des *ConfigurationManager* durchgeführt.

```
ConfigurationManager cm = new ConfigurationManager
      (SpeechToTone.class.getResource
      ("speechtotone.config.xml"));
```

Danach können mit der Funktion *cm.lookup()* die zu verwendenden Module aus dem *ConfigurationManager* geladen und allokiert (Spracherkenner) bzw. gestartet (Mikrofon) werden.

```
Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
recognizer.allocate();
Microphone microphone = (Microphone) cm.lookup("microphone");
microphone.startRecording();
```

Die Spracherkennung selbst wird mit dem Befehl *recognize()* durchgeführt und die Ergebnisse, der erkannten Texte, in einem Objekt vom Typ *Result* abgelegt. Die Methode *getBestFinalResultNoFiller()* liefert anschließend den am wahrscheinlichsten erkannten Sprachbefehl und legt diesen in *resultText*.

```
Result result = recognizer.recognize();
String resultText = result.getBestFinalResultNoFiller();
```

Die Klasse *SpeechRecog.java* fungiert dabei als Schnittstelle zwischen dem Sphinx4-Framework und dem Hauptprogramm (*SpeechToTone.java*). Ihre Grundfunktion besteht darin, den Spracherkennungsprozess einzuleiten (recognize()) und das Ergebnis (resultText) an das Hauptprogramm weiterzugeben(siehe Abbildung 10).

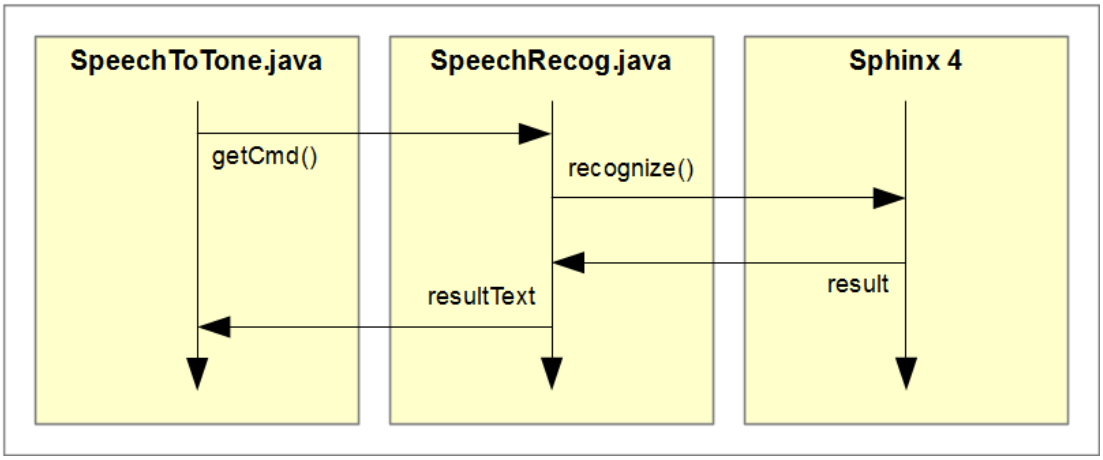

Abbildung 10 Kommunikation Sphinx4 - SpeechToTone

5.1.3 OutListItem.java

Die Klasse OutListItem.java stellt einen Container dar, der vom Hauptprogramm dazu genutzt wird, die bei der Befehlsverarbeitung anfallenden Datensätze strukturiert zwischen zu speichern. Diese abgelegten Daten werden zum Beispiel bei der Korrektur eines falsch erkannten Sprachbefehls oder bei der Formatierung bzw. beim generieren der Ausgabedatei benötigt.

5.1.4 SpeechToTone.java

Das Hauptprogramm ist als die Klasse SpeechToTone.java implementiert. Die Grundfunktionen der Implementierung sollen mit dem in Abbildung 11 dargestellten einfachen Ablaufplan erläutert werden.

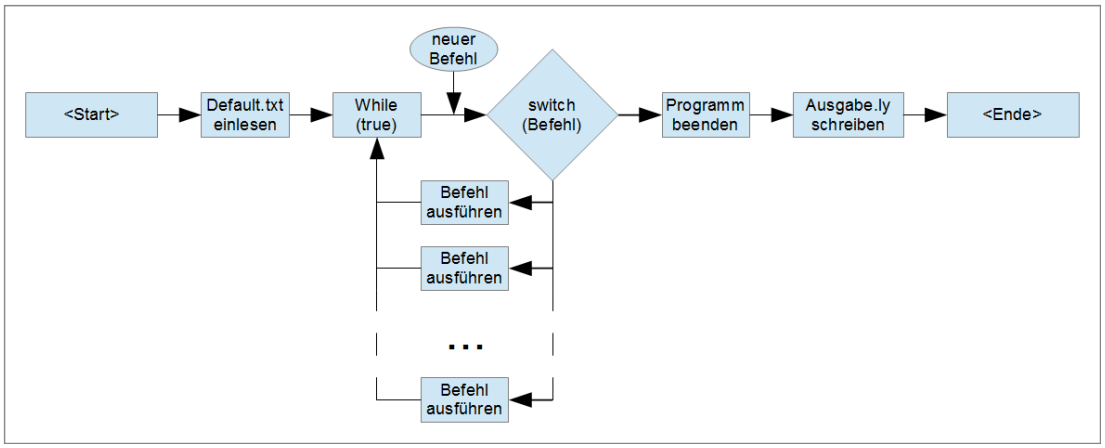

Abbildung 11: vereinfachter Programmablaufplan der Klasse SpeechToTone,java

Nach dem Programmstart wird die übergebene txt-Datei eingelesen und die daraus gewonnenen Informationen in einer Datenstruktur, die durch die Klasse *OutListItem.java* definiert ist, abgelegt. Anschließend wird über eine Funktion der Klasse *SpeechRecog.java* der Spracherkennungsprozess eingeleitet und auf einen Befehl gewartet. Sobald ein neuer Befehl zur Verfügung steht, wird dieser entsprechend seiner Implementierung abgearbeitet und die Ergebnisse in der Datenstruktur gespeichert. Dieser Vorgang wird solange wiederholt, bis der Benutzer mit dem Befehl *"end application"* die Beendung des Programms einleitet. Daraufhin wird die gesamte vom Programm erzeugte Datenstruktur formatiert, in einer .ly-Datei abgelegt und die Anwendung geschlossen.

5.1.5 TextFileIn.java

Die TextFileIn.java kommt beim Testen der Programmlogik zum Einsatz und ist für die normale Arbeitsweise des Programms nicht notwendig. Die Umschaltung in den Testmodus erfolgt im Quellcode durch das Ersetzen der Spracherkennungsschnittstelle mit einem *FileReader*. Eine dem Reader übergebene Datei enthält die vom Programm umzusetzenden Sprachbefehle und kann dazu genutzt werden, reproduzierbare Testszenarien (Kapitel 6.2.) zu erstellen.

5.2 Verwendung des Programmes

5.2.1 Inbetriebnahme

Das Programm liegt als .jar-Datei vor und kann mittels Konsole mit folgendem Befehl ausgeführt werden.

java -jar SpeechToTone.jar default.txt

Die übergebene Datei (hier default.txt) muss sich im Programmordner befinden, da sich das Programm sonst wieder beendet.

5.2.2 Verwendung der Sprachbefehle

Die Beschreibung zur Erzeugung der Sprachbefehle ist nachfolgend in Tabellen dargestellt und anhand von Beispielen erläutert. Die erste Zeile der Tabelle beinhaltet den Namen oder die Funktion des Befehls. Die zweite Zeile beschreibt den Befehlsaufbau anhand von Gruppenbezeichnern. Diese Bezeichner sind in **Dunkelrot** dargestellt. Die dritte Zeile gibt an, wie oft ein Wort ,an dieser Position, in dem Befehl vorkommen kann (* bedeutet: beliebig oft). Alle weiteren Zeilen beinhalten den LilyPond-Ausdruck (in **Grün** dargestellt) und das dazu passende, in natürlicher englischer Sprache notierte, Wort (in **Dunkelblau** dargestellt). Diese sind spaltenweise dem jeweils übergeordneten Gruppenbezeichner zugeordnet. Ein gültiger Sprachbefehl wird erzeugt, indem die blauen Worte, unter Einhaltung der für diese Gruppe geltenden Anzahl, von links nach rechts aneinandergereiht werden.

Eingabe der Noten

Eingabe von Noten

Tonhöhe			Halbton		Oktave		Tonlänge		Punkt		Bogen	
1			0 - 2		*		1		0 - 1		0 -1	
c	c	Charlie	es	es	'	up	1	one	.	point	~	slur
d	d	Delta		flat	,	down	2	two				
e	e	Echo	is	is			4	four				
f	f	Florida		sharp			8	eight				
g	g	Golf					16	sixteen				
b	b	Bravo					32	thirty two				

Beispiele:

Sprachbefehl:	c es up two point	Florida eight	b up up four slur
erzeugter LilyPond-Quellcode:	ces'2.	f8	b''4~

Der \relative - Befehl

\relative - Befehl						
Befehl		Tonhöhe			Oktave	
1		1			*	
\relative	relative	c	c	Charlie	'	up
		d	d	Delta	,	down
		e	e	Echo		
		f	f	Florida		
		g	g	Golf		
		b	b	Bravo		
		Schließen				
		1				
		}			close	

Beispiele:

Sprachbefehl:	relative Florida up	relative close
erzeugter LilyPond-Quellcode:	\relative f' {	}

Wiederholungen von musikalischen Ausdrücken

Wiederholungen

Befehl		Typ		Zahl	
1		1		1	
\repeat	repeat	volta	volt	1	one
		unfold	unfold	2	two
		percent	percent	3	three
		tremolo	tremor	4	four
				5	five
				6	six
				7	seven
				8	eight
				9	nine
		Schließen			
		1			
		}		close	

Beispiele:

Sprachbefehl: **repeat volt two** **repeat close**
erzeugter LilyPond-Quellcode: **\repeat volta 2 {** **}**

Triolen und Taktanpassung

Triolen und Taktanpassung					
Befehl		Zahl		Zahl	
1		1		1	
\tublet	triplet	1	one	1	one
\time	time	2	two	2	two
		3	three	3	three
		4	four	4	four
		5	five	5	five
		6	six	6	six
		7	seven	7	seven
		8	eight	8	eight
		9	nine	9	nine

Beispiele:

Sprachbefehl: **triplet three two + Noten** **time three four**

erzeugter LilyPond-Quellcode: **\tuplet 3/2 { c d e }** **\time 3/4**

Akkorde setzen

Erzeugen von Akkorden					
Befehl		Zahl		Tonlänge	
1		1		1	
\cordmode	chord	1	one	1	one
		2	two	2	two
		3	three	4	four
		4	four	8	eight
		5	five	16	sixteen
		6	six	32	thirty two
		7	seven		
		8	eight		
		9	nine		

Beispiele:

Sprachbefehl: **chord three eight + Noten** **chord three one + Noten**
erzeugter LilyPond-Quellcode: **< c8 d8 e8 >** **< c1 d1 e1 >**

Einfügen aus der Default.txt

Einfügen aus der Default.txt					
Befehl		Zahl		Zahl	
1		1		1	
	file	1	one	1	one
	variable	2	two	2	two
		3	three	3	three
		4	four	4	four
		5	five	5	five
		6	six	6	six
		7	seven	7	seven
		8	eight	8	eight
		9	nine	9	nine

Beispiele: siehe Kapitel 4.2.

Sprachbefehl: **file two two** **variable three four**

erzeugter LilyPond-Quellcode: **Abschnitt 22** **Abschnitt 3 Zeile 4**

Notation eines Auftaktes

Auftakt			
Befehl		Tonlänge	
1		1	
\partial	partial	1	one
		2	two
		4	four
		8	eight
		16	sixteen
		32	thirty two

Beispiele:

Sprachbefehl: **partial eigth** **partial two**
erzeugter LilyPond-Quellcode: **\partial 8** **\partial 2**

Die open/close Befehle

Öffnen und Schließen von Befehlen			
Befehl		Open/Close	
1		1	
\new Book	book		open
\new Score	score		close
\new Staff	staff		
\new Voice	voice		
\alternative	alternative		
\unfold	unfold		
	legato		

Beispiele:

Sprachbefehl: **book open** **legato close**
erzeugter LilyPond-Quellcode: **\new Book {** **)**

Befehle mit einfachen Syntax sind

close application beendet das Programm

new line	fügt eine Leerzeile in den Quellcode ein
break	fügt **\break** in den Quellcode ein
correction	korrigiert letzten verstandenen Befehl

5.2.3 Programmfeatures

In diesem Kapitel wird kurz auf die Programmfunktionen eingegangen, die den Anwender bei der Bedienung unterstützen oder die Lesbarkeit des erzeugten LilyPond-Quellcodes erleichtern.

Da die Anwendung nicht über eine GUI verfügt, muss der Benutzer dennoch nicht auf hilfreiche Informationen verzichten. Diese werden übersichtlich in der Konsole ausgeben, wie folgendes Beispiel zeigt.

```
Formatierungsebenen: [->legato]

You said: g two

-->toneCount:0.0
6letzter Eintrag:   c2          0.5
5letzter Eintrag:   c2          0.5
4letzter Eintrag:   a4          0.25
3letzter Eintrag:   g8          0.125
2letzter Eintrag:   b8          0.125
1letzter Eintrag:   (           0.0
0letzter Eintrag:   g2          0.5
```

Die Formatierungsebene informiert den Benutzer darüber in welchem LilyPond-Befehl er sich gerade befindet. Zum Beispiel wurde im obigen Fall mit dem vorletzten Befehl ein Legatobogen begonnen. Weiterhin wird der letzte erkannte Befehl (You said:) und ein *toneCount* ausgeben. Dieser zählt die Länge der Noten bis der definierte Takt voll ist und erzeugt dadurch einen Zeilenumbruch in der Ausgabedatei. Bei einer Taktanpassung des Musikstückes mit dem Befehl *time* wird der Taktzähler ebenfalls angepasst.

6 Tests

6.1 Test01 Erkennungsrate Buchstaben - Natoalphabet

Testbeschreibung

In diesem Test sollen die Erkennungsraten der Implementierungen für die Tonhöhe ermittelt werden. Die Tonhöhe kann in natürlicher Sprache zum einen als Buchstabe (c-d-e-f-g-a-b) und zum anderen im Natoalphabet (charlie-delta-echo-florida-golf-alfa-bravo) angegeben werden.

Testaufbau und Durchführung

Als Erstes wird eine Datei (Test01-Default.txt) erstellt, die den Musiktakt für diesen Test auf 75/8 setzt. Dies dient der allein der Übersichtlichkeit. Danach wird das Programm mit *java -jar SpeechToTone.jar Test01-Default.txt* ausgeführt. Anschließend erfolgt die gesprochene Eingabe der Tonhöhen nach folgenden Schema.

fünf mal: <Tonhöhe> one <Tonhöhe> two <Tonhöhe> four <Tonhöhe> eight

Der Bezeichner <Tonhöhe> steht dabei für die einzelnen Buchstaben bzw. Wörter des Natoalphbetes. Das Ergebnis dieser Eingabeprozedur ist in der vom Programm erzeugten .ly-Datei zu finden. Auszug:

```
c1 c2 e4 c8 c1 c2 c4 c8 c1 c2 c4 c8 c1 c2 c4 c8 c1 c2 c4 c8
b1 d2 d4 e8 d1 d2 d4 g8 d1 d2 d4 e8 d1 d2 d4 g8 d1 d2 d4 g8
e1 e2 e4 e8 e1 e2 e4 e8 e1 b2 e4 e8 e1 e2 e4 e8 e1 e2 e4 e8
a1 f2 a4 f8 f1 f2 f4 f8 f1 f2 f4 f8 f1 f2 f4 f8 f1 f2 f4 f8
g1 g2 g4 g8 g1 c2 g4 g8 g1 c2 g4 g8 g1 d2 d4 g8 g1 g2 g4 g8
a1 a2 a4 a8 a1 a2 a4 a8 a1 a2 a4 a8 a1 a2 a4 a8 a1 a2 a4 a8
b1 e2 b4 e8 b1 d2 b4 b8 b1 e2 b4 b8 e1 b2 b4 d8 b1 e2 e4 b8
```

Die Auswertung dieser Datei erfolgt durch zählen der falsch erkannten Tonhöhen.

Testergebnis

Der oben beschriebene Test wurde insgesamt fünf mal durchgeführt um 100 Werte für jede Tonhöhe zu erhalten. Die Ergebnisse sind in folgender Tabelle dargestellt.

Buchstaben	Erkennungsrate in %	Natoalphabet	Erkennungsrate in %
C	91	Charlie	99
D	73	Delta	94
E	94	Echo	99
F	89	Florida	98
G	82	Golf	96
A	98	Alfa	99
B	66	Bravo	97

Die Testergebnisse zeigen durch eine deutliche Steigerung der Erkennungsrate, dass das in Kapitel 3.2.3. Homophonenproblem, durch die Implementierung des Natoalphabetes, gelöst werden konnte.

6.2 Test02 Programmfunktionstest

Testbeschreibung

In diesem Test soll die Programmlogik untersucht werden. Dazu wird das Programm von der Spracherkennungssoftware entkoppelt und die Erkennung der Sprachbefehle mit einer Datei (Test02-dieNacht-eingabe.txt) realisiert. Dieser Test soll überprüfen ob die, aus den Dateien "dieNacht.ly" und "Test02-dieNacht-ausgabe.ly", von LilyPond erstellten Notenblätter Unterschiede aufweisen.

Testaufbau und Durchführung

Die Datei "Test02-dieNacht-eingabe.txt" wird auf Grundlage der Datei "dieNacht.ly" erstellt und enthält die zur Steuerung des Programme nötigen Sprachbefehle, welche die Datei "dieNacht.ly" abbilden. Auszug "Test02-dieNacht-eingabe.txt":

```
variable three
relative c up up
repeat volt two
g is four
legato open
d four
correction
e four
```

Nach erfolgreichem Programmdurchlauf werden mit Lilypond die Notenblätter für die Dateien "dieNacht.ly" und "Test02-dieNacht-ausgabe.ly" erstellt.

Testergebnis

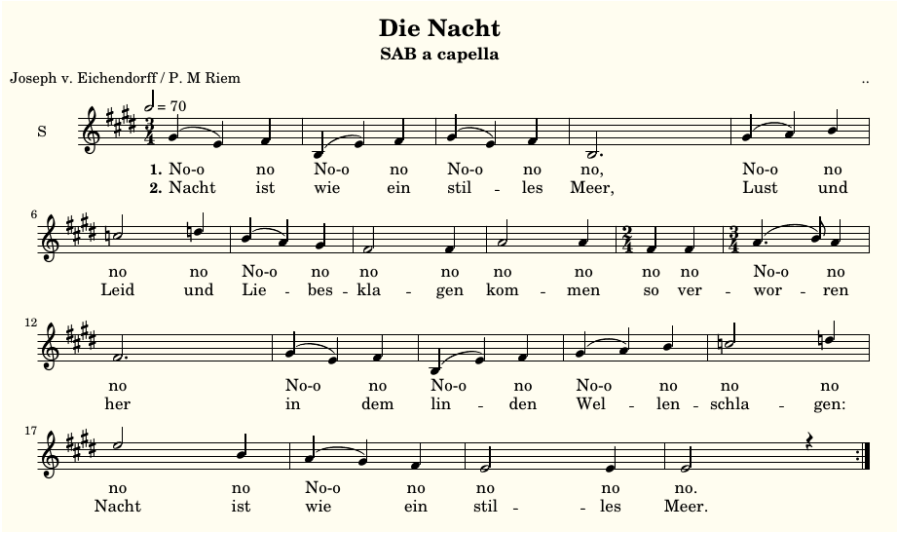

Abbildung 12: dieNacht

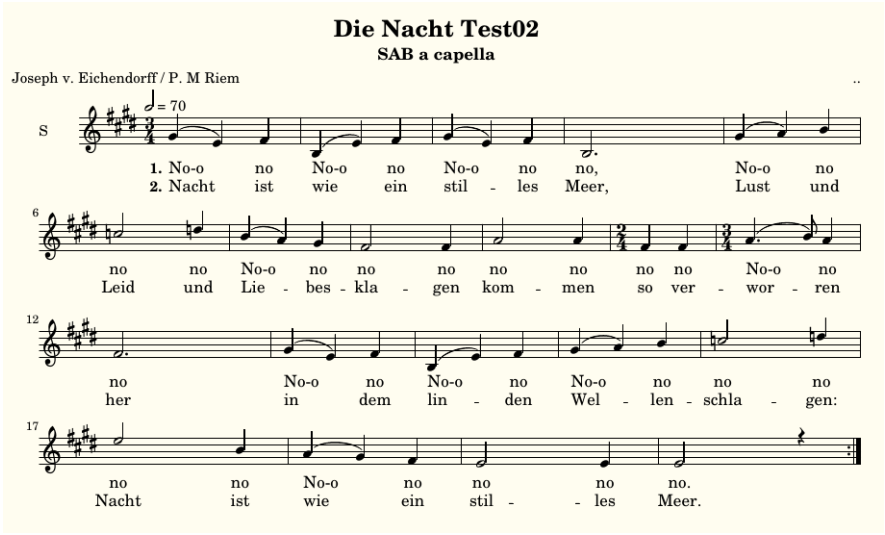

Abbildung 13: dieNacht Test02

Die korrekte Funktion der Programmlogik kann mit den oben dargestellten Notenblättern bestätigt werden, da sich diese bis auf den Titel gleichen.

6.3 Test03 Erkennungsrate dieNacht.ly

Testbeschreibung

Dieser Test soll die allgemeine Erkennungsrate der Befehle ermitteln. Dazu wird die Datei "dieNacht.ly" in natürlichen Sprachbefehlen erzeugt und die dabei nötigen Sprachkorrekturen mitgezählt. Anschließend wird aus dem Verhältnis von Korrekturen und der Anzahl gesprochener Befehle die Erkennungsrate bestimmt.

Testergebnis

Der Test wurde zweimal durchgeführt. Im ersten Durchlauf musste sieben und im Zweiten neun mal korrigiert werden. Jeder Testlauf umfasst 70 Sprachbefehle.

Erkennungsrate: 1 - ((7+9)/(70*2)) = 1 - (16/140) = 1 - 0,114 = 0,886

Die Auswertung der Tests liefert eine Befehlserkennungsrate von 88,6 Prozent.

7 Zusammenfassung

Das Ziel diese Arbeit war die Entwicklung einer Anwendung die es ermöglicht mit einfachen Sprachbefehlen eine Textdatei zu erstellen, die anschließend von dem textbasierenden Notensatzprogramm LilyPond genutzt wird um daraus einen Notensatz zu erzeugen. Um diese Ziel zu erreichen, wurde zunächst das Thema der Spracherkennung genauer beleuchtet. Indem die Grundlagen der Spracherkennung erläutert und Spracherkennungs-software vorgestellt wurden. Im weiteren Verlauf der Arbeit konnten auch Probleme der Spracherkennung dargestellt und Lösungsansätze angeboten werden. Der Syntax des vorgestellten Notensatzprogrammes LilyPond bildete die Grundlage zum erstellen einer geeigneten Befehlssprache. Durch die Analyse der LilyPond-Struktur und die Bewertung der LilyPond-Ausdrücke konnte entschieden werden, welches die zu implementierenden Befehle sind. Aus diesen Befehlen wurde unter Verwendung einer einfachen Grammatik eine Befehlssprache erschaffen. Da diese Sprache nicht ausreicht um alle LilyPond-Befehle abzudecken wurde eine Dateistruktur entwickelt, mit der sich der Befehlspool indirekt erweitern lässt. Nach der Implementierung und Vorstellung des Programms wurde die Befehlssprache ausführlich behandelt. Bei den abschließenden Tests konnte gezeigt werden, dass sich das Homophonenproblem mit Hilfe des Natoalphabetes lösen lässt, die Programmlogik fehlerfrei arbeitet und eine Befehlserkennungsrate von 88 Prozent ist.

Literaturverzeichnis

[Car12] Kai-Uwe Carstensen
Sprachtechnologie Ein Überblick, 2012
[Wöl09] Matthias Wölfel, John McDonough
Distant Speech Recognition, 2009
[Wal04] : Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj,
Rita Singh, Evandro Gouvea, Peter Wolf, Joe Woelfel
Sphinx-4: A Flexible Open Source Framework for Speech Recognition, 2004
[Rab93] Lawrence Rabiner, Biing-Hwang Juang
Fundamentals of Speech Recognition, 1993
[Yan11] HJ. Yang, C. Oehlke, Ch. Meinel,
German Speech Recognition: A Solution for the Analysis and Processing of Lecture Recordings, 2011
[LEE10] Akinobu LEE
The Julius Book, 2010
[Gra10] Peter H. Grasch
Das simon Handbuch, 2010
[JSGF] JSGFGrammar URL: 25.07.2015
http://cmusphinx.sourceforge.net/doc/sphinx4/edu/cmu/sphinx/jsgf/JSGFGrammar.html
[HTK] Documentation for HTK URL:17.04.2015
http://htk.eng.cam.ac.uk/
[Lily] Handbücher für LilyPond 2.18.2 URL:13.09.2015
http://www.lilypond.org/manuals.de.html
[Fres] Frescobaldi Manual URL 26.09.2015
http://frescobaldi.org/uguide
[CoGr] commandGrammer.jpg, Bild URL 26.09.2015
http://cmusphinx.sourceforge.net/doc/sphinx4/edu/cmu/sphinx/jsgf/doc-files/commandGrammar.jpg
[Spek] specktrumsanalyse, Bild URL 13.08.2015
http://medi.uni-oldenburg.de/download/docs/lehre/kollm_audiologie/audiol6.pdf
[FresA] frescobaldi1-en, Bild URL 26.09.2015
http://frescobaldi.org/images/frescobaldi1-en.png
[FresB] uguide_getting_started1, Bild URL 26.09.2015
http://frescobaldi.org/uguide
[LilyB] bach-bwv610, Bild URL 26.09.2015
http://www.lilypond.org/text-input.de.html
[VoSp] VorgehensweiseSpracherkennung, Bild URL: 17.06.2015
httpwiki.infowiss.netimages-infowisswikieebVorgehensweiseSpracherkennung.jpg

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____