Institut für
Technische Informatik und
Kommunikationsnetze

Sprachsteuerung eines Audiogerätes

Semesterarbeit SA-2004-25
Sommersemester 2004

Simon Haegler
Andreas Ulmer

Institut für Technische Informatik
und Kommunikationsnetze

Betreuer: H. Romsdorfer
Verantwortlicher: Prof. Dr. L. Thiele

ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo

Zusammenfassung

Heutige harddisk-basierte MP3-Player besitzen enorme Speicherkapazitäten (bis zu 120Gb) was schnell zu einer grossen Menge Dateien führt (mehrere 10k) die aufgrund der Nachfrage nach kleineren Geräten mit immer weniger Bedienelementen verwaltet werden müssen. Wir wollen durch Einsatz von Sprachsteuerung untersuchen, inwiefern sich damit die Bedienung der Geräte verbessern lässt. In diesem Bericht wird die Konzeption und Implementation eines *Sprachgesteuerten MP3-Players* auf Softwarebasis (Matlab) beschrieben. Schwerpunkte bilden dabei die Konzeption der Dialogsteuerung und die Bedienung, mit dem Ziel die Navigation in einer grossen Anzahl Dateien zu vereinfachen.

Untersuchungen an bestehenden MP3-Playern zeigten deren Vor- und Nachteile auf und inspirierten die Implementierung einer sprachgesteuerten Variante. Es wurde ein Prototyp entwickelt der die notwendige Funktionalität beherrscht und als flexible und einfach ausbaubare Testumgebung für die Untersuchung des Bedienkomforts dient. Erste Ergebnisse zeigen, dass eine alleinige Steuerung mittels Sprache nur bedingt sinnvoll ist. Anhand der implementierten sprachgesteuerten Suchfunktion und benutzerdefinierbaren Sprachmustern, die als Shortcuts auf Wiedergabelisten dienen, wurde die Sprachsteuerung jedoch als Bereicherung der Bedienung erkannt.

Inhaltsverzeichnis

1 Einleitung

1.1 Idee und Motivation

Aktuelle, Festplatten-basierte MP3-Player bieten Speicherkapazitäten bis zu 120Gb. Als typische Vertreter gehören hierzu der Apple iPod (Abbildung 2) oder der Archos Gmini220 (Abbildung 1). Daneben gibt es auch USB-Sticks mit eingebauten MP3-Playern, welche Speicherplatz bis 1.5Gb besitzen. Abbildung 3 zeigt als Beispiel den MSI MegaStick. Diese grossen Speicherkapazitäten führen erfahrungsgemäss zu grossen Dateiansammlungen mit bis zu mehreren zehntausend Files.

Figur 1: *Archos Gmini220*

Figur 2: *Apple iPod*

Allen Festplatten-basierten Geräten gemeinsam ist die Navigation über eine Art Cursorsteuerung. Archos hat hierfür einen Kippschalter gewählt, welchen man in alle vier

Figur 3: *MSI MegaStick 256Mb*

Richtungen betätigen kann. Bei Apple dient eine berührungsempfindliche Glasplatte als Cursorsteuerung.

Beide Geräte unterscheiden sich auch in der Art der Dateiorganisation. So ist bei Apple kein direkter Zugriff auf die Files möglich, die Steuerung geschieht hier über sogenannte ID3-Tags welche aus den MP3-Files herausgelesen werden. Siehe dazu Kapitel 4.5.

Der Archos bietet im Gegensatz dazu beide Arten der Navigation an, es kann also auch direkt auf die Dateien auf der Festplatte zugegriffen werden, siehe Abbildung 4.

Nichtsdestotrotz ist die Navigation durch diese grossen Datenmengen mittels Cursortasten mühsam, liegt ein Titel beispielsweise irgendwo in der Mitte der Liste so muss ziemliche lange hinunter- bzw. hinaufgescrollt werden. Beim Gmini220 ist auch gut die Auswirkung der Miniaturisierung ersichtlich: entweder bleibt immer weniger Platz für die Bedienelemente oder deren Grösse konvergiert in Richung Unbedienbarkeit. Daher haben wir uns die Frage gestellt:

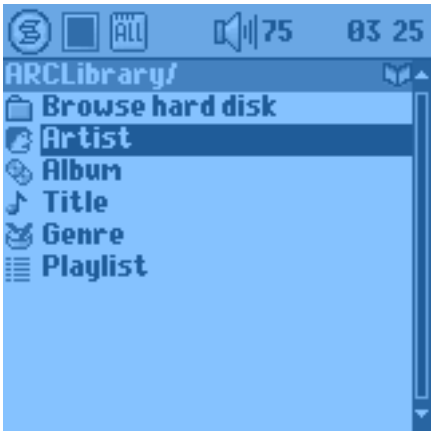
Warum also nicht die bereits vorhandene Audio-Ein- und -Ausgabe zur Steuerung verwenden?

So lässt sich beispielsweise das Problem der Miniaturisierung mit Sprachsteuerung elegant kompensieren, da die Sprache beliebig viele Bedienelemente ersetzen kann. Das akustische Feedback kann ein Blick aufs Display unnötig machen, dies ist praktisch falls sich der Player selber gerade ausserhalb der Sichtweite befindet (zB. in der Tasche).

1.2 Roadmap

Zur Beginn dieser Semesterarbeit haben wir folgende Roadmap zur Bearbeitung vorgeschlagen:

1. Konzeption des Gesamtsystems
2. Evaluation von Spracherkennungs- und synthese-Alternativen



Figur 4: *Der Archos Gmini220 bietet Tag- sowie File-basierte Navigation.*

3. Konzeption von Befehlssätzen & Sprachausgaben für eine intelligente Navigation
4. Erste experimentelle Implementation in Matlab
5. *Implementation eines PlugIns für Software-Player (zB. XMMS)*
6. *Realisierung in Hardwareform mit Interface zu Hardware-Player*

Als Ziel formulierten wir den Abschluss der Punkte 1 bis 4. Die letzten beiden Listenpunkte sollten eine Idee für das Fernziel der Arbeit geben und zu weiterführender Arbeit motivieren.

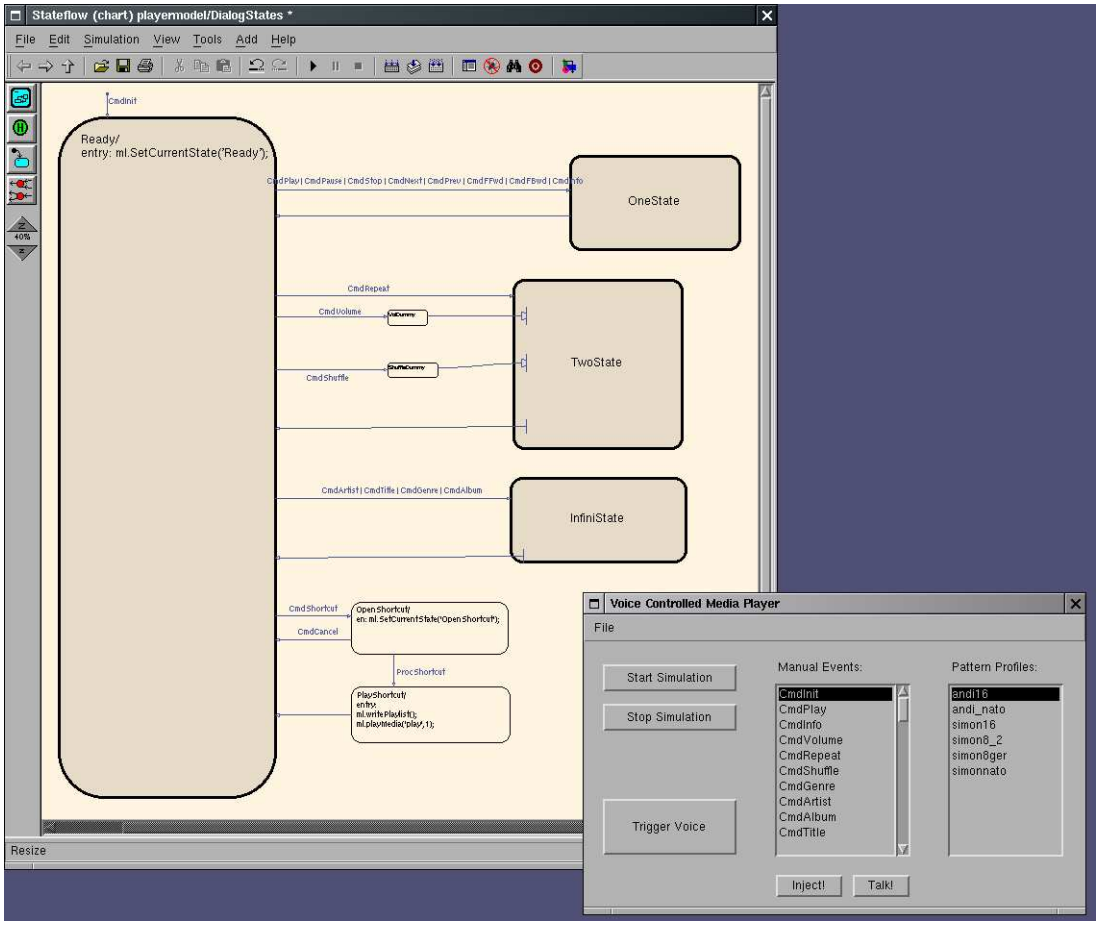
1.3 Arbeits- und Zeitplanung

Um die zu Verfügung stehende Zeit gut nutzen zu können, haben wir das Projekt in die folgenden Tasks unterteilt:

	Woche	Task
1	16	Einarbeiten
2	17	Featureanalyse und Review SPV Übung 13
3	18	Identifier, Befehlssatz, Dialogsteuerung
4	ab 19	Implementation Prototyp
5	23	Vergleiche
6	24	Vorbereiten Präsentation
7	25	Präsentation
8	26-27	Report fertigstellen und Abgabe

1.4 Über dieses Dokument

Dieser Bericht setzt sich aus drei Hauptbereichen zusammen: Evaluierung, Konzept und Implementation. In der Evaluierung haben wir uns mit erhältlichen MP3-Playern



Figur 5: *Das Steuerfenster von VoCoMP mit Stateflow-Ansicht der Dialogsteuerung*

auseinandergesetzt und die Bedienstruktur analysiert. Ziel war Bedienungsablauf und Funktionsumfang auszuloten um daraus eine Sammlung an möglichen Zielfunktionen unseres Systems zu erhalten. Diese Sammlung haben wir mit eigenen Ideen ergänzt.

Im Konzept haben wir die Komponenten des Systems skizziert und den Funktionsumfang des Gerätes definiert, bevor im Kapitel Implementation schliesslich die konkrete Ausarbeitung in Matlab und Stateflow behandelt werden.

1.5 Kurzanleitung zu VoCoMP

Im Folgenden soll eine kurze Anleitung zu VoCoMP gegeben werden. In Abbildung 5 ist dargestellt wie sich VoCoMP nach dem Start präsentiert. Rechts unten befindet das Hauptfenster von VoCoMP, dort wird die Emulation gestartet und Spracheingaben getriggert. Falls es Probleme mit der Spracherkennung gibt, kann dort ein Befehl auch manuell ausgelöst werden.

1.5.1 Aufnehmen der Sprachmuster

Für eine einigermassen gute Mustererkennung, ist es nötig, dass jeder Benutzer eigene Muster anlegt, und zwar mit denselben Bedingungen die nachher auch bei der Benutzung vorhanden sind (gleiche Raumakustik, gleiche Hintergrundgeräusche, gleiches Mikrofon).

In Tabelle 5 sind die nötigen Befehle aufgelistet, für welche ein Sprachmuster vorhanden sein muss (inkl. Ziffern 0 bis 9 und A bis Z). Als erstes muss in `~/src/detector/signals` ein Unterverzeichnis erstellt werden. Der Name dieses Verzeichnisses wird dann in der Patternliste aufgelistet (siehe Abschnitt 4.3.4).

Die Dateien sollen das SUN Audioformat besitzen (`.au`) und mit einer Wortbreite von 16bit (linear) und 8kHz Samplingrate aufgenommen werden. Der Dateiname soll jeweils mit dem entsprechenden Befehl übereinstimmen, zB. `ArgOne.au`, `CmdPlay.au`, `ArgW.au`, etc.

Der Befehl zur Aufnahme sieht beispielsweise so aus (siehe auch `man audiorecord`):

```
audiorecord -s 8k -p mic -e linear -v 30 CmdShuffle.au
```

1.5.2 Starten von VoCoMP

Wichtig ist, dass Matlab im `~/src` Verzeichnis gestartet wird und zwar ohne Desktop, da dies enorm auf die Performance drückt:

```
[spr2:src]$ matlab -nodesktop
```

Nun kann das System mit dem Hauptfile gestartet werden:

```
» vocomp(1)
```

Das Argument entscheidet darüber, ob der Sprachsynthese-Server SVOX gestartet werden soll (1) oder nicht (0). Beim Aufstarten gibt VoCoMP über Statusmeldungen Auskunft über etwaige Probleme. Wurde die Synthese aktiviert, dann sollte die Bereitschaft auch akustisch angekündigt werden.

Bevor die Simulation gestartet werden kann, muss noch die Zustandsmaschine mit dem entsprechenden Befehl aus dem File-Menü gescannt werden, damit werden die internen Datenstrukturen erstellt (siehe Abschnitt 4.3.2).

1.5.3 Bedienung des Hauptfensters

Nachdem die Zustandsmaschine gescannt wurde, kann die Simulation mit dem entsprechenden Button gestartet werden, es werden jeweils Statusmeldungen auf der Matlab-Konsole ausgegeben. Mit dem Trigger-Button kann die Sprachaufnahme gestartet und gestoppt werden, auf der Matlab-Konsole kann nach dem Stoppen die Berechnung der kleinsten Gesamtdistanz verfolgt werden. Weitere Informationen zur Bedienung befinden sich in Abschnitt 4.3.4.

2 Evaluierung

2.1 Analyse von erhältlichen MP3-Playern

Im Umfeld der Authoren konnten vier aktuelle harddiskbasierte MP3-Player organisiert und untersucht werden, die in Tabelle 1 aufgelistet sind.

Typ	Hersteller	Link
iHP140	iRiver	www.iriver.com/product/info.asp?p_name=H120
iPod	Apple	www.apple.com/chde/ipod/
Jukebox Recorder	Archos	www.archos.com/download/spec_sheets.html
NOMAD Jukebox Zen	Creative	www.creative.com/products/product.asp?prodid=9019

Tabelle 1: *Liste der analysierten MP3-Player*

Um gute Lösungsansätze betreffend Menüsteuerung zu übernehmen und Problembereiche zu erkennen, wurde für die Auswertung die folgende Fragestellung formuliert:

1. Welchen Funktionsumfang bieten Geräte dieser Art dem Benutzer?
2. Sind die MP3-Daten sortiert bzw. sortierbar? Wenn ja, nach welchen Kriterien / Kategorien?
3. Wie unterscheiden sich Bedienabläufe im Menü für den Benutzer?
4. Wo liegen die Schwachstellen in der Bedienung? Welche Funktionen sind nur mühsam und/oder zeitraubend zu bedienen?

Von jedem der untersuchten Player wurde ein symbolischer Menüablauf skizziert, der Abläufe im Menü aufzeigen und die Beantwortung der Fragen vereinfachen sollte. Diese Stateflowcharts sind im Anhang A zu finden.

Funktionsumfang Die Frage nach dem Funktionsumfang wird nachfolgend in 2.2 detalliert erläutert.

Datenstruktur Ausser dem Archos Jukebox Recorder legen alle Player intern eine weitere, von der Ordnerstruktur unabhängige Datenbank an, die die MP3-Sammlung nach den wichtigsten Kategorien Genre, Artist, Album und Titel sortiert. Die Information dazu wird aus den im MP3-File mitabgespeicherten ID3-Tags[1] gewonnen. Voraussetzung dazu ist allerdings, dass die ID3-Tags vorhanden sind sowie korrekt und einheitlich generiert wurden. Andererseits ist eine ungenügend organisierte Ordnerstruktur im alternativen System einer unbeschwerten Navigation ebenso abkömmlich. Der iPod verlässt sich ausschliesslich auf die Indizierung mittels Kategorien, der iHP140 und der Nomad bieten kombiniert sowohl Ordner- als auch Kategorienstruktur an.

[1] http://www.id3.org

Bedienabläufe Unterschiedliche Bedienungsabläufe finden sich ausser bei der Dateinavigation nur bedingt. Die häufigsten Funktionen fürs Abspielen und die Lautstärkeregelung sind unkompliziert zu bedienen und erfordern keine langwierige Navigation durchs Menü. Unterschiede, die das Suchen von Dateien betreffen sind hauptsächlich durch die verschiedenen Arten der Sortierung bedingt. So kann in einer Ordnerstruktur beliebig viele Male in ein Unterverzeichnis abgestiegen werden, während dies bei der Kategoriesuche auf die Anzahl Kategorien beschränkt ist. Die Auswahl von Suchkategorien ist logischerweise bei ordnerbasierten Systemen obsolet, was den Suchablauf grundsätzlich verkürzt und die Menüsteuerung vereinfacht. Dies auf Kosten der Option, ordnerübergreifende Wiedergabelisten erstellen und abspielen zu können.

Schwachstellen Dieser letzte Absatz beantwortet auch schon Punkt 4 der Fragestellung: Den weitaus kompliziertesten Teil, der aber auch am unbefriedigsten gelöst ist, stellt sicherlich die Auswahl von bzw. die Suche nach MP3-Dateien dar. Trotz teilweise intelligenter Menüführung und komfortabler Bedienelemente (z.B. Scrollräder beim iPod und Nomad) ist der Vorgang je nach Aufgabe langwierig. Dazu kommt, dass gewisse Geräte zwar eine Fernbedienung besitzen, der Funktionsumfang auf dieser jedoch aufgrund der geringen Menge an Bedienelementen stark eingeschränkt ist. Der iHP140, dessen Fernbedienung den vollen Funktionsumfang enthält und sogar ein funktionell dem Gerät identisches Display besitzt, bildet hier die Ausnahme. So ist eine Neuauswahl an Titeln bei drei der vier untersuchten Player ohne direkten physikalischen und visuellen Zugriff auf das Hauptgerät unmöglich.

2.2 Beschreibung der Benutzerfunktionen

Die folgenden Abschnitte fassen unsere Beobachtungen bezüglich Funktionsumfang bei der Analyse der Geräte im vorigen Abschnitt zusammen, zusätzlich werden sie durch eigene Ideen ergänzt.

Player Controls Die Player-Controls beinhalten primär Steuerungselemente die man von üblichen Musikwiedergabegeräten kennt: Grundlegende Funktionen wie `Play`, `Stop`, Funktionen zur Steuerung des Ablaufs (`Next/Prev Title`) oder `ffwd/rew` für das verschieben des Abspielcursors innerhalb eines Songs. Dazu kommen verschiedene Abspielmodi, die das Wiederholen von Titeln und Wiedergabelisten (`Repeat`) oder zufällige Abspielreihenfolge (`Shuffle`) erlauben. Die letzten beiden Funktionen beinhalten meist einige zusätzliche Argumente, die den Rahmen, in dem der Modus operiert, beispielsweise ein Album oder ein Verzeichnis, festlegen.

Sound Controls Die Sound Controls beinhalten die Lautstärke- und Equalizerregelung. Darunter finden sich sowohl manuelle als auch adaptive Regelungen, die die Gesamtlautstärke konstant halten. Die manuelle Steuerung der Lautstärke ist durchgehend inkrementell gelöst, das heisst, ein Befehl erhöht respektive verringert die momentane Lautstärke um einen gewissen Minimalwert. Als Alternative kommt eine absolute Steuerung in Frage, die ein schnelleres Anwählen einer bestimmten Lautstärke

erlaubt. Für das Anpassen des Sounds an die eigenen Bedürfnisse existieren separate Höhen- und Bassregelung sowie vordefinierte Presets (Rock, Vocal, Stadion usw.).

Info-Ausgabe Der Benutzer wird auf expliziten Wunsch oder implizit über den aktuellen Status informiert. Im Abspielmodus beinhaltet diese Ausgabe vor allem Informationen über den laufenden Titel, also Angaben über Länge, Bitrate oder ID3-Tags. Auf den hochauflösenden Displays der untersuchten Player wird gleichzeitig eine grosse Menge Information dieser Art dargestellt. Während der Navigation wird der User informiert über seinen momentanen "Standort" im Menüablauf und die kontextabhängig zur Verfügung stehenden Optionen.

Dateinavigation Die Dateinavigation ist bei allen Playern relativ ähnlich. Man browst innerhalb Kategorien oder Verzeichnisse durch alphabetisch sortierte Einträge, bis die gewünschte Auswahl, ein Titel, ein Album oder ein Artist, gefunden ist. Der zeitliche Aufwand dazu steht in Relation mit der Grösse der MP3-Sammlung, der Geschwindigkiet der Scrollfunktion sowie der Art der Sortierung der Dateien. Einer der untersuchten Player (Nomad) besitzt zusätzlich eine Suchfunktion, die jedoch ausser Komplexität nichts zur Bedienung hinzufügt.

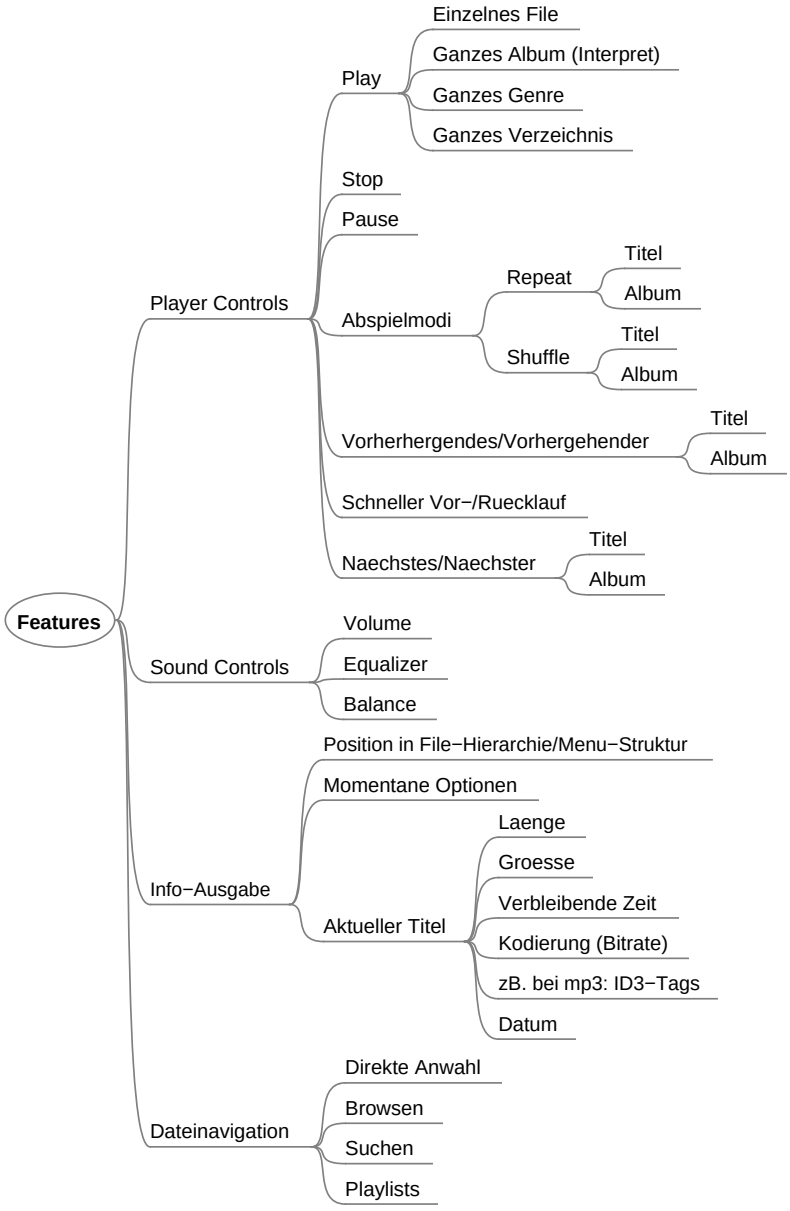
Playlists Alle Player unterstützen Wiedergabelisten, wobei diese entweder auf dem PC oder sogar auf dem Gerät (iPod, iHP140 und Archos) erstellt werden. Der Liste können beliebig Titel und ganze Alben hinzugefügt werden.

Personalisierung Die Geräte bieten nur sehr geringe Konfigurierbarkeit, der Benutzer muss sich mit den vordefinierten Einstellungen abfinden. Einzig der Nomad bietet eine, wenn auch sehr limitierte Form von Shortcuts.

2.3 Spracherkennung

Erkennertypen Als mögliche Erkennertypen stehen Einzelworterkenner, Keyword-Spotter und kontinuierliche Spracherkenner zur Diskussion. Befehle mit (optionalen) Argumenten werden dem Einzelwort-Erkenner zugeordnet. Bei einem Einzelwort-Erkenner muss sich der Sprecher stark den Anforderungen des Erkenners anpassen (abgehacktes Sprechen), wogegen ein kontinuierlicher Spracherkenner vergleichsweise komplexe Ansprüche an Implementierung und Training stellt, dafür aus Benutzersicht am bequemsten ist.

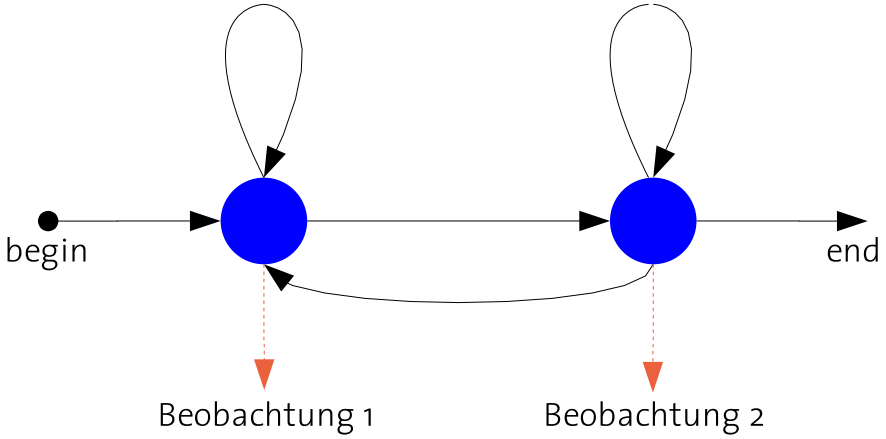
Die zwei am häufigsten verwendeten Systeme zur Spracherkennung sind die Hidden-Markov-Modelle (HMM) und der Mustervergleich (DTW). Tabelle 2 liefert einen Überblick. Beide Verfahren beruhen auf Merkmalssequenzen die aus dem eigentlichen Sprachsignal berechnet werden, es folgt deshalb ein Überblick über mögliche Merkmalsarten.



Figur 6: *Features der analysierten MP3-Player*

Hidden Markov Model		Dynamic Time Warping	
+	Offene Erkennungsmenge	-	Erkennt nur vordefinierte Muster
+	Hohe Erkennungsrate	-	Schlechte Unterscheidung ähnlich klingender Wörter
-	Grosser Trainingsaufwand	+	Nur Aufnahme von Mustern nötig
-	Aufwändige Implementierung	+	Einfache, robuste Implementierung

Tabelle 2: *Vergleich zwischen Hidden-Markov-Modellen (HMM) und Mustervergleich (DTW)*



Figur 7: *Allgemeines Modell eines HMM*

Merkmalsextraktion Das aufgenommene Sprachsignal wird zunächst in kurze Abschnitte unterteilt (gefenstert). Auf diese Abschnitte wird dann ein homomorphes System für die Faltung angewendet und als Resultat erhält man das *DFT-Cepstrum*, welches eine erste Variante für die Merkmalsextraktion darstellt. Allerdings muss hier der übliche Kompromiss zwischen zeitlicher und spektraler Auflösung gefunden werden (Fouriertheorem). Eine bessere und oft verwendete Möglichkeit ist das *LPC-Cepstrum*, welches eine cepstrale Glättung bewirkt. Der Vergleich von LPC-Cepstren wird so robuster gegenüber kleinen spektralen Verschiebungen der Formanten.

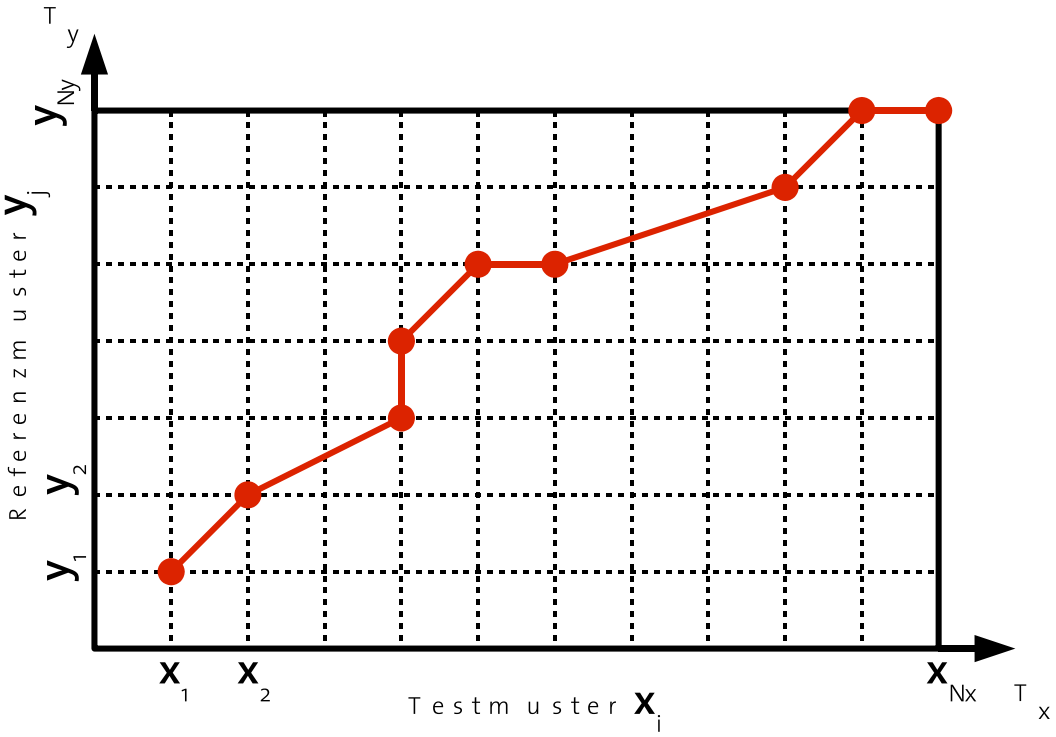
Eine weitere Möglichkeit die Merkmale zu extrahieren bietet das *Mel-Cepstrum*, welches nicht über das Fourierspektrum berechnet wird, sondern über eine Filterbank, die aus psychoakustischen Erkenntnissen aufgebaut ist.

Als weitere Merkmale werden auch Ableitungen der cepstralen Koeffizienten verwendet, das sogenannte *Delta-Cepstrum*. Für weitere Informationen sei auf das Skript *Sprachverarbeitung I* Abschnitt 4.6 verwiesen.

Hidden-Markov-Modelle Hidden-Markov-Modelle bieten einen statistischen Ansatz zur Spracherkennung und eignen sich sowohl für Einzelwort- wie auch für kontinuierliche Spracherkennung. Für jedes Wort wird ein Netzwerk mit einer bestimmten Anzahl Zuständen, Übergangs- und Beobachungswahrscheinlichkeiten erstellt, welche dann wiederum zu Netzwerken für die Satzerkennung zusammengeknüpft werden können.

Bevor ein HMM-Erkenner (siehe Abbildung 7) einsatzbereit ist, müssen die Wahrscheinlichkeits-Parameter trainiert werden, jeder Pfeil in der Abbildung steht jeweils für eine Übergangswahrscheinlichkeit. Nach einem erfolgten Initialtraining für den Wortschatz muss das Modell noch benutzerspezifisch nachtrainiert werden. Man erkauft sich also die relativ hohe Erkennungsrate durch einen grossen Trainingsaufwand. Schätzungen ergeben, dass etwa 10 Beobachtungen (Trainingssätze) pro freiem Parameter benötigt werden.

Um einen HMM-Erkenner trainieren zu können, müssen aus dem Trainingsmate-



Figur 8: *Warping-Kurve für den Vergleich zweier Sprachmuster*

rial zuerst Merkmalsvektoren gewonnen werden. Ist der HMM diskret aufgebaut (Discrete-Density-HMM, DDHMM), so müssen diese Merkmalsvektoren noch vektorquantisiert werden. Im Gegensatz dazu können für CDHMM (Continous-Density-HMM) direkt die Merkmale verwendet werden, da mit kontiuierlichen Wahrscheinlichkeitsdichten (Mischkomponenten) gearbeitet wird[2].

Sprachmustervergleich Der Mustervergleich ist keine statistische Methode, sondern basiert auf der direkten Berechnung von euklidischen Abständen zwischen Merkmalsvektoren von Signalabschnitten. Die Merkmalsvektoren setzen sich meistens aus der Signalenergie und den ersten zwölf bis sechzehn Quefrenzen[3] des betreffenden Signalabschnitts zusammen. Dadurch ist der Mustervergleich eine sehr schnelle Methode, die allerdings Mühe hat mit ähnlich klingenden Wörtern. Wegen seiner einfachen Implementation wird der Mustervergleich beispielsweise in Mobiltelefonen zur sprachgesteuerten Direktanwahl eingesetzt.

In Abbildung 8 ist eine Warping-Kurve dargestellt, wie sie beim Vergleich zweier Sprachmuster entsteht. Sie beschreibt wie die einzelnen Abschnitte relativ zueinander dynamisch verschoben werden. Dabei lässt sich für jeden diskreten Zeitpunkt k ein Paar von Indices $i(k)$ und $j(k)$ angeben, für welche die jeweiligen Merkmalsvektoren $x(i)$ und $y(j)$ eine minimale lokale Distanz besitzen. Die optimale Warping-Kurve entspricht nun dem Pfad, welcher am Ende eine minimale Gesamtdistanz besitzt.

[2] siehe Skript *Sprachverarbeitung II* Kapitel 9.2
[3] Bezeichnung der Koeffizienten des Cepstrums

Als Distanzmass wird häufig das Integral der logarithmisch-spektralen Differenz zwischen den Mustern X und Y verwendet:

$$d = \left(\frac{1}{2\pi} \int_{-\pi}^{\pi} |\log X(\omega) - \log Y(\omega)|^p d\omega \right)^{\frac{1}{p}}$$

Durch vereinfachende Annahmen kann diese Distanz auch mit den entsprechenden cepstralen Koeffizienten $c(m)$ berechnet werden:

$$d^2 = \sum_{m=-\infty}^{\infty} m^2 (c_X(m) - c_Y(m))^2$$

Die optimale Warping-Kurve wird mittels dem Optimalitätsprinzip von Bellman und der Dynamischen Programmierung[4] berechnet (siehe Implementation im Abschnitt 4.2.2).

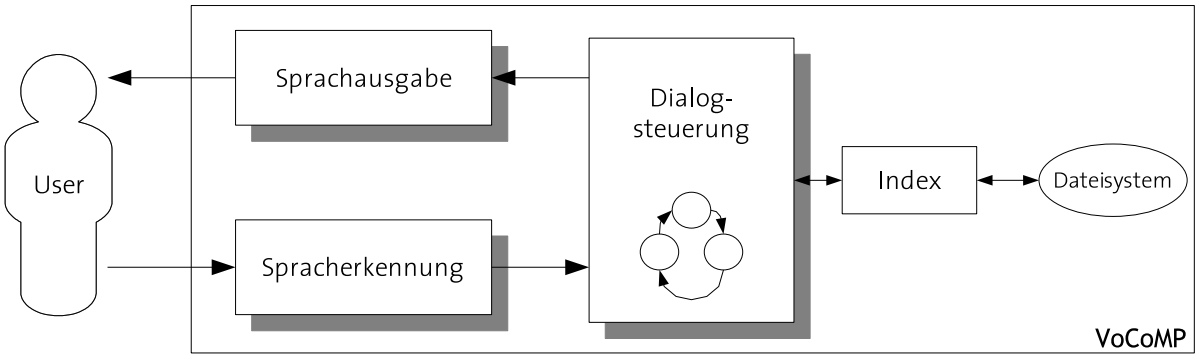
[4] siehe Skript *Sprachverarbeitung I* Kapitel 6.8.3

3 Konzept

Der Name unseres Systems lautet VoCoMP - *V*oice *C*ontrolled *M*edia *P*layer.

3.1 Grundaufbau

Eine Unterteilung des Systems in (Audio) Eingabe, Verarbeitung und (Audio) Ausgabe ist naheliegend. In diesem Zusammenhang wird im Weiteren von der Spracherkennung, der Ablaufsteuerung und der Sprachsynthese die Rede sein. Der Grundaufbau ist in Abbildung 9 dargestellt.



Figur 9: *Grundaufbau*

In Tabelle 3 sind die Bausteine zusammen mit ihren möglichen Ausbaustufen dargestellt. Die Komplexität (für Implementation und/oder Benutzer) nimmt jeweils von oben nach unten zu. Es folgt eine Beschreibung der einzelnen Stufen.

Der Trigger löst im Erkenner den Aufnahmevorgang aus. Ein Preprocessing des aufgenommenen Audiosignals ist nötig, wenn die Qualität des Mikrofonsignals den Anforderungen des Erkenners nicht genügt. Die Dialogsteuerung kontrolliert den eigentlichen Dialog mit dem Benutzer und verteilt Aufgaben an Erkenner, Media-Engine und Sprachausgabe. Die Media-Engine ist für die Steuerung der MP3-Dateien verantwortlich und benötigt deshalb Zugriff zum Dateisystem. Um eine schnelle Suche zu ermöglichen, müssen die Meta-Informationen (Titel, Künstler, Genre, Album) der vorhandenen MP3-Dateien in einer Suchstruktur zwischengespeichert werden. Damit muss dem Benutzer auch nicht die Forderung gestellt werden, die Dateien zu sortieren.

3.2 Komponenten

Im Folgenden werden die Systemkomponenten analysiert und einer Bewertung und Auswahl unterzogen. Dabei wird der Spracherkennung und der Dialogsteuerung jeweils ein eigenes Kapitel spendiert, da sie die zentralen Komponenten unseres Systems darstellen.

AUDIO INPUT		
Mikrofon	*Trigger*	*Preprocessing*
Eingebaut	kein	Silence-Detection
Externes Mik	Taster	Einfaches Filter
Headset	Threshold	Rauschunterdrückung
SPRACHERKENNUNG		
Erkennertyp	*Befehlssatz*	
Einzelwort	Fix	
Keyword-Spotter	Fix mit Argument	
Fliessende Sprache	Erweiterbar	
KONFIGURATION		
Personalisierung	*Feedback*	
keine	keine	
Spezifischer Erkenner	Presetauswahl	
Volle Anpassung	Frei konfigurierbar	
STEUERUNG		
Zustandsfolge	*File-Hierarchie*	*Media-Player*
Kontext insensitiv	Sortierte Mediafiles	Grundfunktionen
Kontext sensitiv	Unsortierte Mediafiles	Erweiterte Funktionen
		Intelligente Funktionen

Tabelle 3: *Die verschiedenen Komponenten des Steuersystems und ihre Ausbaustufen*

3.2.1 Mikrofon

Das Mikrofon kann entweder im Device eingebaut sein, als externes Mikrofon in der Hand gehalten oder als Teil eines Headsets getragen werden. Ein körperfernes Mikrofon ist bezüglich Umgebungslärm empfindlicher, ein eingebautes empfängt eventuell noch Störungen die durch das Gehäuse übertragen werden. Deshalb erwarten wir mit einem Headset oder ev. einem Kehlkopfmikrofon die beste Erkennungsqualität, da diese einen festen Abstand zum Sprecher haben.

3.2.2 Trigger

Als Trigger steht ein Taster am Device selbst oder in einem eigenen Gerät zur Diskussion. Eine elegantere - wenn auch komplexere - Möglichkeit ist das automatische Thresholding des Eingangssignals. Mit einem Trigger lässt sich der interessierende Signalabschnitt bereits deutlich eingrenzen, was den Overhead und somit den Rechenaufwand verkleinert.

Wir haben uns gegen einen automatischen Threshold entschieden, da es die Implementation in Matlab unnötig verkompliziert hätte und auch gar nicht in der Aufgabenstellung liegt. Das stetige Mithören in einem Hintergrundprozess würde auch auf die Performance drücken.

3.2.3 Preprocessing

Je nach Anforderung des Erkenners ist eine Vorverarbeitung des Audiosignals nötig. Dies kann ein einfaches Filter sein, welches zB. Störgeräusche, die nicht im Sprachband liegen, wegfiltert. Eine weitere Möglichkeit besteht in der Implementation einer adaptiven Rauschunterdrückung.

Da wir unser System nur in einer kontrollierten und ruhigen Laborumgebung testen wollen, ist keine Vorverarbeitung zur Rauschunterdrückung nötig. Es wird jedoch ein Thresholding angewandt um etwaige Stille am Anfang und Ende der Vergleichmuster wegzuschneiden (siehe Abschnitt 4.2.2 Endpunktdetektion).

3.2.4 Befehlssatz

Der Korpus der möglichen Steuerwörter (oder -sätze) kann entweder begrenzt sein oder später erweitert werden (= Lernen von neuen Befehlen). Eine Zwischenvariante besteht im Versehen der Befehle mit freien Argumenten, wie Interpreten-Namen oder Alben-Namen. Einem fixen Befehlsatz mit einer relativ einfachen Implementierung steht der erhöhte Benutzerkomfort eines erweiterbaren Befehlssatzes gegenüber.

Bei unserem Mustererkenner ist der Befehlssatz in Form der FSM-Übergangsbefehle realisiert und physikalisch als Audio-Vergleichsmuster in einem Verzeichnis abgelegt (siehe Abschnitt 4.2.4).

3.2.5 Personalisierung

Die Personalisierung des Systems kann auf verschiedenen Ebenen stattfinden. So besteht die Möglichkeit eigene Befehle aufzunehmen, was auch zu einer besseren Erkennungsrate führt, sowie die Einführung von Benutzerprofilen, falls das Device von mehreren Personen verwendet wird.

Unser System erlaubt in diesem Rahmen die Definition von beliebig vielen Vergleichsmustersammlungen, dh. jeder Benutzer kann sich ein eigenes Set von Mustern anlegen. Dies ist auch interessant, um verschieden ausgesprochene Befehle zu evaluieren (zB. die Verwendung des Phonologischen Alphabets, siehe Anhang C).

3.2.6 Zustandsfolge

Die Dialogablaufsteuerung kann kontext sensitiv sein, dh. eine Eingabe hängt von vorhergehenden Befehlen ab. Dies kann dazu führen, dass nicht alle Befehle in jeder Situation angewandt werden können.

Wir benutzen diese Eigenheit in der FSM um Befehle mit Argumenten zu versehen, dabei wird zwischen einstufigen, zweistufigen und mehrstufigen Befehlen unterschieden (siehe Abschnitt 4.2.4).

3.2.7 File-Hierarchie

Die Organisation der Medienfiles kann entweder flach oder hierarchisch sein, wobei im letzteren Fall auch mehrstufige Hierarchien mit Genre- oder Bandnamen denkbar sind.

Die Hierarchie wird losgelöst vom physikalischen Speicherplatz der Medienfiles gespeichert. Mittels Shell-Skript werden die Informationen über Künstlernamen, Titel, etc. extrahiert und in einer Datenbank hierarchisch zwischengespeichert (siehe Abschnitt 4.5).

3.2.8 Media-Player

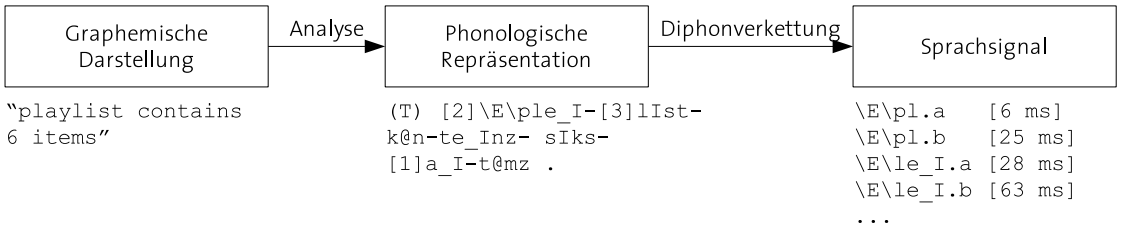
Die Steuerung der zu abspielenden Dateien kann entweder nur die grundlegenden Funktionen a la `play`, `stop`, etc umfassen oder auch komplexe Funktionen, wie die Suche nach bestimmten Files etc.

Die Ansteuerung des Mediaplayers wird mit Hilfe von Playlists[5] durchgeführt, welche je nach Treffermenge einen oder mehrere Tracks beinhalten, die Medienbefehle operieren dann immer auf dieser Playlist.

3.2.9 Feedback

Die akustische Antwort des Systems kann in verschiedenen Ausbaustufen realisiert werden, so ist entweder gar keine akustische Kontrolle, eine durch Pfeiftöne oder statische, voraufgenommene Sprachabschnitte denkbar. Die eleganteste Methode besteht natürlich in der kontextabhängigen Synthese der Antwort.

Bei der Analyse der Suche nach Dateien mit buchstabenweisem Eingeben von Titel, Künstler, Genre oder Album hat sich schnell gezeigt, dass eine akustische Wiedergabe der Anzahl Treffer oder sogar der einzelnen Treffer selbst notwendig ist. Damit ist die Forderung nach einer freien Text-to-Speech Synthese gestellt.



Figur 10: *Datenfluss der Diphonsynthese*

Um die in Textform vorliegenden Meldungen in ein akustisches Signal zu wandeln, verwenden wir die Diphonsynthese, die in Form des SVOX-Systems[6] am TIK bereits verfügbar ist. Eine grobe Übersicht liefert Abbildung 10. Die Ausgabe liegt zu Beginn

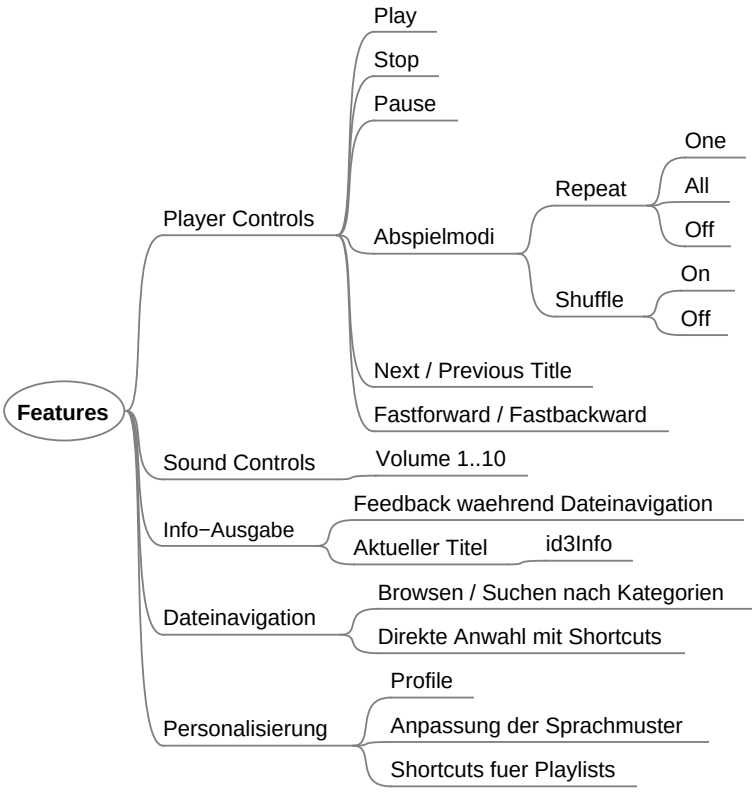
[5] Liste von abzuspielenden MP3-Dateien

[6] `http://www.tik.ee.ethz.ch/~spr/SPGinfo/node13.html`

in graphemischer Form vor. Mittels eines Parsers werden die Produktionsregeln einer Sprach-Grammatik angewandt und so eine Phonologische Repräsentation erstellt. Diese Darstellung kann dann unter Einbezug einer Diphon-Datenbank in das akustische Sprachsignal umgewandelt werden. Weitere Informationen zum Parsing sind im Skript der Vorlesung *Sprachverarbeitung II* zu finden.

3.3 Funktionsumfang

In diesem Abschnitt wird nun konkret der Funktionsumfang unseres Gerätes definiert. Inspiriert wurde dies durch die unter 2.2 evaluierten Funktionen, wobei natürlich speziell dem Aspekt der Sprachsteuerung Rechnung getragen wurde.



Figur 11: *Konzipierte features*

3.3.1 Player Controls

Die elementaren Abspielfunktionen werden wie gewohnt in der Art `Play`, `Pause` und `Stop` implementiert. Zwei Kommandi, die auf der Playlist einen Titel vorwärts oder zurück springen, dürfen natürlich ebensowenig fehlen wie die Möglichkeit, innerhalb eines Songs vor- und zurückzuspulen.

Als erweiterte Abspielmodi werden eine Wiederholfunktion und eine Zufallswiedergabe implementiert. Dabei soll wahlweise ein einzelner Titel oder die aktive Playlist

wiederholt werden können. Ähnlich dazu lässt sich mit dem Zufallsmodus die zufällige Reihenfolge der momentanen Wiedergabeliste ein- und ausschalten. Da diese Befehle auf Playlisten operieren, sind die gewünschten Modi somit bereits abgedeckt. Zusätzliche komplizierte Optionen wies sie beispielsweise der iHP140 verwendet (bis 10 verschiedene Zustände, siehe Anhang B), können vermieden werden.

3.3.2 Sound Controls

Lautstärke Im Unterschied zur gängigen schrittweisen Regelung, bei der die Lautstärke jeweils vom momentanen Wert um einen kleinen Wert erhöht oder verringert wird, soll unser Player eine absolute Lautstärkekontrolle verwenden. Innerhalb eines definierten Bereichs, z.B. von 0 bis 9, kann die gewünschte Lautstärke mit einem numerischen Wert direkt angewählt werden. Von dieser Methode erhoffen wir uns nach kurzer Gewöhnungsphase eine schnellere Kontrolle und ein erhöhter Bedienkomfort. Zudem wäre in einer sprachgesteureten Umgebung das bei einer relativen Steuerung notwendige wiederholte Eingeben eines Kommandos wie ”leiser“ für eine schnelle Absenkung der Lautstärke sehr mühsam.

Equalizer Eine detaillierte Equalizerregelung ist unserer Meinung nach ohne Display nicht funktionabel und wurde deshalb für die Implementierung nicht berücksichtigt. Vorabgespeicherte Soundpresets würden sich für ein sprachgesteuertes System eignen. Die strukturelle Implementation liefe analog zur Lautstärkeregelung ab (z.B. `Preset Rock`). Da der verwendete Audioplayer (XMMS, siehe 4.6) Equalizerpresets nicht unterstützt, erhielt diese Funktionalität nur eine geringe Priorität.

3.3.3 Info-Ausgabe

Informationen über Bitrate, ID3-Tags oder Volumepegel die heutige Player im Abspielmodus auf dem Display anzeigen sind zwar ein ”nice-to-have“, für den täglichen Gebrauch jedoch nicht immer sinnvoll oder unbedingt notwendig. Insbesondere bei einer rein akustischen Ausgabe sollte dieser Punkt aufs notwendige Minimum beschränkt werden. Der User soll die Möglichkeit haben, während dem Abspielen eines Files akustische Information über Titel und Artist zu erhalten. Dies muss jedoch im Gegensatz zu einem Display vom User mittels eines Kommandos initiiert werden.

Während der Navigation durchs Menü muss an gewissen Punkten ebenfalls ein Feedback stattfinden, um dem User eine Bestätigung über einen erkannten Befehl oder eine vorgenommene Aktion zu geben. Allerdings sollten diese Meldungen so kurz wie möglich gehalten werden, damit ein erfahrener Benutzer sich nicht unnötig über zeitraubende, bekannte Abläufe ärgern muss.

3.3.4 Dateinavigation

Datenstruktur Wir verfolgen den Ansatz einer selber generierten sortierten Datenbank, die ID3-Tags enthält. Damit wird die Schnittstelle zwischen unserem Player

und dem Filesystem auf zwei Funktionen reduziert, nämlich auf das einmalige Auslesen der Fileinformationen zur Erstellung der Datenbank und dem Zugriff des Mediaplayers auf die abzuspielende MP3-Datei. Als weiterer Vorteil steigt der Komfort bei der Suche nach Dateien und der Zusammenstellung von Wiedergabelisten in verschiedenen Kategorien.

Playlists Das Prinzip der Playlisten wird übernommen und ausgebaut. Jede Zusammenstellung an Titeln, auch ein einzeln ausgewählter Song, wird strukturell als Playlist aufgefasst, um ein konsistentes Verhalten über jede mögliche Auswahl zu gewährleisten. Entgegen dem normalen Vorgehen, wo Files einer Wiedergabeliste hinzugefügt werden, verfolgen wir hier jedoch einen umgekehrten Ansatz. Zu Beginn einer Zusammenstellung enthält die virtuelle Playlist die komplette MP3-Sammlung. Durch iteratives Einschränken der Auswahl mit Suchstrings in Kategorien wird die Playlist kontinuierlich auf eine Untermenge reduziert, bis sie die gewünschte Auswahl enthält. Wird die auf diese Weise erstellte Liste dann vom Benutzer aktiviert, agieren alle nachfolgenden Operationen bis zur nächsten Zusammenstellung jeweils auf dieser Playlist.

Browsing Das Suchen von Titeln sowie das Zusammenstellen von Wiedergabelisten geschieht simultan. Durch Auswahl einer Kategorie (Genre, Artist, Album oder Title) wird die Suchmaske gesetzt, auf der mit Buchstabieren die Treffermenge mit jedem zusätzlichen Buchstaben verkleinert wird. Nach jedem Buchstabe wird der Benutzer über die neue Untermenge informiert, entweder wird ihm die Anzahl Treffer mitgeteilt oder ihm werden bei einer genügend kleinen Playlist die Einträge zurückgegeben. Er hat jederzeit die Möglichkeit, durch Auswahl einer weiteren Kategorie und erneutes Buchstabieren die Menge weiter einzuschränken.

Zur Erläuterung folgt ein Beispiel:

Es soll eine Playlist mit allen Songs der Künstlerin "Alanis Morissette“ erstellt werden. In der Kategorie `Artist` wird die Menge durch Eingabe von `m` in unserer Beispielsdatenbank (siehe 4.6) bereits auf 35 Künstler beschränkt. Das Anhängen des nächsten Buchstabens `o` liefert nur noch die drei Treffer "Morcheeba“, "Monster Magnet“ und "Alanis Morissette“ . Die Eingabe zweier weiterer Buchstaben `r-i` würde die gewünschte Playlist erstellen, alternativ kann auch via Kategorie `Genre` und `r-o` (für `Rock`) zusätzlich der Musikstil eingeschränkt werden, was zum gleichen Resultat führte, da von den beiden anderen Kandidaten weder "Morcheeba“ noch "Monster Magnet“ demselben Musikstil zugeordnet ist.

3.3.5 Personalisierung

Shortcuts Um einmal erstellte Wiedergabelisten wiederverwenden zu können und den Zugriff auf diese zu beschleunigen dienen die Shortcuts: Die im obigen Beispiel erstellte Songliste kann unter einem beliebigen Sprachmuster abgespeichert werden, z.B. unter "meine Hits“. Bei Bedarf kann der Benutzer dann direkt aus dem Grundzustand des Systems mit der erneuten Spracheingabe dieses Musters die dazugehörige Playlist aktivieren, die dann ohne weiteres Kommando abgespielt wird.

Sprachausgabe Die Konfigurierbarkeit der Sprachausgabe ist denkbar, jedoch stark abhängig von der gewählten Synthesemethode. Parameter wie Stimmtyp, Grundfrequenz und Sprechgeschwindigkeit erlauben dem Benutzer ein Anpassen der Ausgabe an seine persönlichen Vorlieben, denkbar ist auch eine Auswahl aus vorgegebenen Sprechtypen. Vor allem die Sprechgeschwindigkeit könnte ein wichtiger Faktor sein, Menüabläufe zu beschleunigen und den Komfort für den Benutzer zu erhöhen.

Spracherkennung Durch die Verwendung des Mustervergleichs für die Spracherkennung ist dieser Teil des Systems bereits implizit relativ frei konfigurierbar. Vor der ersten Verwendung erstellt der Benutzer seine persönliche Musterkollektion, indem er den kompletten Sprachbefehlssatz aufnimmt. Bei schlechter Erkennungsrate kann er diese Muster später neu aufnehmen oder durch andere Kommandi austauschen. Dank dem Ansatz des Mustervergleiches hat der User bei der Wahl der Sprachbefehle die totale Freiheit, es können beliebige Wörter in einer beliebige Sprache verwendet werden. Die besten Resultate werden jedoch mit phonologisch gut unterscheidbaren Mustern erreicht (siehe Anhang C).

Profile Damit ein Gerät von verschiedenen Benutzern verwendet werden kann sind Profile notwendig. Ein Profil beinhaltet einen kompletten Satz an Sprachbefehlen sowie eventuell erstellte Shortcuts. Der Benutzer wählt zu Beginn sein Profil aus und lädt damit seine Einstellungen. Um verschiedene Muster für den gleichen Befehl zu vergleichen, kann auch während der Simulation umgeschaltet werden.

3.4 Spracherkennung

Aufgrund der Erkenntnissen aus Abschnitt 2.3 haben wir den Mustervergleich für die Spracherkennung gewählt. Folgende Punkte haben den Ausschlag gegeben:

- Der Mustervergleich benötigt eine kleinere Setupzeit für einen neuen Benutzer als ein HMM.
- Er passt als Worterkenner genau zu unseren Anforderungen.
- Es ist ausser der Aufnahme der Muster kein Training nötig.
- Er ist einfach zu implementieren.
- Befehle mit Argumenten lassen sich als verkettete Einzelworterkennungen realisieren.

3.5 Dialogsteuerung

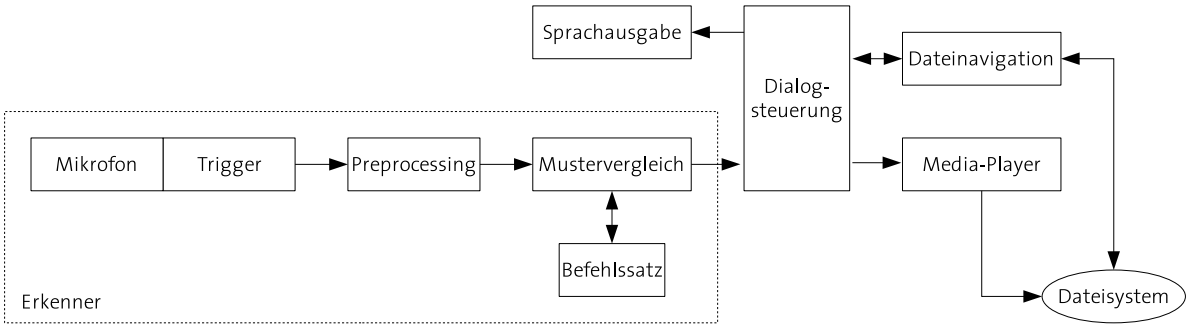
Die Dialogsteuerung sollte in jedem Zustand des Systems eine Rückmeldung auf mögliche Eingaben und Folgezustände geben. So hat sich die Verwendung einer Endlichen Zustandsmaschine (FSM) angeboten. Für die Implementierung haben wir das

Stateflow-Paket[7] von Matlab verwendet (Version 5.0), da aus den einzelnen States auf Matlab-Funktionen (m-files) und globale Variablen zugegriffen werden kann und wir die Buchführung über den aktuellen Zustand nicht selbst führen müssen. Zusätzlich bietet Stateflow eine bequeme, graphische Eingabemöglichkeit für den Dialogablauf, was uns beim Debuggen und Erweitern des Dialogs hilft.

[7] `http://www.mathworks.com/products/stateflow`

4 Implementation

4.1 Gesamtaufbau

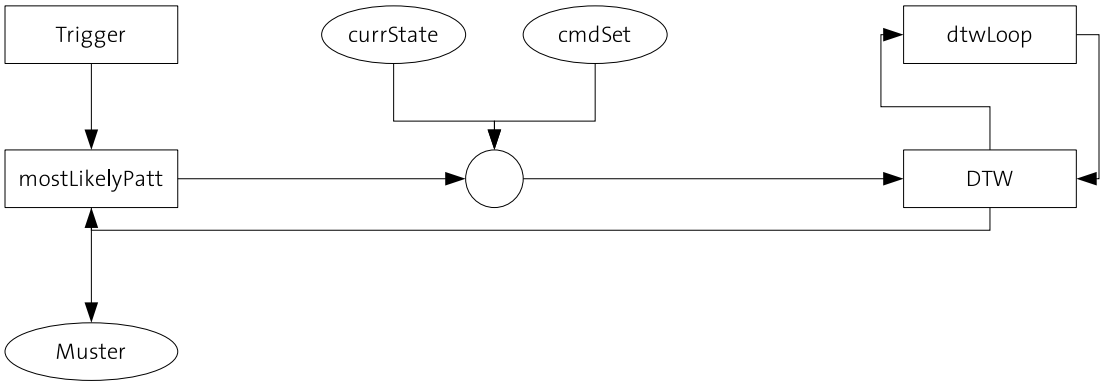


Figur 12: *Komponenten im Detail*

In Abbildung 12 ist das detaillierte Blockschema unseres Systems dargestellt. In den nachfolgenden Abschnitten wird die Implementierung der einzelnen Komponenten Spracherkennung, Dialogsteuerung, Dateinavigation und Mediaplayer dargestellt.

4.2 Erkenner

Der Aufgabenbereich des Erkenners startet bei der Übergabe eines Kommandos des Users und endet mit der Rückgabe des erkannten Musters. Abbildung 13 zeigt den Datenfluss im Erkenner. Die notwendigen Module werden nachfolgend detailliert erläutert.



Figur 13: *Datenfluss im Erkenner*

4.2.1 Triggerung und Aufnahme

Aus Performance- und Implementierungsgründen entschieden wir uns, die Aufnahme und darauffolgende Erkennung des Sprachsignals manuell mittels Benutzereingabe zu triggern. Per Mausklick wird die Aufnahme gestartet, das Mikrofonsignal wird in

eine FIFO-Pipe aufgezeichnet. Ein erneuter Klick stoppt die Aufnahme, die Pipe wird ausgelesen und kann als Vektor in Matlab weiterverwendet werden.

4.2.2 Mustervergleich

Endpunktdetektion Der Mustererkennungsalgorithmus startet mit der Detektion von Start- und Endpunkt im Testsignal (`detect_endpoints`). Über das Histogramm der ersten cepstralen Koeffizienten der Signalfenster wird der Threshold zwischen Sprachsignal und Rauschen gesetzt, als Start- bzw. Endpunkt gilt der Punkt, der zusammen mit seinem übernächsten Nachbarn den Threshold über- bzw. unterschreitet. Können in einem Signal keine Endpunkte detektiert werden, wird es als Rauschen deklariert und es findet kein Mustervergleich statt. Übersteigt der Unterschied in der Länge zweier zu vergleichender Muster eine bestimmten Faktor (im vorligenden Beispiel 2), gelten die zwei Signale a priori als zu verschieden, der Vergleich wird ohne DTW-Analyse abgebrochen.

Dynamic Time Warping (DTW) Zuerst werden die lokalen Distanzen zwischen Test- und Referenzsignal berechnet. Auf dieser zweidimensionalen Matrix wird dann nach dem Prinzip der dynamischen Programmierung der optimale Pfad von Start- zu Endpunkt ermittelt. Dieser Vorgang besteht aus zwei Teilbereichen: Im Forward processing werden mit einem Brute-Force-Ansatz iterativ für jeden Teilpunkt die lokalen Distanzen bis zum Start akkumuliert, damit besitzt nach dem Optimalitätsprinzip von Bellman[8] jeder Teilpunkt den optimalen Pfad zum Startpunkt. Im zweiten Teil wird dann vom Endpunkt ausgehend rückwarts der optimale Pfad gefunden, indem jeweils der momentan optimale Pfad weiterfvefolgt wird. Offensichtlicherweise ist der Teil des Forward Processings ein deutlicher Bottleneck bezüglich der Performance dieses Erkenners. Um dennoch einen grösseren Satz an Referenzmustern in akzeptabler Zeit vergleichen zu können musste dieser Programmteil optimiert werden.

DTW Forward Processing Das Forward Processing der Dynamischen Programmierung besteht im vorliegendem Fall aus zwei verschachtelten Schlaufen, die die Matrix der lokalen Distanzen der zwei zu vergleichenden Mustern durchläuft. Diese Matrix besitzt die Dimension der Länge der Merkmalsvektoren der zwei Muster. Da Matlab bekanntlich mit For-Loops nicht sonderlich performant arbeitet, wollten wir diesen Teil als externe C-Funktion implementieren.

In einem Zwischenschritt wurde dazu zuerst eine Matlab-Subfunktion `dtwLoop.m`) erstellt, die genau diese Funktionalität erbringen und dank darin definierter Übergabeparameter das Übersetzen in C vereinfachen sollte. Im zweiten Schritt implementierten wir dann mit Hilfe des Matlab-Interfaces MEX die effektive C-Routine. Die kompilierten MEX-files sind dynamisch gelinkte Subroutinen die der Matlab interpreter automatisch laden und ausführen kann. Ausserdem bieten sie bequeme Möglichkeiten, die Datenstrukturen von Matlab (v.a. Matrizen) einfach in eine entsprechende C-Variante

[8] siehe auch Skript *Sprachverarbeitung I*

zu übersetzen[9]. Die neue C-Funktion `dtwLoopC.c` kann nun aus der DTW-Routine identisch zur Matlabvariante aufgerufen werden.

Eher zufälligerweise testeten wir ebenfalls die Performance des ersten Zwischenschrittes. Überraschenderweise lieferte bereits dieser den grössten Zeitgewinn, wie in Tabelle 4 ersichtlich ist. Der totale Zeitaufwand konnte zum Schluss erheblich auf 7.63% reduziert werden, was nun auch eine grössere Anzahl zu vergleichender Muster erlaubt und das Testen des Erkenners um vieles komfortabler gestaltet.

Implementierungsvariante	abs. Zeit [ms]	Zeitgewinn [%]
Innerhalb DTW-Routine	2.9685	100.00
Matlab Subroutine	0.5205	17.53
C Subroutine	0.2264	7.63

Tabelle 4: *Optimierung des DTW-Loops: Absolute Zeit pro Mustervergleich und relativer Zeitgewinn der implementierten Varianten*

4.2.3 Mustererkennung

Die Funktion `mostLikelyPatt` übernimmt die Aufgabe, das spektral dem Testsignal ähnlichste Referenzsignal zurückzuliefern. Um dies perfomanter zu erledigen wird jeweils nur die für den momentanen Zustand relevante Untermenge an Referenzmustern verglichen. Dies erlaubt die Information über den momentanen Zustand des Zustandsautomaten (`gState`) und der globale Struct `gEvents`, der die ausgehenden Transitionen (und damit die möglichen Muster) für den gewünschten Zustand bereitstellt. Für jedes dieser Muster liefert der DTW-Algorithmus eine Distanz, das Muster mit der kleinsten Distanz wird als erkanntes Muster zurückgegeben.

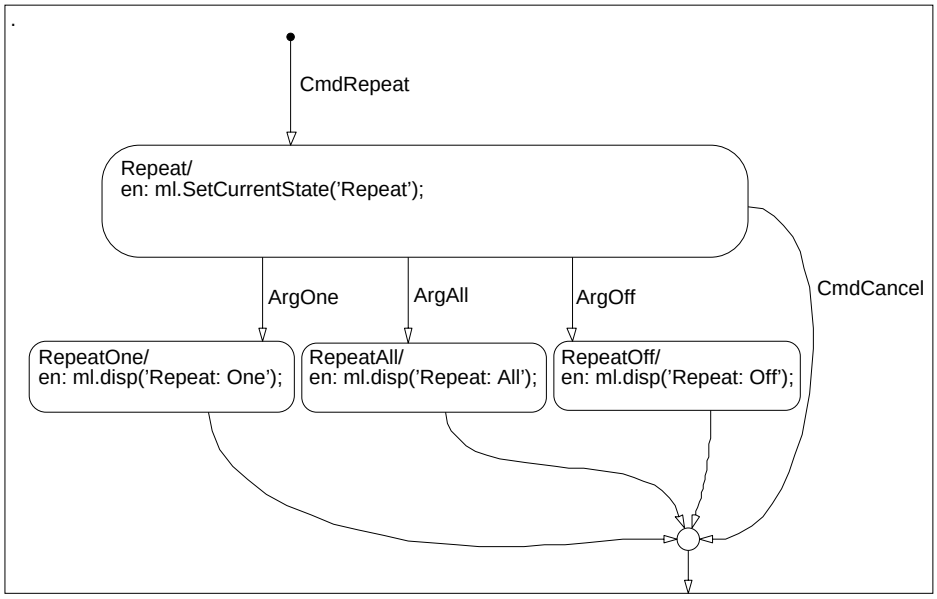
4.2.4 Befehlssatz

Der Befehlssatz muss den in 3.3 gewünschten Funktionsumfang ermöglichen. Gleichzeitig müssen bei der Definition die Einschränkungen berücksichtigt werden, welche die unter 2.3 gewählte Erkennungsmethode mit sich bringt. In Tabelle 5 ist der komplette Befehlssatz ersichtlich. Die Befehle können in drei Kategorien unterteilt werden:

Einstufige Befehle Diese Kommandi sind von der Komplexität die einfachsten im Befehlssatz. Ein einzelner Befehl wird eingegeben und direkt verarbeitet, das System kehrt in den Grundzustand zurück. Beispiele für solche Befehle sind `CmdStop` der die Wiedergabe anhält oder `CmdInfo`, der zum Abruf von Informationen über den momentanen Titel dient.

Zweistufige Befehle erwarten immer ein Argument, bzw. einen Folgebefehl. Nach erkanntem ersten Befehl ist die folgende mögliche Kommandomenge eingeschränkt

[9] `http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/matlab_external.html`



Figur 14: *`CmdRepeat`: Beispiel eines zweistufigen Befehls mit den 3 Argumenten `ArgOne`, `ArgAll` und `ArgOff`*

auf die Menge der für diesen Befehl möglichen Argumente. So erwartet das System besipielsweise nach dem Befehl `CmdRepeat` eines der drei Argumente `ArgRepeatOne`, `ArgRepeatAll` oder `ArgRepeatOff` (das Abruchkommando `CmdCancel` ist ebenfalls erlaubt). Figur 14 zeigt einen Screenshot, der diese Situation in unsrerer Applikation verdeutlicht.

Mehrstufige Befehle Für die komplexere Aufgabe der Navigation bzw. der Suchfunktion genügten zweistufige Befehle nicht mehr. Um den in 3.3.4 definierten Ansatz umzusetzen wurden Kommandi benötigt, die eine unbeschränkte Anzahl an Argumenten erlauben. So kann nach den Suchbefehlen `CmdGenre`, `CmdArtist`, `CmdAlbum` und `CmdTitle` eine beliebige Anzahl alphanumerischer Argumente (`ArgAlpha` und `ArgNumber`) folgen, die dann zu einer Argumentenkette verknüpft werden und den gewünschten Suchstring generieren (siehe 4.3.5).

4.3 Dialogsteuerung

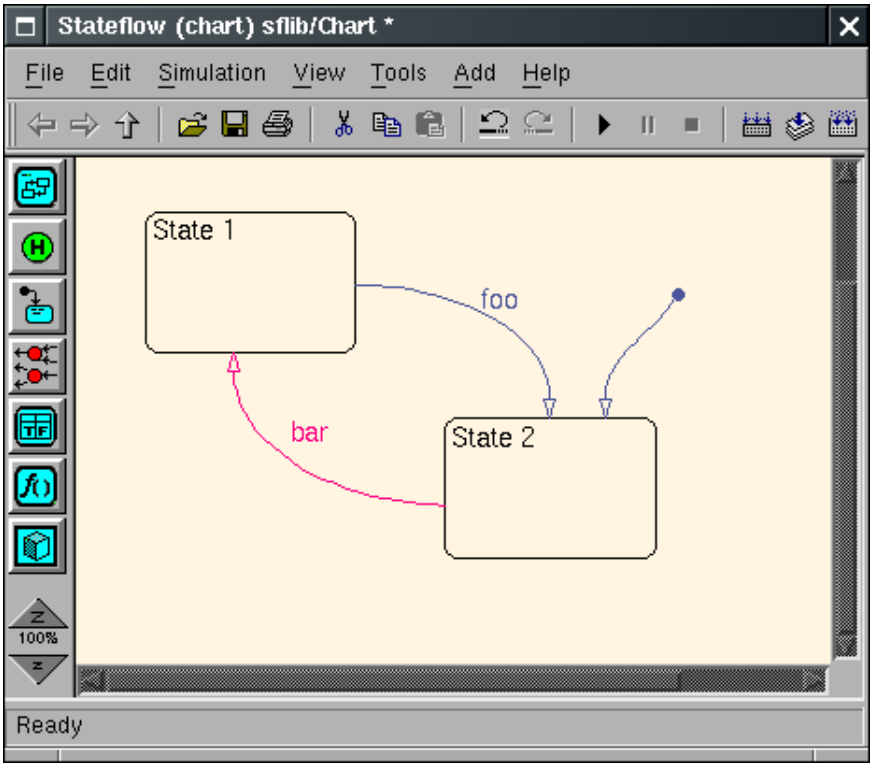
4.3.1 Stateflow

Elemente von Stateflow Stateflow ist Bestandteil von Simulink und setzt somit auf einer leistungsfähigen Engine für numerische Simulationen auf. Zusätzlich - und dies ist der interessante Punkt für unsere Arbeit - bietet Stateflow einen graphischen Editor um Endliche Zustandsmaschinen (FSM) zu designen und zu debuggen.

Die wichtigsten Elemente in Stateflow sind logischerweise die *States* (Zustände) welche via *Transitions* (Übergänge) verbunden sind. Abbildung 15 stellt eine einfache FSM

Name	*Befehl*	*Argument*
Player-Steuerung		
Play	`CmdPlay`	-
Pause	`CmdPause`	-
Stop	`CmdStop`	-
Next Item	`CmdNext`	-
Previous Item	`CmdPrev`	-
Fast Forward	`CmdFFwd`	-
Fast Backward	`CmdFBwd`	-
Set Volume to [1..10]	`CmdVolume`	`ArgNumber`
Set Repeat Mode to [One | All | Off]	`CmdRepeat`	`ArgOne` | `ArgAll` | `ArgOff`
Set Shuffle Mode to [On | Off[	`CmdShuffle`	`ArgOn` | `ArgOff`
Get Information about Title	`CmdInfo`	-
Navigation		
Suche Genre [:A-Z:]	`CmdGenre`	`{ArgAlpha}*`
Suche Künstler [:A-Z:]	`CmdArtist`	`{ArgAlpha}*`
Suche Album [:A-Z:]	`CmdAlbum`	`{ArgAlpha}*`
Suche Titel [:A-Z:]	`CmdTitle`	`{ArgAlpha}*`
Shortcuts		
Setze/Spiele Shortcut	`CmdShortcut`	`PattShortcut`

Tabelle 5: *Befehlssatz*



Figur 15: *Einfache FSM in Stateflow*

in StateFlow dar.

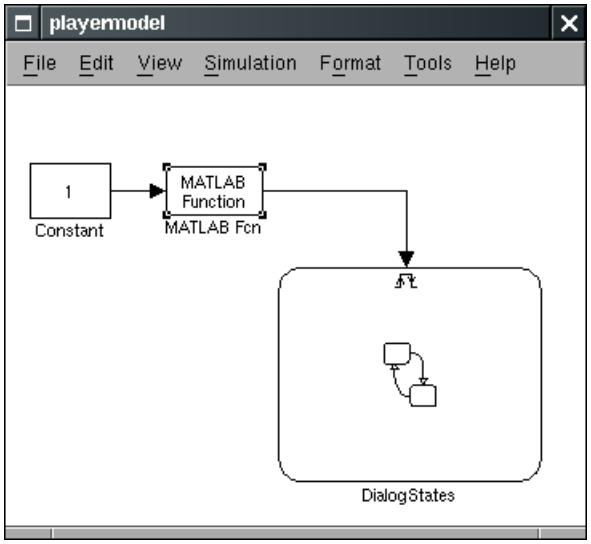
Beim Eintreten in einen State lassen sich sowohl Befehle in der Stateflow-eigenen Programmiersprache "Action" ausführen, als auch direkt Matlab-Befehle. Diese Eigenschaft nutzen wir unter anderem aus, um im Matlab-Workspace jeweils den aktuellen State mitzuführen.

Ein weiteres Element in Stateflow sind die *Events* welche intern oder extern generiert werden können und einen mehr oder weniger direkten Einfluss auf die Transitions haben.

Nachdem eine FSM designed worden ist, muss sie kompiliert werden. Für Matlab/Simulink erscheint das Stateflow-Objekt wie eine extern eingebundene C-Funktion.

Matlab-Stateflow Interface Die Objekte von Stateflow (States, Transition, ...) sind intern in einer baumförmigen Datenstruktur angelegt. Matlab bietet mit dem Befehl `sfroot` Zugriff auf die Wurzel dieses Baums. Damit lässt sich nach verschiedenen Objekten suchen und die FSM scannen, siehe unten.

Für das Auslösen von Transitions wird ein anderer Weg gewählt: Wie Abbildung 16 zeigt wird in Simulink eine Matlab-Funktion `sf_inject_event` als Input angehängt, welche bei Aufruf jeweils einen Vektor an die FSM übergibt. Ändert sich eine Komponente dieses Vektors (von 0 nach 1 und zurück) so wird dies von Stateflow als Event interpretiert und entsprechend verarbeitet. Der Index der sich ändernden Komponente entspricht der internen ID des Events.



Figur 16: *Verbindung von Matlab und Stateflow via Simulink*

4.3.2 Interne Datenstrukturen

Mit dem Matlab-Befehl `object.findDeep` (Stateflow API) sucht unsere Funktion `scanfsm` nach allen in Stateflow definierten States und Events und generiert daraus die Arrays `StateStruct` und `EventStruct`. Global werden diese beiden Arrays unter `gS` und `gE` im Workspace abgespeichert.

Zusätzlich wird auch noch eine Zustands-Übergangsmatrix `TransMat` bzw. global `gT` erzeugt. Diese wird verwendet um im Mustervergleich nur mit den momentan in Frage kommenden Befehlsmustern arbeiten zu müssen.

Bei den Identifieren (IDs, positive Integer) muss zwischen den Stateflow IDs (sfid) und unseren internen IDs unterschieden werden. Wir vergleichen die Stateflow IDs der State- und Transition-Objekte um die Matrix `gT` aufzubauen (siehe `scanfsm`). Schlussendlich werden aber die Indices der Elemente der State- und Transition-Vektoren in `gT` abgespeichert.

Die folgende Tabelle stellt die Struktur der Transition Matrix dar. Die Zeilennummer entspricht der internen IDs der Events, die Spaltennummer der internen ID des Quellzustands und das Element selbst der internen ID des Zielzustands.

	Spaltenindex: Quellzustand					
Transitions	0	0	0	0	…	0
	3	0	0	0	…	0
	0	0	0	18	…	0
	0	0	15	0	…	0
	…	…	…	…	…	0
	7	0	0	0	…	0

Die folgende Tabelle beschreibt die Struktur der Zustands- und Eventlisten.

`StateStruct`	Struct Array
`StateStruct.sfid`	Stateflow ID
`StateStruct.name`	Name des Zustands
`EventStruct`	Struct Array
`EventStruct.sfid`	Stateflow ID
`EventStruct.name`	Name des Events

4.3.3 Zustandsreferenz

In Tabelle 6 sind alle Zustände des Dialogablaufs mit ihren Parent-States und Beschreibung aufgelistet. Es wurden einige Hilfsstates weggelassen.

Tabelle 6: *Liste der Zustände*

Name	*Parent*	*Beschreibung*
Ready	Root	Grundzustand
OneState	Root	Fasst alle Befehle ohne Argument zusammen: Play, Pause, Stop, Forward, Backward, Info
TwoState	Root	Fasst alle Befehle mit einem Argument zusammen: Lautstärke, Shuffle, Repeat
InfiniState	Root	Beinhaltet die Datei-Suche und das Setzen von Shortcuts. Initialisiert die Datenbank-Matrix.
OpenShortcut	Root	Wartet auf die Eingabe eines Shortcut-Signals und löst dann den entsprechenden Event aus, um in den PlayShortcut State zu gelangen.
PlayShortcut	Root	Dieser State schreibt die aus dem Shortcut-Befehl resultierende Playlist und started den Mediaplayer.
Play	OneState	Ruft `playMedia('play',0)` auf.
Pause	OneState	Ruft `playMedia('pause',0)` auf.
Stop	OneState	Ruft `playMedia('stop',0)` auf.
Fwd	OneState	Ruft `playMedia('fwd',0)` auf.
Bwd	OneState	Ruft `playMedia('rew',0)` auf.
Info	OneState	Ruft `id3Info()` auf.
Volume	TwoState	Ruft `SetCurrentState` auf und wartet dann auf einen Befehl der Menge `ArgZero` bis `ArgNine`.
SetVolume	TwoState	-
Shuffle	TwoState	Ruft `SetCurrentState` auf und wartet dann auf einen Befehl der Menge `ArgOn` oder `ArgOff`.
ShuffleOn	TwoState	-
ShuffleOff	TwoState	-
Repeat	TwoState	Ruft `SetCurrentState` auf und wartet dann auf einen Befehl der Menge `ArgOn`, `ArgOne` oder `ArgOff`.
RepeatOne	TwoState	-
RepeatAll	TwoState	-
RepeatOff	TwoState	-
Title	InfiniState	Setzt `AssemblyType` auf 4 und geht zum Collector über.
Artist	InfiniState	Setzt `AssemblyType` auf 2 und geht zum Collector über.

Name	*Parent*	*Beschreibung*
Album	InfiniState	Setzt `AssemblyType` auf 3 und geht zum Collector über.
Genre	InfiniState	Setzt `AssemblyType` auf 1 und geht zum Collector über.
PlayDist	InfiniState	Wird beim Event `CmdPlay` aus dem Collector heraus aufgerufen, schreibt die Playliste (`writePlaylist`) mit den aktuellen Suchtreffern und startet anschliessend den Mediaplayer.
NewShortcut	InfiniState	Dieser State wird via `CmdShortcut` aus dem Collector heraus aufgerufen. Dort wird auf einen Sprach-Trigger gewartet (siehe 4.3.6), welcher die Aufnahme des neuen Musters erledigt.
StringAssembler	InfiniState	Bildet das Gefäss für Collector und Assembly. Beim Eintreten in diesen State wird der globale Suchstring `gAssemblerStr` zurückgesetzt.
Collector	StringAssembler	Hier wird jeweils auf die nächste Spracheingabe eines alphanumerischen Zeichens gewartet.
Assembly	StringAssembler	Beim Eintreten wird hier jeweils das Resultat des entsprechenden Sprachbefehls ausgewertet (es sollte ein alphanumerisches Zeichen sein) und der entsprechende ASCII-Code an Matlab übergeben, wo die Funktion `StrAssembly` das Zeichen an den globalen Suchstring anhängt. Anschliessend wird die Funktion `selectionOutput` aufgerufen um die Suchmenge zu aktualisieren.

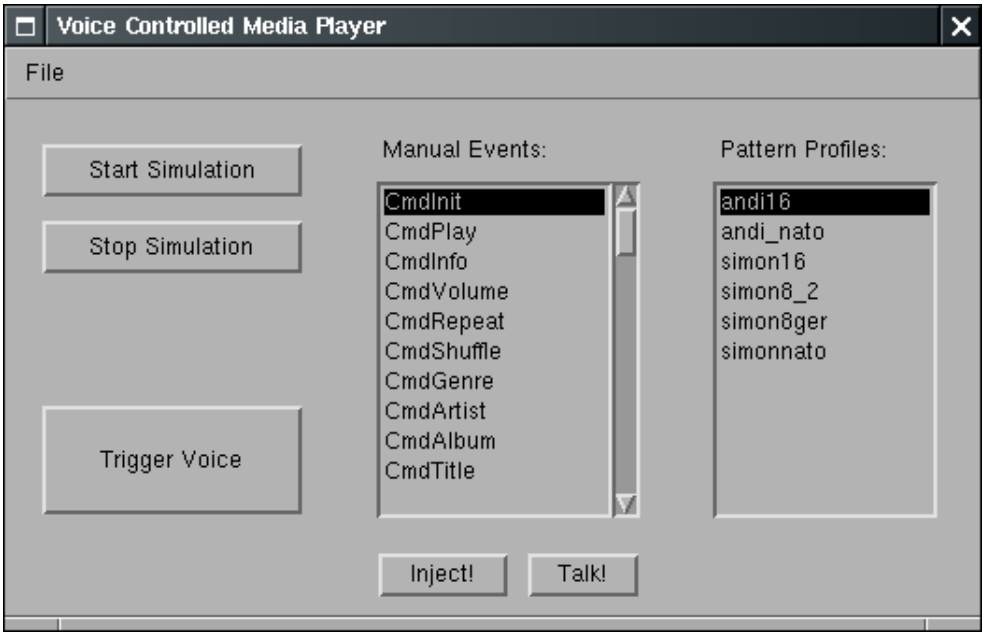
4.3.4 User-Interface

In Abbildung 17 ist das Hauptfenster von VoCoMP dargestellt, nachfolgend eine Beschreibung der einzelnen GUI-Elemente.

Start Simulation Diese callback-Funktion sendet einen "start simulation" Befehl an Simulink, dadurch wird auch automatisch die FSM gestartet. Während der Simulation wird im Stateflow-Fenster der aktuelle Zustand blau umrandet.

Stop Simulation Die Simulation wird beendet, beim nächsten Start wird wieder im Grundzustand *Ready* begonnen.

Trigger Voice Diese callback-Funktion setzt ein Flag und beginnt den FIFO mit Audiodaten vom Mikrofoneingang zu füllen. Wird der Button nochmals betätigt, dann



Figur 17: *Das Hauptfenster von VoCoMP*

wird die Aufnahme gestoppt, der FIFO wird ausgelesen und die Daten an den Mustervergleich weitergereicht. Befindet sich die FSM im Zustand `NewShortcut` so wird kein Mustervergleich durchgeführt, sondern das aufgenommene Signal im Shortcut-Ordner abgelegt (siehe 4.3.6).

Inject! Diese Funktion injiziert den aus der "Manual Events" Liste ausgewählten Event in die FSM.

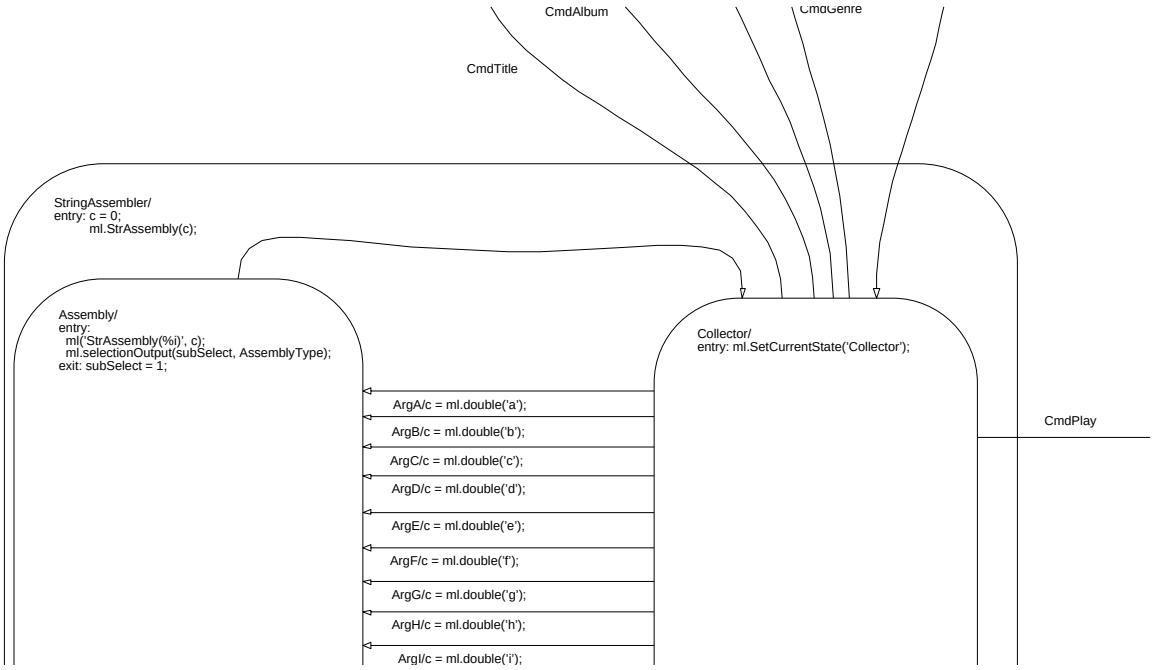
Talk! Diese Funktion ist für Demonstrationszwecke gedacht. Sie nimmt das Vergleichspattern des ausgewählten Events und fügt eine kleinen Offset hinzu. Diese Summe wird danach in den Mustervergleich geschickt, so dass sicher ein Treffer erzielt wird. Gleichzeit wird das Muster noch abgespielt. Diese Funktion macht vor allem Sinn um die Rechengeschwindigkeit zu prüfen und ein intuitives Gefühl für die Bedienung zu erhalten.

Pattern Profiles Diese Liste wird beim Programmstart aus den entsprechenden Unterverzeichnissen im ∼`/src/detector/signals/` Verzeichnis aufgebaut. Die Auswahl wird global gespeichert, so das der Mustervergleich das jeweilige Musterset verwendet.

File > ScanFSM Dieser Menüpunkt generiert die oben beschriebenen internen Datenstrukturen und füllt die Liste mit den Events. Dieser Aufruf wurde nicht automatisiert, da sie jeweils aufgerufen werden muss, nachdem die Dialog-FSM verändert wurde.

4.3.5 String Assembler

In Abbildung 18 ist der Hauptbereich des StringAssemblers dargestellt. Die FSM geht nach einem der Befehle `CmdArtist`, `CmdAlbum`, `CmdTitle` oder `CmdGenre` in den *`Collector`* Zustand über, wonach auf die Eingabe eines alphanumerischen Zeichens gewartet wird. Bei jeder Eingabe wird das entsprechende Zeichen vom Zustand `Assembly` mit der Funktion `StrAssembly()` an den Suchstring angehängt und dieser neu auf die Datenbank (siehe 4.5) angewendet. So wird die Treffermenge sukzessive reduziert.

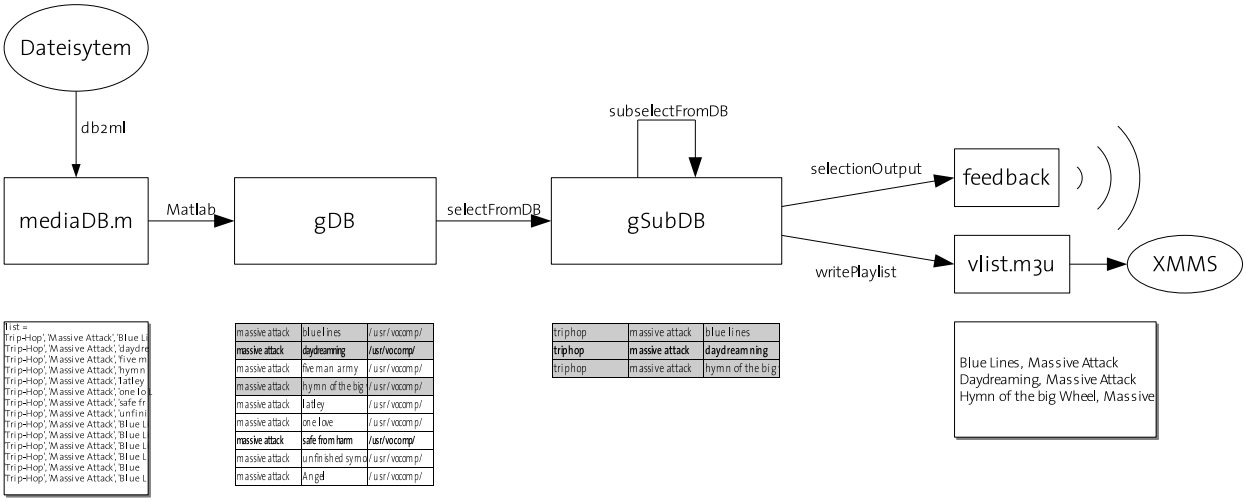


Figur 18: *Hauptbereich des String-Assemblers*

Die Variable `AssemblyType` entscheidet jeweils in welcher der vier Kategorien Artist, Album, Title und Genre gesucht wird (siehe auch Abschnitt 4.5). Dieser Loop wird verlassen, sobald einer der Befehle `CmdPlay`, `CmdArtist`, `CmdAlbum`, `CmdTitle`, `CmdGenre` oder `CmdCancel` eingesprochen wird.

4.3.6 Hinzufügen von Shortcuts

Wird die Trigger-Funktion im Zustand *`NewShortcut`* aufgerufen, so wird mit dem aufgenommenen Sprachsignal kein Mustervergleich durchgeführt, sondern das Signal wird der Funktion `addPat` übergeben, welche im Unterverzeichnis `shortcuts/` des aktuellen Pattern-Verzeichnisses fortlaufend nummerierte Shortcut-Files anlegt (siehe Funktion `pb_trigvoice_Callback`).



Figur 19: *Datenfluss vom Dateisystem zur Playlist*

4.4 Synthese

Die Gruppe für Sprachverarbeitung unseres Departements stellte uns ihr Sprachsynthesesystem SVOX zur Verfügung. Dieses Paket verringerte unsere Arbeit bezüglich Synthese auf ein Minimum. Dank der in Matlab vorhandenen Java-Schnittstelle konnte der SVOX-Server direkt aus unserer Applikation angesprochen werden.

VoCoMP kann mit oder ohne Sprachsynthese gestartet werden. Wird die Synthese aktiviert, wird unmittelbar nach Start des VoCoMP die Funktion `initSynthesizer`aufgerufen, welche SVOX als Hintergrundprozess startet und initialisiert. Danach kann mit der Funktion `talk` ein beliebiger String an den Server gesendet werden, der dann als synthetisiertes Audiosignal zu hören ist.

4.5 Dateinavigation

Abbildung 19 zeigt den Ablauf von der MP3-Sammlung im Dateisystem zu einer Playlist für den VoCoMP. Die verschiedenen Schritte, welche zu diesem Zweck implementiert wurden, werden in den folgenden Abschnitten dargelegt.

MP3-Database einlesen Mittels des Shellskripts `db2ml.sh` wird das Filesystem das die MP3-Sammlung enthält rekursiv nach Files mit der Endung *.mp3* durchsucht. Von jedem gefundenem File wir der Name inklusive der Information seines ID3-Tags zeilenweise in eine Textdatei (`mediaDB.m`) geschrieben. Dieses File wird nun in Matlab direkt als Matrix eingebunden und repräsentiert als globale Variable `gDB` die MP3-Datenbank. Listing 1 zeigt einen Ausschnitt aus dieser Datei. Folgende Informationen werden pro Datei abgespeichert:

```
Genre, Artist, Album, Title, Path
```

Die Reihenfolge der Spalten wurde aus hierarchischen Gründen gewählt. Ein Eintrag

```
function [] = mediaDB
list = {
...
'Trip-Hop', 'Massive Attack', 'Mezzanine',   'Risingson',     './albums/Massive Attack/Mezz...
'Trip-Hop', 'Massive Attack', 'Mezzanine',   'Teardrop',      './albums/Massive Attack/Mezz...
'Trip-Hop', 'Massive Attack', 'Protection',  'Better Things', './albums/Massive Attack/Prot...
...
};
```

Listing 1: *Ausschnitt aus mediaDB.m.*

ist immer eine Untermenge seines linken Nachbars. So ist beispielsweise ein Titel immer in einem bestimmten Album vorhanden, dieses gehört wiederum zu einem bestimmten Künstler, welcher Teilmenge einer Musikrichtung ist[10]. Der letzte Eintrag `Path` existiert ausserhalb dieser Hierarchie. Er wird für die usergesteuerte Filesuche nicht verwendet und dient lediglich dem Player zur Lokalisation des MP3Files auf der Harddisk. Sind weitere Kategorien erwünscht, soll beispielsweise auch nach Erscheinungsjahr gesucht werden können, lassen sich diese der Liste einfach hinzufügen. Die Hierarchie erlaubt auch eine einfache, alphabetische Sortierung der Liste nach Kategorien. Diese Reihenfolge wird nachher auch beim der Ausgabe der selektierten Titel verwendet.

Untermenge der DB selektieren Mit der Funktion `selectFromDB` kann eine Untermenge der MP3-Datenbank ausgewählt werden. So selektiert beispielsweise der Aufruf Morïn Ärtist"diejenigen Titel, bei denen der Künstler der String `Mor` enthält und speichert diesen als Subset in `gSubDB` ab. Auf unserer testdatenbank werden dadurch die zwei Künstler "Morcheeba" und "Alanis Morissette" ausgewählt. Der nächste Aufruf von `selectFromDB` operiert dann wahlweise erneut auf der gesamten Datenbank `gDB` oder auf der vorher eingeschränkten Menge `gSubDB`. Somit lässt sich die Menge der Titel iterativ nach Kategorien verkleinern, bis die gewünschte Liste übrigbleibt.

Die Funktion `selectionOutput` bereitet die erstellte Liste auf die Ausgabe vor. Liegt die Anzahl der noch vorhandenen Treffer über einem bestimmten Threshold, wird nur die Anzahl der Treffer ausgegeben. Enthält das Suchresultat wenige (1 bis threshold) Treffer, werden diese in alphabetischer Reihenfolge und nummeriert ausgegeben. Liefert die Suche keinen Treffer, wird dies gemeldet und die Auswahl auf die letzte, nichtleere Menge zurückgesetzt.

4.6 Media Player

XMMS Um unserem Testsystem ein wenig Leben einzuhauchen sollte das Abspielen der MP3-Dateien ermöglicht werden. Von der auf unserem System vorhandenen Software wählten wir XMMS aus, ein unter UNIX/Linux häufig verwendeter Multimedia Player[11]. Dies aufgrund optionaler Parameter, die es erlauben, XMMS via Konsolenbefehle "fernzusteuern". Die Funktionalität ist jedoch verglichen mit der norma-

[10] Dies stimmt in gewissen Ausnahmefällen nicht. Ein Künstler kann auch in mehreren Genres vorkommen.

[11] http://www.xmms.org

len Bedienung leider stark eingeschränkt: Neben Play, Pause und Stop beenden Skip Forward/Backward die Liste der Konsolenkommandi, diese Funktionen wurden in unserer Applikation auch implementiert. Lautstärkeregelung sowie die Auswahl des Abspielmodus ist mit dieser Art der Steuerung nicht möglich.

m3u-Playlists Da die selektierte Untermenge der MP3-Sammlung (`gSubDB`, siehe 4.5) strukturell bereits eine Art Playlist darstellt, wurde dieser Ansatz auch auf den Player weitergeführt. Hat der Benutzer die Auswahl der gewünschten Titel erstellt und befiehlt dem System, diese abzuspielen, wird die Auswahl in eine m3u-Liste geschrieben (Funktion `writePlaylist`). `.m3u` ist ein verbreitetes Format für Wiedergabelisten, auch XMMS handhabt Playlists in dieser Form. Diese Liste wird nun dem Player übergeben, welcher bis zur Zusammenstellung einer neuen Titelauswahl auf dieser Liste operiert.

Trackpointer Leider fehlt fehlt bei der Konsolensteuerung für XMMS die Möglichkeit, Information über den Status des Players, insbesondere über den laufenden Titel zu erhalten. Um dennoch die in 3.3 spezifizierte Funktionalität der Informationsausgabe zu erhalten, musste zusätzlich eine Variable `gTrackPointer` hinzugefügt und bei den Befehlen `CmdNext` bzw. `CmdPrev` korrekt nachgeführt werden, um jeweils den momentan aktiven Titel anzuzeigen. Dank dieses Trackpointers und der in der Datenbankmatrix vorhandenen ID3-Tags können nun mit der Funktion `id3Info` Informationen zum laufenden Titel in akustischer oder visueller Form ausgegeben werden.

MP3-Sammlung Die MP3-Datenbank, mit der das System getestet wurde, enthält 1594 Titel in 189 Alben, 165 Artisten aus 16 Genres. Die Gesamtgrösse von 5GB entspricht ungefähr einer 25%igen Auslastung des Speichervermögens eines aktuellen portablen harddiskbasierten MP3-Players.

5 Zusammenfassung und Ausblick

Resultate

Die Verwendung von Matlab und Stateflow zur Implementierung einer Dialogsteuerung als Endlichen Zustandsautomaten und einer einfachen Spracherkennung in Form eines Mustervergleichs haben sich positiv ausgewirkt. Entstanden ist ein flexibel erweiterbarer Prototyp mit dem sich durch einfache Eingriffe verschiedene Dialogführungen evaluieren lassen. Der Sprachbefehlssatzes lässt sich zur Erweiterung des Funktionsumfangs einfach ausbauen, verschiedene Menüabläufe können ausprobiert werden.

Ein direkter Vergleich im Bedienkomfort zwischen unserem System und herkömmlichen Playern ist ohne aufwändige Versuche mit einer grösseren Anzahl Testpersonen schwierig. Zwar lassen sich die notwendigen Schritte, die zur Abarbeitung ausgewählter Use Cases notwendig sind, bei den verschiedenen Systemen noch relativ einfach formulieren. Wie sich der zeitliche Aufwand von einfachen Klicks bis hin zu Scrollen durch die komplette Menge an Titeln in Relation zur Performanz von Sprachkommandos setzen lässt, ist dagegen ziemlich unklar.

Die subjektive Meinung der Authoren nach kurzen Tests des VoCoMP ist positiv. Der Ansatz, die Kategoriesuche nach Dateien mittels Buchstabieren anzupacken bewährt sich in der Praxis, steht und fällt aber mit der Erkennungsrate des Mustervergleichs. Das schnelle Anwählen von Wiedergabelistem mittels benutzerdefinierten Sprachshortcuts verspricht einen deutlichen Zeit- und Komfortgewinn gegenüber den untersuchten Playern.

Weiterführende Arbeiten

In folgenden Punkten erachten wir Verbesserungen und Erweiterungen als sinnvoll:

Performancesteigerung Bei der Implementation dieses Testsystems wurden die Prioritäten auf Flexibilität und weniger auf Performance gesetzt. Durch die Wahl der Toolchain ist der Player nicht besonders schnell. Um einen fairen Vergleich zu anderen marktreifen Systemen ziehen zu können wären Optimierungen in der bestehenden Applikation oder eine Neuimplementierung in einer anderen Programmierumgebung notwendig.

Spracherkennung optimieren Der Teil der Spracherkennung lässt sich in verschiedenen Bereichen optimieren. Der Erkennungsalgorithmus wurde abgesehen von kleinen Ausnahmen direkt übernommen. Eine spezifische Anpassung an unser System könnte sich bezüglich Erkennungsrate und Ausführungszeit lohnend auswirken. Wir testeten das System nur unter idealen Laborbedingen. Damit der Erkenner auch in Alltagssituationen robust funktioniert ist ein erheblicher Aufwand im Funktionsablauf notwendig. Ein erweitertes Preprocessing das den Erkenner robust gegen Umgebungslärm ist dafür unabdingbar. Durch eine verfeinerte Methode, die Sprachmuster aufzunehmen (beispielsweise Mittelung über

mehrmals aufgenommene Muster) liessen sich höchstwahrscheinlich ebenfalls Verbesserungen der Erkennungsrate erzielen. Ein zuverlässiges automatisches Thresholding anstelle eines manuellen Triggers, das den Schritt der Spracherkennung auslöst wäre ebenfalls ein erwünschtes Feature, das sich positiv auf den Bedienkomfort auswirkte.

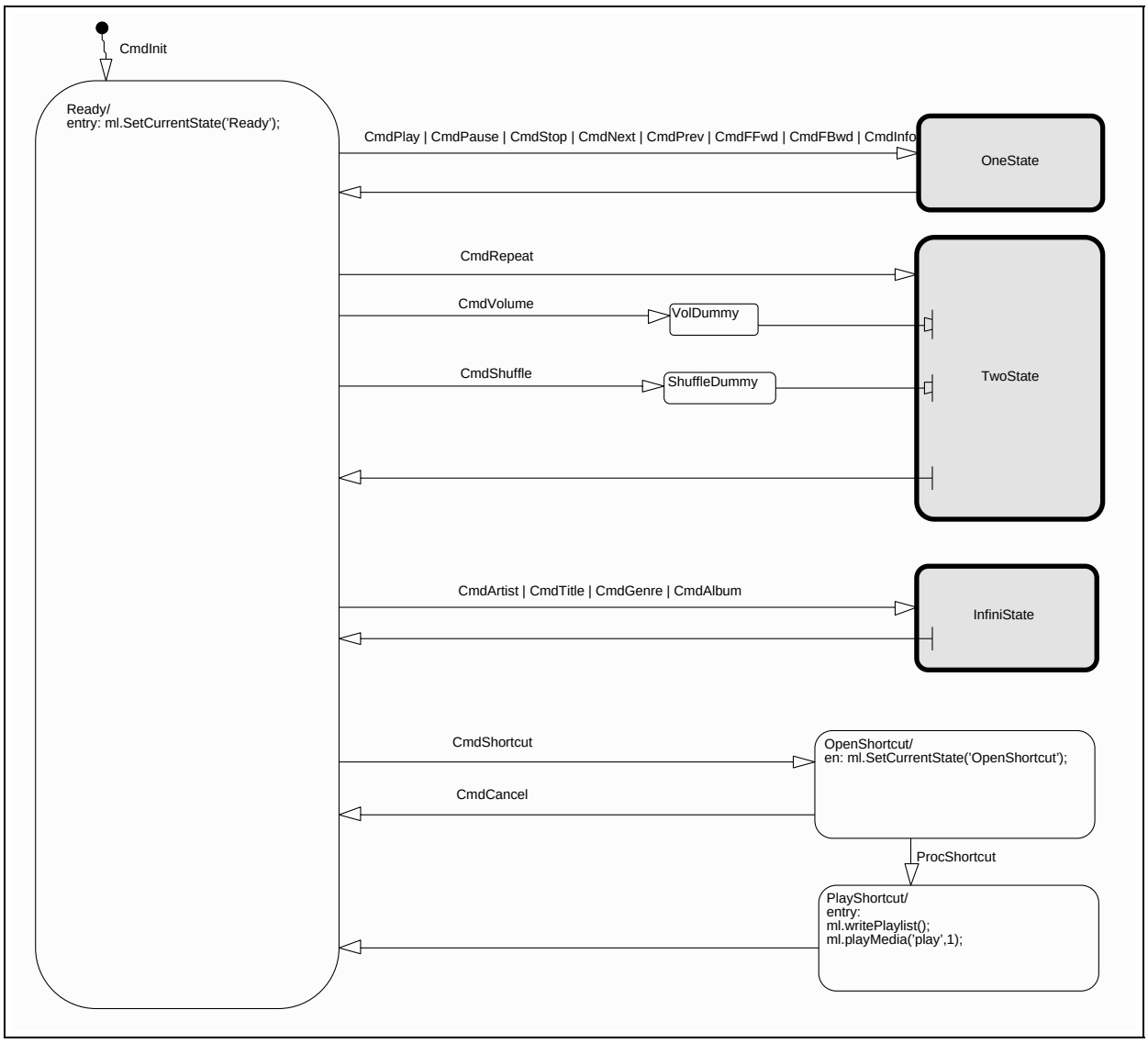
Evaluierung mit Testpersonen Testläufe, bei denen Probanden die Bedienung des Gerätes lernen, können Erkenntnisse über schlechte Abläufe in der Dialogsteuerung oder über fehlende Funktionalität liefern. Weiterführende Experimente in dieser Richtung wären Voraussetzung für weitere Implementierungen.

Weiterführende Soft- und Hardwareprojekte Die Punkte 5 und 6 unserer Roadmap beschäftigen sich mit der optimalen Implementierung eines sprachgesteuerten Mediaplayers. So ist es denkbar, ein vollständig in C/C++ implementiertes Plug-In für XMMS, WinAmp und Konsorten zu entwickeln.

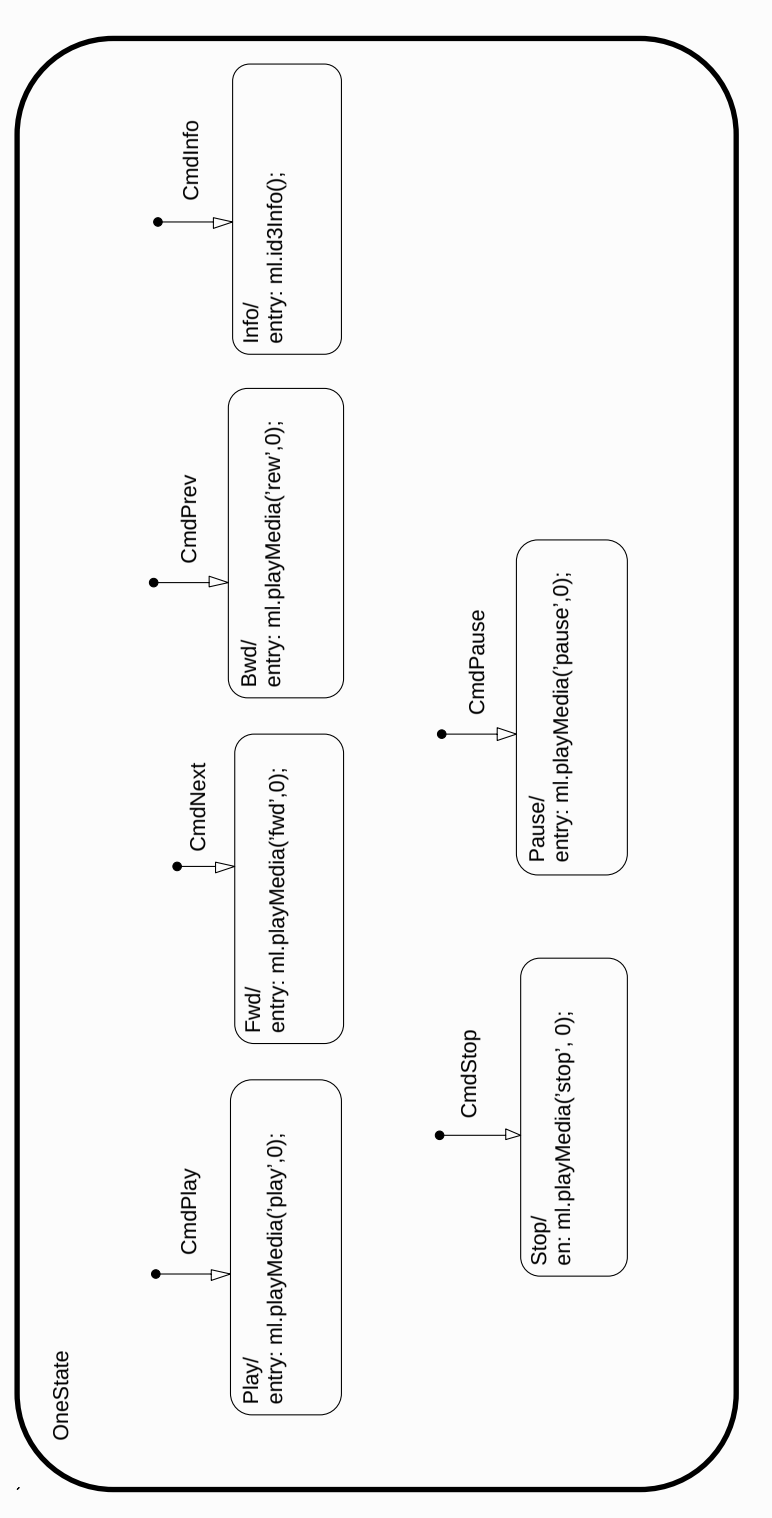
Interessant ist sicher auch die Entwicklung von Hardwarezusätzen für bereits erhältliche Player oder sogar Stereoanlagen, wobei sich dort in Sachen Bedienung wieder neue Probleme ergeben (zum Beispiel die grössere Distanz zwischen Benutzer und Gerät).

Last but not least existieren in aktuellen Mobiltelefonen bereits die Resourcen einen sprachgesteuerten Mediaplayer zu realisieren. Diese Geräte haben teilweise auch schon die Funktionalität für Mustervergleiche implementiert. Eine Dialogsteuerung nach der Art vom VoCoMP für eine im Handy integrierte MP3-Applikation könnte auch dort interessante Aspekte liefern.

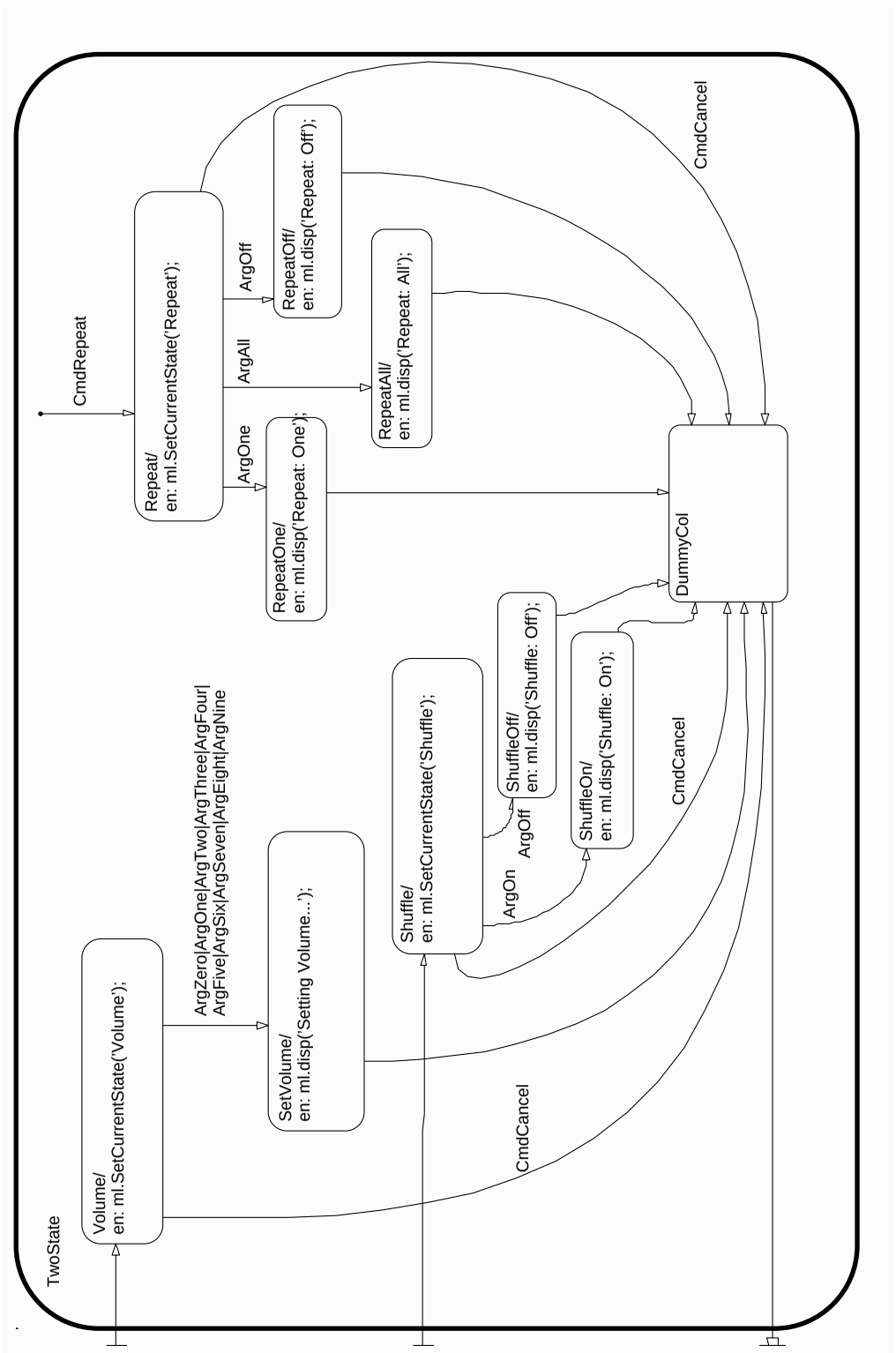
A Stateflowcharts des VoCoMP



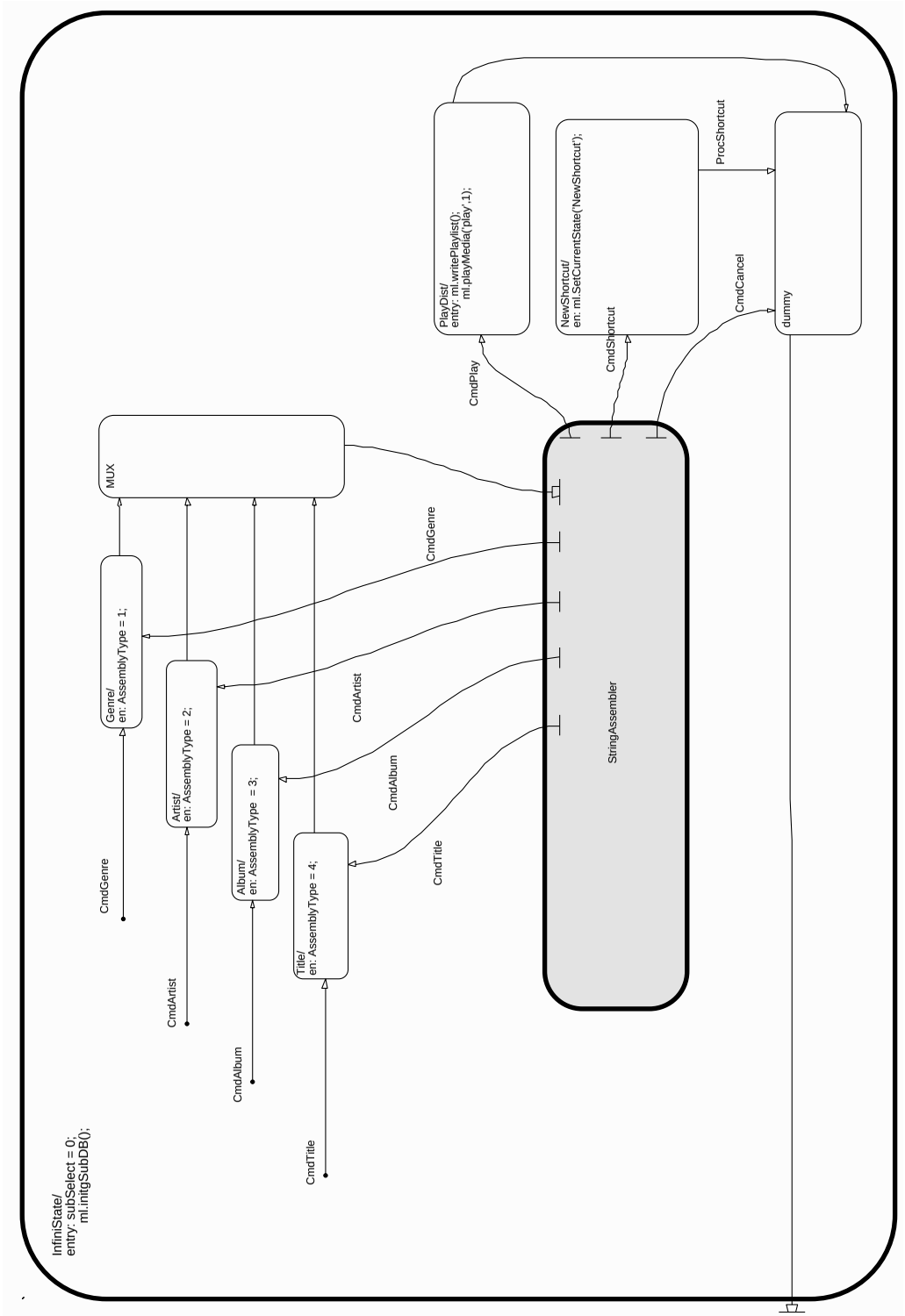
Figur 20: *VoCoMP Playermodel MainView*



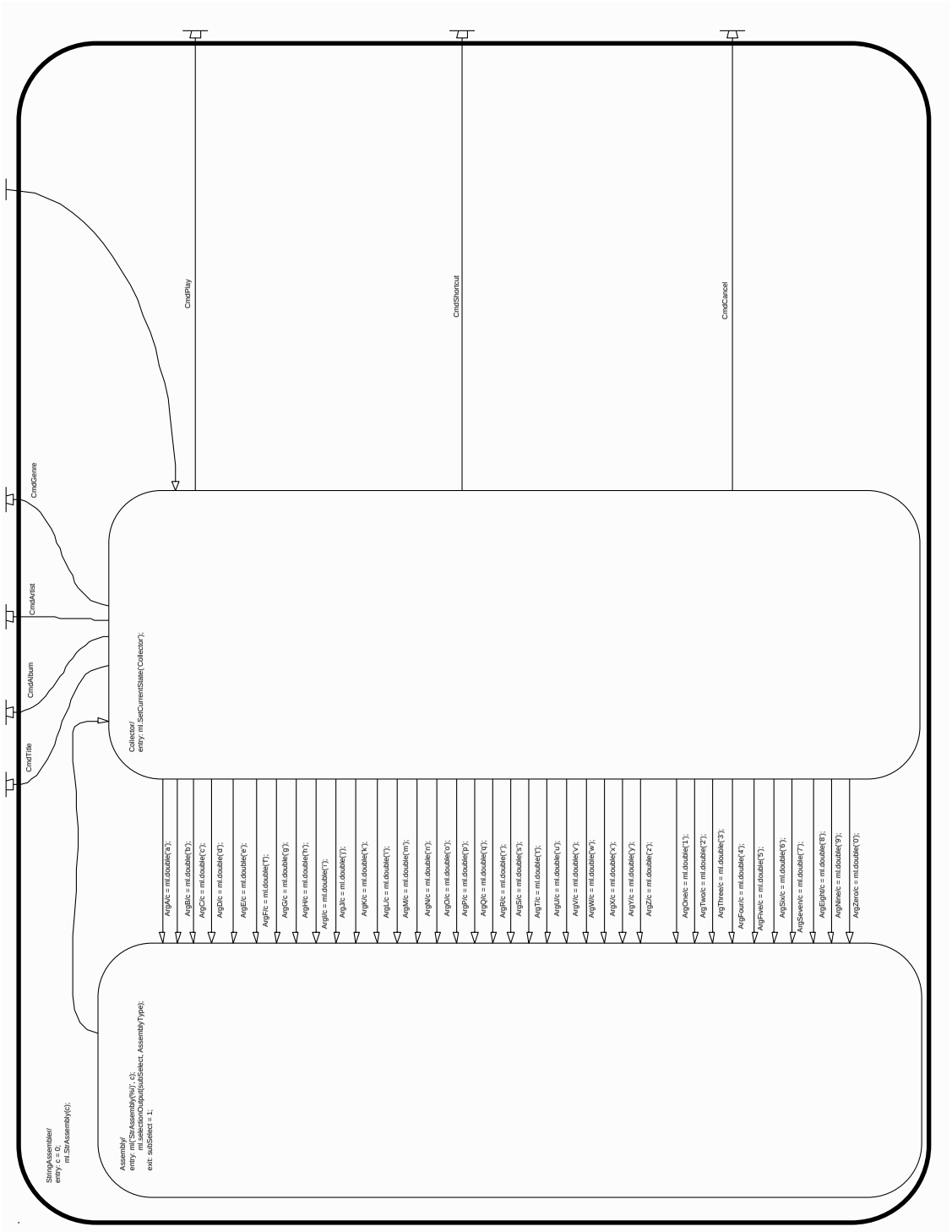
Figur 21: *VoCoMP Playermodel OneState*



Figur 22: *VoCoMP Playermodel TwoState*

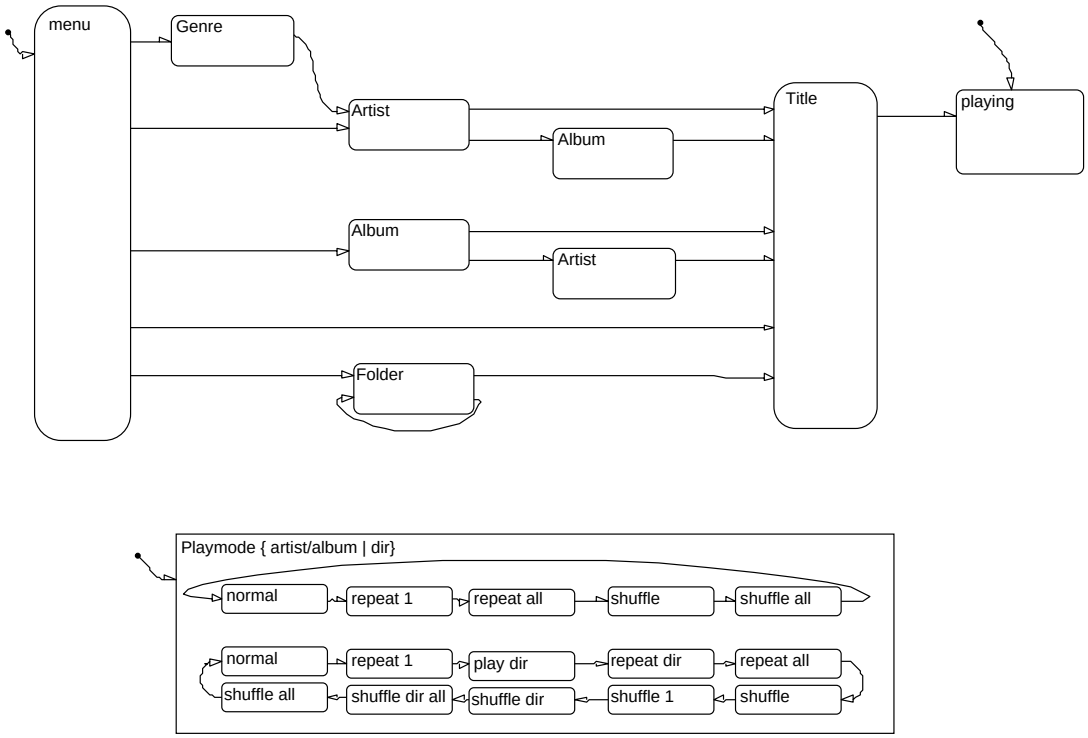


Figur 23: *VoCoMP Playermodel InfiniState*

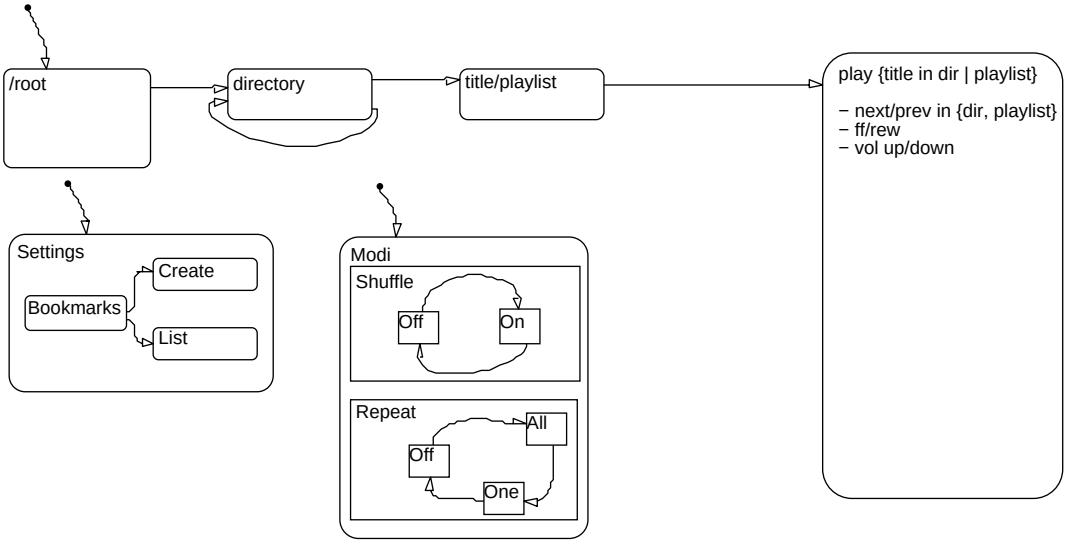


Figur 24: *VoCoMP Playermodel StringAssembler*

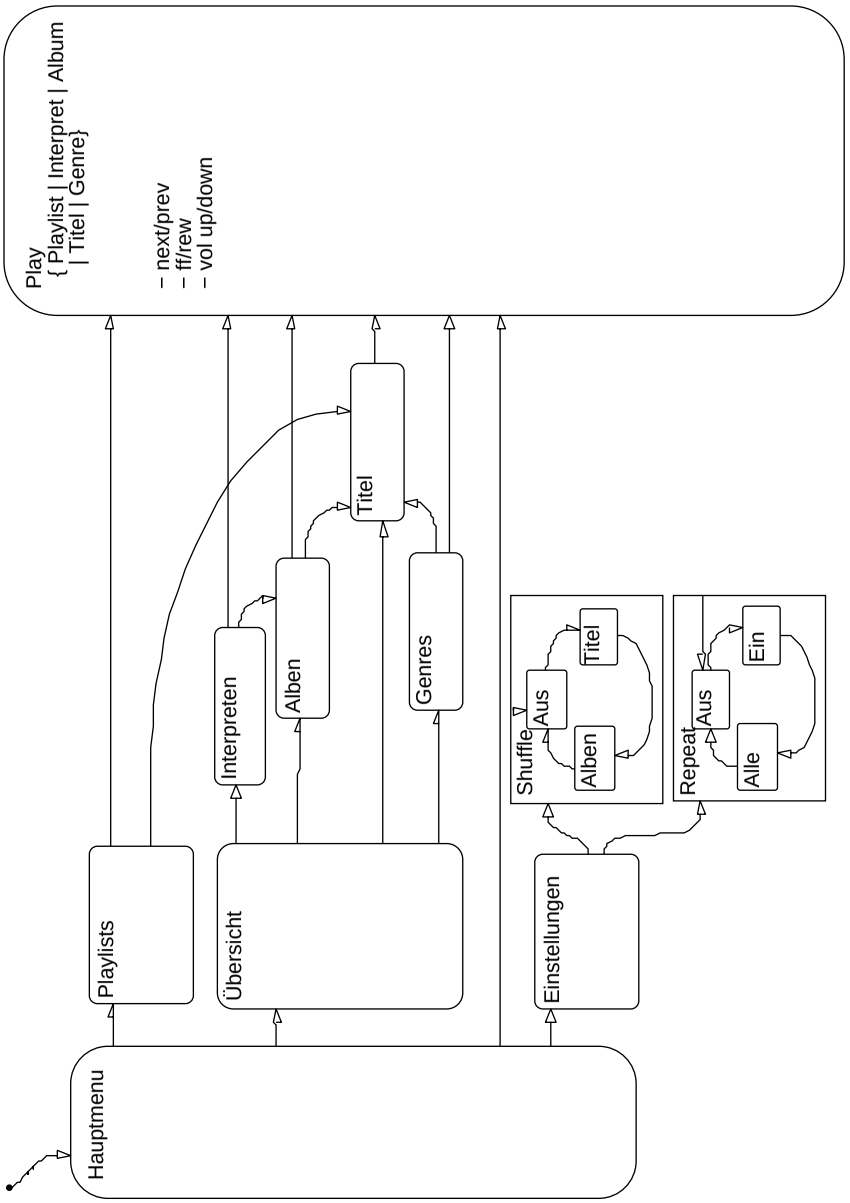
B Dialogcharts der analysierten MP3-Player



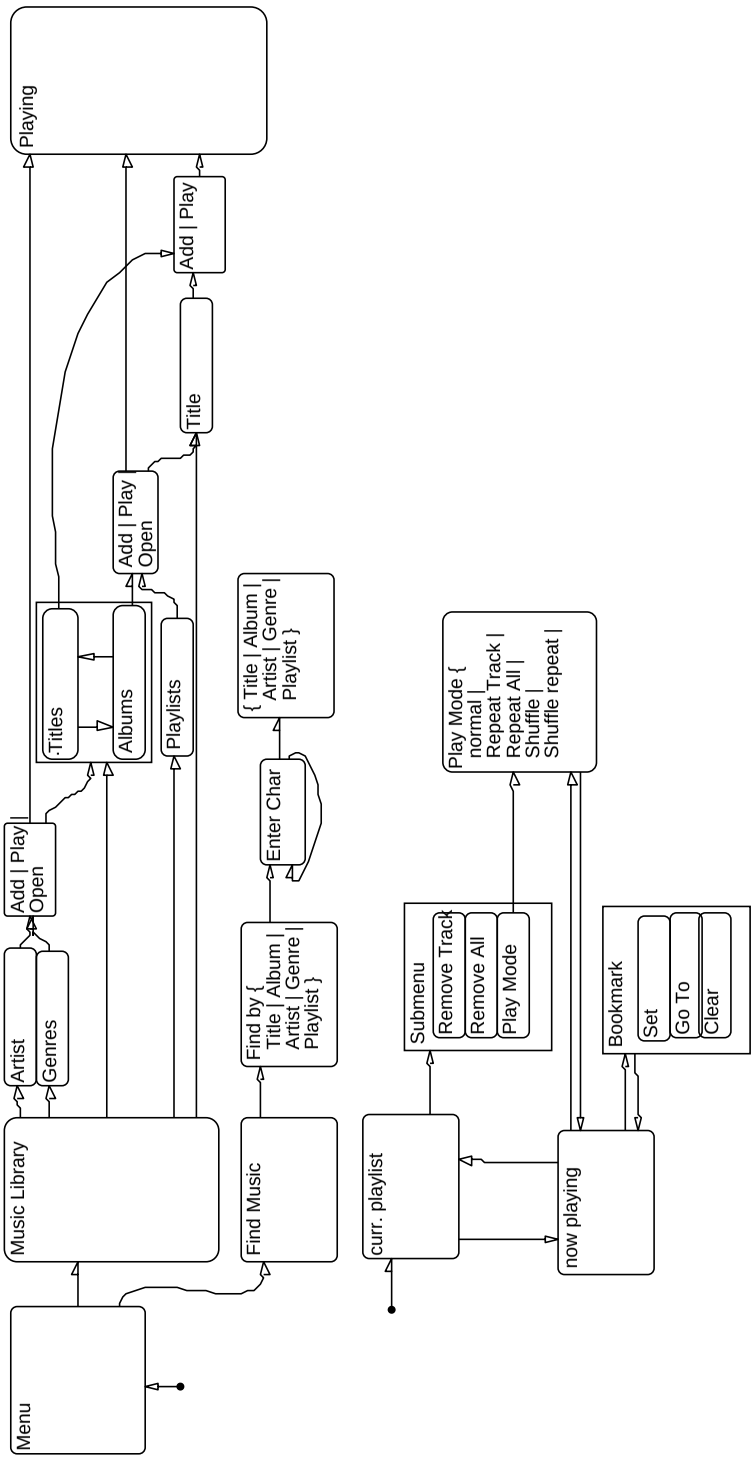
Figur 25: *Dialogchart von iRiver's iHP140*



Figur 26: *Dialogchart von Archos' Jukebox*



Figur 27: *Dialogchart von Apple's iPod*



Figur 28: *Dialogchart von Creative's Zen*

C Phonologisches Alphabet

A	Alpha
B	Bravo
C	Charlie
D	Delta
E	Echo
F	Foxtrot
G	Golf
H	Hotel
I	India
J	Juliet
K	Kilo
L	Lima
M	Mike
N	November
O	Oscar
P	Papa
Q	Quebec
R	Romeo
S	Sierra
T	Tango
U	Uniform
V	Victor
W	Whiskey
X	Xray
Y	Yankee
Z	Zulu

D Matlab Sourcecode

D.1 Main

Listing 2: *vocomp.m*

```
function [] = vocomp(ArgSpeech)
% function [] = vocomp(ArgSpeech)
%
% The Voice Controlled Media Player
% starts with additional speech (ArgSpeech = 1) or console only (ARgSpeech = 0) output

%clear all;
cd ~/src;

disp('VOCOMP - VOice COntrolled Media Player')
disp('Starting up...');

% define global variables
global gFs   % global sampling frequency
global gBits % bit depth of signals
global gS % contains all states
global gE % contains all events
global gT % contains the state transition with internal ids (not sfid!)
global gModelname
global gEventVector

% set vars
gFs = 8000;
gBits = 16;

% set simulink modelname
gModelname = 'playermodel';

% add pathes
path(path, '~/src');
path(path, '~/src/detector');
path(path, '~/src/audioproc');
path(path, '~/src/synthesizer');
path(path, '~/src/mediaControl');

% scan state machine and get states, events and transition matrix
%disp('Scanning dialog state machine...');
%[gS, gE, gT] = scanfsm(gModelname);
%gEventVector = ones(1, length(gE));
gEventVector = [];

if (ArgSpeech)
% activate and initialize speech synthesis
initSynthesizer;
end


% start gui
disp('Starting GUI...');
playerui

talk('.');


% initialize mediaControl variables
initMediaControl;

talk('Welcome to Voco m p, the Voice controlled media player.');
```

Listing 3: *playerui.m*

```
function varargout = playerui(varargin)
% PLAYERUI M-file for playerui.fig
%      PLAYERUI, by itself, creates a new PLAYERUI or raises the existing
%      singleton*.
%
%      H = PLAYERUI returns the handle to a new PLAYERUI or the handle to
%      the existing singleton*.
%
%      PLAYERUI('Property','Value',...) creates a new PLAYERUI using the
%      given property value pairs. Unrecognized properties are passed via
%      varargin to playerui_OpeningFcn.  This calling syntax produces a
%      warning when there is an existing singleton*.
%
%      PLAYERUI('CALLBACK') and PLAYERUI('CALLBACK',hObject,...) call the
%      local function named CALLBACK in PLAYERUI.M with the given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Last Modified by GUIDE v2.5 17-Jun-2004 19:46:42

% ------------------------------------------------------------------
% Begin initialization code - DO NOT EDIT
% ------------------------------------------------------------------

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @playerui_OpeningFcn, ...
                   'gui_OutputFcn',  @playerui_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% ------------------------------------------------------------------
% OpeningFcn
% ------------------------------------------------------------------

% --- Executes just before playerui is made visible.
function playerui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for playerui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% load system
global gModelname
disp('Loading System...')
load_system(gModelname);
```

```
70
global gPatSel
gPatSel = 1;

% --------------------------------------------------------------------
75 % OutputFcn
% --------------------------------------------------------------------

% --- Outputs from this function are returned to the command line.
function varargout = playerui_OutputFcn(hObject, eventdata, handles)
80 % varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

85 % Get default command line output from handles structure
varargout{1} = handles.output;

% --------------------------------------------------------------------
% StartSim Callback
90 % --------------------------------------------------------------------

% --- Executes on button press in pb_startsim.
function pb_startsim_Callback(hObject, eventdata, handles)
% hObject    handle to pb_startsim (see GCBO)
95 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% set init state
global gCurrentState
100 gCurrentState = 1;

% start simulation
global gModelname
disp('Starting Simulation...')
105 set_param(gModelname, 'SimulationCommand', 'start')
disp('Ready.')

% --------------------------------------------------------------------
% StopSim Callback
110 % --------------------------------------------------------------------

% --- Executes on button press in pb_stopsim.
function pb_stopsim_Callback(hObject, eventdata, handles)
% hObject    handle to pb_stopsim (see GCBO)
115 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global gModelname
if ~isempty(find_system('name', gModelname))
120   disp('Stopping Simulation...')
  set_param(gModelname, 'SimulationCommand', 'stop');
end

% --------------------------------------------------------------------
125 % Voice Trigger Callback
% --------------------------------------------------------------------

% --- Executes on button press in pb_trigvoice.
function pb_trigvoice_Callback(hObject, eventdata, handles)
130 % hObject    handle to pb_trigvoice (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global gOnAir
135 global gPatSel
global gPatterns
global gModelname
global gCurrentState
global gS
140 global gE
```

```
% get id of NewShortcut and OpenShortcut states
constIDOpenShortcut = find(strcmp({gS(:).name}, 'OpenShortcut'));
constIDNewShortcut = find(strcmp({gS(:).name}, 'NewShortcut'));
constIDProcShortcut = find(strcmp({gE(:).name}, 'ProcShortcut'));
constIDCmdCancel = find(strcmp({gE(:).name}, 'CmdCancel'));

% if recording is running
if gOnAir == 1

  % stop recording
  sig = record_stop;

  gOnAir = 0;
  rid = 0;

  % record new shortcut
  if gCurrentState == constIDNewShortcut
    if addPatt(sig, gPatterns(gPatSel, :)) > 0
      rid = constIDProcShortcut;
    else
      disp('ERROR in playerui.m: failed to call addPatt\n');
    end

  % compare with shortcuts on disk
  elseif gCurrentState == constIDOpenShortcut
    sid = mostLikelyShortcutPatt(sig, [gPatterns(gPatSel, :)]);
    if (sid > 0)
      loadShortcutData(sid, [gPatterns(gPatSel, :)]);
      talk('Shortcut recognized.');
      rid = constIDProcShortcut;

    else
      talk('No matching Shortcut found.');
      rid = constIDCmdCancel;

    end


  % process normal set of possible speech commands
  else
    rid = mostLikelyPatt(sig, gCurrentState, [gPatterns(gPatSel, :)]);
  end

  if rid > 0
    sf_inject_event(rid);
  end


else
  record_init
  record_start
  gOnAir = 1;
end

% --------------------------------------------------------------------
% Key Pressed Callback
% --------------------------------------------------------------------

% --- Executes on key press over figure1 with no controls selected.
function playerui_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --------------------------------------------------------------------
% Event ListBox Creation
% --------------------------------------------------------------------

% --- Executes during object creation, after setting all properties.
function lb_events_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to lb_events (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% create empty command list, we'll fill it later
set(hObject, 'String', []);

% --------------------------------------------------------------------
% Event ListBox Callback
% --------------------------------------------------------------------

% --- Executes on selection change in lb_events.
function lb_events_Callback(hObject, eventdata, handles)
% hObject    handle to lb_events (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global selection
selection = get(hObject, 'Value');

% --------------------------------------------------------------------
% Event Injection Button Callback
% --------------------------------------------------------------------

% --- Executes on button press in pb_inject.
function pb_inject_Callback(hObject, eventdata, handles)
% hObject    handle to pb_inject (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global selection
sf_inject_event(selection);

% --------------------------------------------------------------------
% Pattern ListBox Creation
% --------------------------------------------------------------------

% --- Executes during object creation, after setting all properties.
function lb_patterns_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lb_patterns (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% fill pattern profile list box
global gPatterns;
[s, w] = system('ls -1 detector/signals/');
gPatterns = [];
while (length(w) > 0)
  [s, w] = strtok(w);
  gPatterns = strvcat(gPatterns, s);
end
set(hObject, 'String', gPatterns);

% --------------------------------------------------------------------
% Pattern ListBox Callback
% --------------------------------------------------------------------
```

```
% --- Executes on selection change in lb_patterns.
function lb_patterns_Callback(hObject, eventdata, handles)
% hObject    handle to lb_patterns (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global gPatSel;
gPatSel = get(hObject,'Value');

% --------------------------------------------------------------------
% Menu: root Callback
% --------------------------------------------------------------------

function file_Callback(hObject, eventdata, handles)
% hObject    handle to file (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --------------------------------------------------------------------
% Menu: exit Callback
% --------------------------------------------------------------------

function exit_Callback(hObject, eventdata, handles)
% hObject    handle to exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global gModelname
if ~isempty(find_system('name', gModelname))
  disp('Closing System...')
  close_system(gModelname);
end

close all
clear all
disp('... done.')

% --------------------------------------------------------------------
% Status Text Creation
% --------------------------------------------------------------------

% --- Executes during object creation, after setting all properties.
function st_status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to st_status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --------------------------------------------------------------------
% scanfsm menu item callback
% --------------------------------------------------------------------

function scanfsm_Callback(hObject, eventdata, handles)
% hObject    handle to scanfsm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global gS % contains all states
global gE % contains all events
global gT % contains the state transition with internal ids (not sfid!)
global gModelname
global gEventVector

disp('Scanning FSM...');

[gS, gE, gT] = scanfsm(gModelname);
gEventVector = ones(1, length(gE));

items = {gE(:).name};
h = findobj('Tag','lb_events');
set(h, 'String', items);
```

```
disp('... done.');


% --- Executes on button press in TalkPattern.
function TalkPattern_Callback(hObject, eventdata, handles)
% hObject    handle to TalkPattern (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global selection
global gPatSel
global gPatterns
global gE
global gCurrentState

filename = ['detector/signals/', deblank(gPatterns(gPatSel, :)), '/',...
deblank(gE(selection).name), '.au'];
[sig,Fs,BITS]=auread(filename);
unix(['audioplay ', filename]);
rid = mostLikelyPatt(.05+sig, gCurrentState, [gPatterns(gPatSel, :)]);
sf_inject_event(rid);
```

Listing 4: *scanfsm.m*

```
function [StateStruct, EventStruct,  TransMat] = scanfsm(name)

    % returns [StateStruct, EventStruct, TransMat]
    % puts all states and events in StateStruct and EventStruct with fields sfid and name
    % its indices are used as internal state and event ids
    % creates the transistion matrix TransMat, with the internal ids of the states
    % in the columns and the internal ids of the transition in the rows
    % scanfsm assumes there is only one loaded system, else there is an error...
    % check for only one loaded system
```

D.2 Detector

Listing 5: *record_start.m*

```
function [] = record_start();

    % records audiostream into FIFO-pipe
```

Listing 6: *record_stop.m*

```
function [ss] = record_stop()

    % ends the recording that was started in record_start
    % returns signal of recorded sound as row vector
```

Listing 7: *mostLikelyPatt.m*

```
function [mlPattID, mlPattWord]= mostLikelyPatt(testsig, stateID, pattSet)

    % compares testsig to a set of reference patterns which depends on
    % the current state the system is in (stateID)
    % and the Pattern Set the user selected (pattSet)
    % and returns the most similar pattern (DTW)
    % uses compare2Patts, extr_fft_ceps and detect_endpoints as subfunction
```

Listing 8: *dtw.m*

```
function [dst,warpcurve,distcurve] = dtw(ref,tst,refwgt,tstwgt)

    % calculates the mean distance 'dst' and the warping curve 'warpcurve'
    % between the contours 'ref' and 'tst'. Each row of 'ref' and 'tst' must
    % contain one parameter vector. If the arguments 'refwgt' and 'tstwgt'
    % are present, the local distances are weigthed with the corresponding
    % factors.
    % the innermost loop (forward processing) is handled in the C-function
    % dtwLoopC.c for performance reasons
```

D.3 MediaControl

Listing 9: *id3Info.m*

```
function [] = id3Info

    % outputs id3-information of currently playing track (TrackPointer set in playMedia)
    % set rowvector 'catVec' to select tags for output, 1 to enable, 0 to disable category
    % (e.g. [0,1,1,0,0] to get a string containing artist and title.
    % categories are [genre, artist, album, title, path]
    % needs global playingDB set in writePlaylist
```

Listing 10: *playMedia.m*

```
function [] = playMedia(xCmd,load)

    % remotely control xmms
    % param1 : rew, play, pause, stop, fwd
    % param2 : 1 to load vcocomp playlist 'vlist.m3u', 0 to operate on current playlist
    % keeps track of currently playing media in XMMS via gTrackPointer
```

Listing 11: *selectFromDB.m*

```
function [totHits, uniqueCatHits] = selectFromDB(subSelect, cat, pat)

    % selects all lines with matching 'pat' of kategory 'cat' of DB
    % use numerical categories 1...5 for genre, artist, album, title, path
    % subSelect = 0: select from whole DB (gDB)
    % subSelect = 1: select from previously selected subset (gSubDB)
    % returns a subset of 'db' as a new array and its length and the first Hit of the 'cat'-column
```

Listing 12: *selectionOutput.m*

```
function [selectionStr] = selectionOutput(subSelect, cat)

    % prepares selection for output into 2 classes (few hits, many hits)
    % outputs hits depending on class
    % uses selectFromDB.m as subfunction
```

Listing 13: *writePlaylist.m*

```
function [] = writePlaylist

    % writes previously defined subselection into m3u-Playlist for XMMS
```

Listing 14: *StrAssembly.m*

```
function StrAssembly(code)

% receives the ascii code from the corresponding state in the
% stateflow model and appends it to the global variable gAssemblerStr
```

E Aufgabenstellung

Institut für Technische Informatik und Kommunikationsnetze
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Gruppe für Sprachverarbeitung

Sommersemester 2004

SEMESTERARBEIT

für

Haegler Simon und Ulmer Andreas

Betreuer: H. Romsdorfer ETZ D97.5
Stellvertreter: Dr. B. Pfister ETZ D97.6

Ausgabe: 29. März 2004
Abgabe: 2. Juli 2004

Sprachsteuerung für Audiogeräte

Einleitung

Das Benutzerinterface von aktuell verfügbaren, mobilen Audiogeräten mit grosser Datenkapazität, wie z.B. MP3-Player, ist zur Zeit nicht wirklich befriedigend gelöst. Der Trend zu immer kleineren Geräten einerseits und zu einer stetig wachsende Speicherkapazität (und einer damit verbundenen grösseren Anzahl von speicherbaren Musikstücken) andererseits macht die Bedienung solcher Geräte (z.B. die Navigation durch die Alben oder das Auffinden bestimmter Musikstücke) via herkömmlicher manueller (plus eventueller visueller) Schnittstelle sehr langwierig und unübersichtlich. Durch die zusätzliche Verwendung der auf vielen derartigen Geräten ohnehin bereits vorhandenen Audio-Schnittstelle (Kopfhörer und Mikrofon) wäre es grundsätzlich möglich, die Bedienung über eine lautsprachliche Kommunikation zu gestalten.

Problemstellung

Aufgabe dieser Arbeit ist es, in einem ersten Schritt ein Konzept für die Steuerung eines solchen Gerätes mittels Spracherkennung für Benutzereingaben und Sprachsynthese für das Benutzer-Feedback zu erstellen. Hierbei sollen mögliche Bedienungsvarianten (sog. Use Cases) eines solchen Gerätes mittels Sprachsteuerung, die daraus entstehenden Randbedingungen für deren technische Realisierung, und schliesslich die Vor- und Nachteile der

jeweiligen Realisierung dargestellt werden.

Mögliche Punkte, die bei der Konzepterstellung zu bedenken sind, wären:

- Interface-Gestaltung (nur Audio, Audio und manuell/visuell, ...)
- Dialogsteuerung und -beschreibung (VoiceXML, SALT, ...)
- Mikrofon-Typ
- Sprachdetektion (Push-To-Talk Button, Utterance Detection, ...)
- Spracherkennungsmethoden (Mustererkennung, statistische Spracherkennung, ...)
- Sprachausgabemethoden (aufgezeichnete Voice Prompts, Sprachsynthese, ...)
- Benutzer-Setup (Adaptierung des Spracherkenners, Einstellung von Benutzerpräferenzen, ...)

Nach Auswertung möglicher Realisierungsalternativen soll ein Prototyp (vorzüglich in Matlab) implementiert werden. Anhand dieses Prototyps soll die Brauchbarkeit (ersichtlich z.B. durch eine Verbesserung des Bedienungskomfort gegenüber einer rein manuellen Schnittstelle) des erstellten Konzepts getestet werden. Auf Einschränkungen, die durch die Hardware Performance bzw. durch Software-Limitierungen eines bestimmten Gerätetyps bedingt sind, soll in dieser Arbeit nicht näher eingegangen werden.

Aufgaben

1. Einarbeitung in die Literatur und in die Arbeitsumgebung (für eine Einführung in die Sprachverarbeitung [1, 2], hier vor allem Übung 13. Modellierung und Abarbeitung von Dialogen sind z.B. in [3, 4, 5, 6] beschrieben.).
2. Entwicklung eines geeigneten Konzepts inklusive möglicher Use Cases.
3. Bewertung der technischen Realisierungsmöglichkeiten.
4. Implementierung des erstellten Konzepts als Prototyp in Matlab.
5. Testen des Prototyps mittels der definierten Use Cases unter verschiedenen Randbedingungen (z.B. mit/ohne Nebengeräusche, Variation der Anzahl von Musikstücken, ...) und Ermittlung der Bedienungseffizenz (z.B. anhand der Zeitdauer vom Start der Interaktion bis zum korrekten Ende, oder der Anzahl der Bedienungsschritte, oder der Relation der korrekt zu den inkorrekt durchgeführten Anweisungen, ...)
6. Die ausgeführten Arbeiten und die erhaltenen Resultate sind in einem Bericht (siehe Richtlinien unter "www.tik.ee.ethz.ch/~spr/SADA/") zu dokumentieren, der in zwei Exemplaren abzugeben ist, wovon eines Eigentum des Instituts bleibt.

Literaturverzeichnis

[1] B. Pfister und H.-P. Hutter. *Sprachverarbeitung I.* Vorlesungsskript für das Wintersemester 2003/2004, Departement ITET, ETH Zürich, 2003.

[2] B. Pfister, H.-P. Hutter und C. Traber. *Sprachverarbeitung II.* Vorlesungsskript für das Sommersemester 2003, Departement ITET, ETH Zürich, 2003.

[3] J. Alexandersson, R. Engel, M. Kipp, S. Koch, U. Küssner, N. Reithinger, and M. Stede. Modeling negotiation dialogs. In W. Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer, 2000.

[4] M. Kipp, J. Alexandersson, R. Engel, and N. Reithinger. Dialog processing. In W. Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer, 2000.

[5] I. Pitt and A. Edwards. *Design of speech based devices. A practical guide*. Springer, London, 2003.

[6] E. Maracke. *VoiceXML 2.0 : Konzeption, Projektmethodik und Programmierung von Sprachdialogystemen*. Galileo Press, Bonn, 2003.

[7] B. Pfister. *Richtlinien für Semesterarbeits- und Diplomarbeitsberichte*. www.tik.ee.ethz.ch/~spr/SADA/, 2004.

Zürich, den 30. März 2004 Prof. Dr. L. Thiele