Friedberger Java-Sprach-Tools
Version 1.0

Stephan Euler
FH-Giessen–Friedberg
Fachbereich MND

29. Februar 2008

Dieses Skript wurde mit LaTeX 2_ε und TeX(Version 3.141592) geschrieben. Eingesetzt wurde die integrierte Benutzeroberfläche WinEdt 5.3 zusammen mit MiKTeX Version 2.2.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Mit HTK steht eine leistungsfähige Umgebung zur Entwicklung von Hidden-Markov-Modellen zur Verfügung. Als Ergänzung dazu wurden Anwendungen für die folgenden Themen realisiert:

- DTW-Erkenner
- Anzeige und Bearbeitung von Sprachaufnahmen
- Erstellen von Wortnetzen
- Dialogsteuerung

Alle Anwendungen sind in der Programmiersprache Java geschrieben und damit weitgehend plattformunabhängig. Alle Programme sind frei verfügbar. Soweit möglich wurden die Standards von HTK berücksichtigt. Insbesondere werden in dem Programm zur Dialogsteuerung der HTK-Erkenner verwendet. Im Folgenden werden die einzelnen Anwendungen kurz vorgestellt.

Kapitel 2

DTW-Erkenner fbdtw

Die Anwendung `fbdtw` stellt eine einfache Realisierung eines Erkennungssystems auf der Basis des DTW-Verfahrens dar. Bei der Entwicklung wurde der Schwerpunkt auf Offenheit und Flexibilität gelegt.

Um ein Wort zu trainieren, muss ein Benutzer eine vorgegebene Anzahl (Standard 3) von Referenzäußerungen aufsprechen. Jede aufgezeichnete Äußerung wird unmittelbar graphisch dargestellt. Der Benutzer kann dann offensichtlich schlechte Referenzen – beispielsweise bei einem Fehler der Wortgrenzendetektion – ablehnen und neu aufsprechen.

Jede Äußerung wird in einer eigenen Datei gespeichert. Der Name der Datei wird aus dem Wort beziehungsweise der Wortfolge, dem Trennzeichen `#` und einem fortlaufenden Zähler gebildet. Ein Beispiel dafür ist:

	physik#2.wav		
physik	#	2	.wav
Wort	Trennzeichen	Zähler	Dateiendung

In diesem Fall wurde das Wort *Physik* gesprochen. Die Datei enthält die zweite Äußerung dieses Wortes des aktuellen Sprechers. Mehrere Wörter in einer Äußerung werden im Dateinamen durch das `_`-Zeichen getrennt. Im Programm `fbdtw` werden alle Audiodaten unkomprimiert im WAV-Format abgespeichert.

Das Programm unterstützt mehrere Benutzer. Für jeden Benutzer wird ein eigenes Verzeichnis angelegt. Alle Referenzen eines Sprechers werden dann in sein Verzeichnis geschrieben. Die verschiedenen Verzeichnisse befinden sich in einem gemeinsamen Hauptverzeichnis (Standardname `userdata`). Bild 2.1 zeigt den Aufbau an einem Beispiel mit zwei Benutzern.

Dieses Konzept ist dafür vorgesehen, mehrere Benutzer zu verwalten. Es ist darüber hinaus geeignet, ein größeres Vokabular in Wortgruppen zu unterteilen. In einer konkreten Anwendung sind dann zu einem Zeitpunkt in Abhängigkeit vom Dialogzustand nur einige der Teilgruppen aktiviert. In `fbdtw` können im Erkennungsmodus beliebig viele der vorhandenen Benutzer oder Wortgruppen

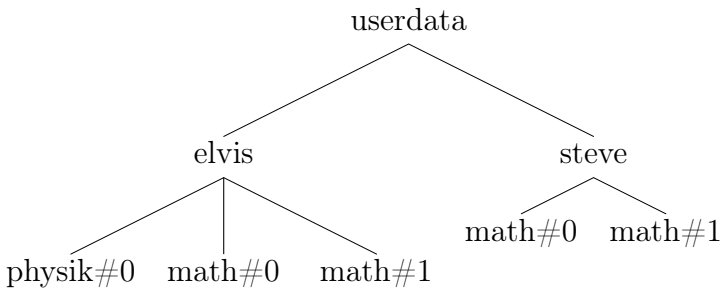

Abbildung 2.1: Verzeichnisstruktur in `fbdtw`, Beispiel mit zwei Benutzern (elvis und steve)

gleichzeitig verwendet werden. Während des Trainings darf allerdings nur genau ein Benutzer aktiv sein, da sonst eine eindeutige Zuordnung nicht möglich wäre.

Die gesamte Information zu den Referenzen ist in den Namen der Verzeichnisse und der Audiodateien kodiert. Daher ist die Verwaltung sehr einfach. Neben den von der Anwendung angebotenen Möglichkeiten kann man auch direkt die Dateien kopieren, löschen oder umbenennen. So kann man beispielsweise einen Benutzer von einem anderen System übernehmen, indem man das entsprechende Unterverzeichnis kopiert.

Bild 2.2 zeigt die Anwendung. Sie besteht aus einem Hauptfenster mit der Benutzerverwaltung und einem Knopf zur Aktivierung der Erkennung sowie einigen Informationsfenstern. Es sind in diesem Fall drei Benutzer eingerichtet (`steve`, `test` und `tmp`), von denen aber nur der erste aktiviert ist. Sein Vokabular besteht aus 11 Städtenamen. Die letzte Testäußerung ist im Fenster unter dem Hauptfenster dargestellt. Erkannt wurde *Berlin*. Zum Vergleich sind alle Abstandswerte in einem Ergebnisfenster angegeben.

Standardmäßig werden auch alle Testäußerungen abgespeichert. Dazu wird ein Verzeichnis `autosave` mit Unterverzeichnissen für die einzelnen Sprecher angelegt. Ist nur ein Sprecher aktiv, wird die Testäußerung in das zugehörige Verzeichnis gelegt. Ein spezielles Unterverzeichnis *unknown* ist für nicht eindeutig zuzuordnende Äußerungen vorgesehen. In allen Fällen wird der Dateiname in der Form `in#n.wav` gebildet, wobei `n` wieder ein fortlaufender Zähler ist. Eine Datei mit dem gleichen Namen und der Endung `mlf` enthält das Erkennungsergebnis.

Abbildung 2.2: Das DTW-Programm `fbdtw`

Kapitel 3

FBVIEW

3.1 Grundlagen

Ein zentrales Werkzeug für die verschiedenen Experimente ist `fbview`. Ursprünglich zur Darstellung von Audiodateien mit dem Schwerpunkt auf den schnellen Zugriff auf eine lange Liste solcher Dateien entwickelt, wurden im Laufe der Zeit mehr und mehr Möglichkeiten integriert [Eul05]. Die Basisfunktionen sind:

- Schneller Zugriff auf Dateien aus einer Liste
- Gleichzeitige Darstellung mehrerer Dateien oder mehrerer Kanäle einer entsprechenden Datei.
- Unterstützung für verschiedene Dateiformate sowohl für Audiodaten als auch Transkriptionen.
- Funktionen zum Bearbeiten der Audiodaten.
- Eingeben und Verändern von Transkriptionen.

Darüber hinaus sind diverse Analysemöglichkeiten sowie eine Schnittstelle zum Datenaustausch mit anderen Anwendungen integriert. Bild 3.1 zeigt das Programm bei einem Aufruf ohne Argumente. In diesem Fall werden vier, zunächst leere Fenster für Sprachdaten angelegt. Die Dateiliste wird dann über ein Dateiauswahl-Menü oder durch *Drag & Drop* gefüllt.

Bild 3.2 zeigt ein Beispiel nach dem Laden von Dateien. Im zentralen Fenster sind die Signale aus vier Dateien dargestellt. Ein weiteres Fenster enthält Informationen über die vier Dateien. In dem Beispiel handelt es sich um Dateien im WAV-Format und in dem Fenster sind die Informationen aus dem jeweiligen Dateikopf zusammengestellt. In dem Transkriptions-Fenster wird als Überschrift der Namen der ersten dargestellten Datei zusammen mit Angaben zur Dauer und zum Wertebereich dargestellt. Darunter befinden sich die Transkriptionen für die aktuellen Dateien.

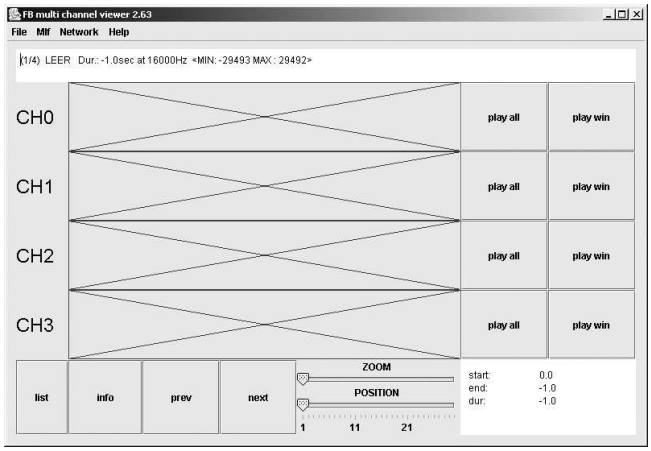

Abbildung 3.1: Das Programm `fbview`

Nur die in den Fenstern wiedergegebenen Dateien sind tatsächlich geöffnet. Wechselt man zu anderen Dateien, so werden eventuelle Änderungen gespeichert und die Dateien geschlossen. Dadurch können ohne Probleme auch sehr lange Dateilisten bearbeitet werden.

In einem eigenen Fenster ist die Liste aller Dateien eingetragen. Dieses Fenster dient zur schnellen Navigation durch die Dateien. Wählt man eine Datei aus der Liste aus, so wird die mit damit beginnende Gruppe von in diesem Fall vier Dateien aktiviert. Alternativ kann man über die `prev`- und `next`-Schalter zur jeweils nächsten oder vorherigen Gruppe wechseln. Die Dateiliste kann über ein Kontextmenü angepasst werden. Im einzelnen kann jede Datei

- um eine Position hoch oder runter geschoben werden
- an den Anfang oder an das Ende gesetzt werden
- aus der Liste gelöscht werden (betrifft nur den Eintrag, die Daten bleiben erhalten)

3.1.1 Interne Daten

In der Regel werden die dargestellten Daten aus Dateien gelesen. Daneben ist es allerdings auch möglich, neue Datensätze dynamisch zu erzeugen. Diese werden intern abgespeichert. Als Kennung werden fortlaufende Nummerierungen in der Form `#0`, `#1`, `#2`, ... vergeben. Unter diesen Namen erscheinen die Daten in der Liste und können dort ausgewählt werden. Standardmäßig wird bei Programmende gefragt, ob und wenn ja unter welchen Namen diese Daten gespeichert werden sollen. Diese Verhalten kann über die Eigenschaft `saveInternalWaves` in der Datei `fbview.ini` geändert werden. Derzeit werden zwei Methoden unterstützt:

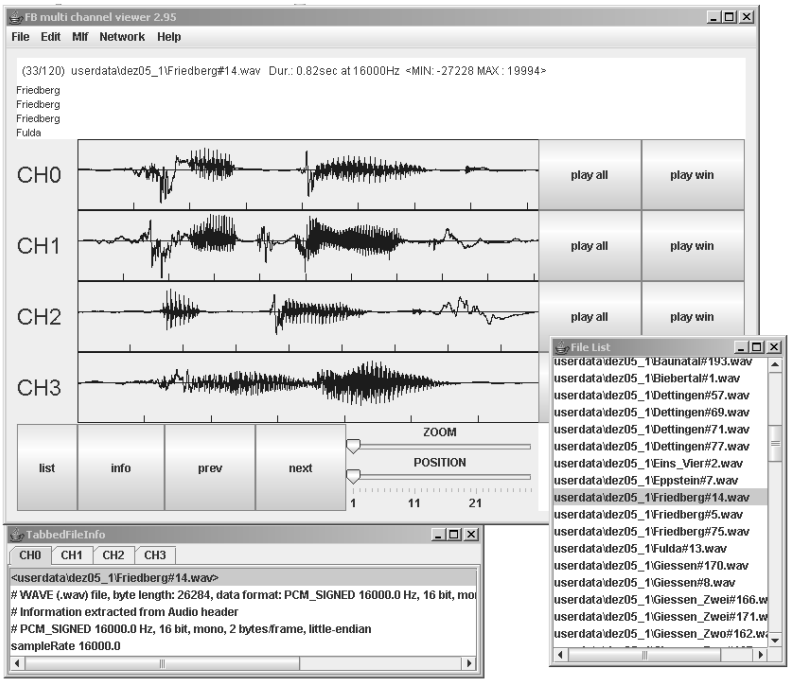

Abbildung 3.2: Das Programm `fbview` nach Laden von Dateien

- `auto`: Die Daten werden automatisch in Dateien `#0.wav`, `#1.wav`, `#2.wav`, ... gespeichert.
- `never`: Die Daten werden ohne nachzufragen nicht abgespeichert.

3.2 Dateiformate

Das Programm benutzt die Standardbibliotheken von Java zum Lesen der Audiodateien. Dabei versucht es, jede Datei als audio input stream zu öffnen. Falls es sich um ein von den Standardmethoden unterstütztes Format handelt, werden anschließend die Details des Formats über entsprechende Methodenaufrufe abgefragt. Anderenfalls, d. h. wenn ein Ausnahmefehler vom Typ UnsupportedAudioFile auftritt, sucht `fbview` nach einem Dateikopf gemäß dem NIST SPHERE-Format. Wird auch dieser nicht gefunden, wird die Datei als einfache Folge von Abtastwerten behandelt. Mit diesem Vorgehen werden automatisch die verschiedensten WAV-Formate erkannt, allerdings unterstützt `fbview` bis jetzt nur Monodateien mit 16 bit PCM-Werten. Neben den Mono-Formaten sind zwei Spezialformate für mehrkanalige Aufnahmen implementiert:

- 4-kanaliges Dateiformat wie in dem Forschungsprojekt SpeechDat-Car definiert [MLD+00][Dra99]
- ASCII-Dateien: jede Spalte wird als eigener Kanal interpretiert.

Es ist allerdings bisher nicht möglich, in einem Aufruf ein- und mehrkanalige Dateien gleichzeitig zu behandeln.

Für die Speicherung der Transkription gibt es verschiedene Vorgehensweisen. Sie kann im Dateikopf in der Audiodatei selbst eingetragen oder in getrennten Dateien gehalten werden. In HTK können die Transkriptionen zu mehreren Dateien gemeinsam in so genannten *master label files* (MLFs) abgelegt werden. Obwohl es auch andere Formate wie NIST unterstützt, ist `fbview` für die Verwendung von MLFs optimiert. So ist es beispielsweise über einen entsprechenden Menüpunkt möglich, aus einer Liste von Dateien, die der oben beschriebenen Namenskonvention entsprechen, ein MLF zu erzeugen. Ein MLF kann Informationen über einzelne Segmente enthalten. In diesem Fall werden die Grenzen in den Sprachdaten eingezeichnet und mit der zugehörigen Beschriftung versehen.

In den meisten Fällen werden die Dateiformate automatisch erkannt und die zugehörigen Transkriptionen gefunden. Allerdings können nicht alle möglichen Kombinationen abgedeckt werden, und einige Fälle führen zu einer fehlerhaften Darstellung. Dann ist es notwendig, über Optionen beim Programmaufruf die richtigen Parameterwerte einzustellen.

3.3 Transkriptionen

Die Transkriptionen werden in einem Bereich über den Audiodaten angezeigt. Es handelt sich um ein Java-Standardelement zur Textdarstellung. Damit werden alle üblichen Methoden zur Textbearbeitung und insbesondere auch der Austausch mit anderen Anwendungen per Ausschneiden und Einfügen unterstützt.

Um die mühsame und zeitaufwändige Arbeit des Verschriftens zu vereinfachen, ist ein Vokabularfenster vorgesehen. Dieses Fenster enthält für jedes zuvor in einer Liste angegebene Wort einen Schalter. Wird dieser Schalter betätigt, so wird automatisch das zugehörige Wort in die Transkription der zuletzt verwendeten Aufnahme übernommen. Speziell bei kleinen Wortschätzen ermöglicht diese Eingabe eine rasche Bearbeitung längerer Listen von Aufnahmen. Gleichzeitig werden durch die automatische Übergabe Schreibfehler verhindert.

3.4 Signalanalyse

Für eine dargestellte Wellenform kann eine spektrale oder cepstrale Analyse des markierten Bereichs ausgeführt werden. Das Ergebnis wird für jedes Signal in einem eigenen Fenster dargestellt. Bild 3.3 zeigt ein Beispiel. Sowohl das FFT- als auch das LPC-Spektrum sind eingetragen. Es ist möglich, einige Spektren für Vergleiche mit anderen Spektren als Referenz festzuhalten.

Neben der direkten spektralen Analyse ist auch eine Korrelationsanalyse zwischen den einzelnen Signalen realisiert. Weiterhin kann für den selektierten Ab-

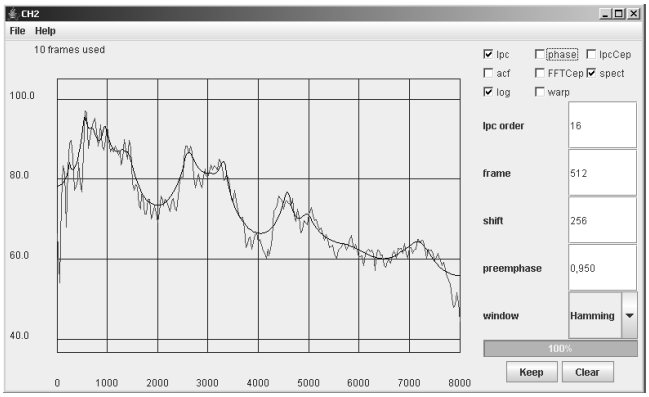

Abbildung 3.3: Die Spektralanalyse in `fbview`

schnitt der Wertebereich sowie ein Histogramm angezeigt werden.

3.5 Statistische Auswertung

3.6 Externe Schnittstelle und Erweiterungen

3.6.1 TCP-Schnittstelle

Andere Anwendungen können mit `fbview` Daten über eine TCP-Schnittstelle austauschen. Als Portadresse ist der Wert 1958 eingestellt. Folgende Befehle sind derzeit implementiert:

- NEW s_1 s_2 s_3 ... s_n ;
  Aus den n Zahlenwerten wird ein internes Audio-Objekt erzeugt. Das Objekt wird an die Liste der Dateien angehängt. Falls gewünscht, kann dieses Objekt später in einer Datei gespeichert werden.

- GET
  Nach Erhalt dieses Befehls schickt `fbview` die Abtastwerte der Aufnahme im obersten Fenster zurück.

- SR r
  Die Abtastrate für die weiteren NEW-Befehle wird auf den Wert r gesetzt.

Bisher wird nur die Darstellung der Zahlen als ASCII-Werte unterstützt. Damit ist eine große Flexibilität gegeben. Bei größeren Aufnahmen entstehen allerdings relativ lange Übertragungszeiten. Sofern erforderlich, kann zukünftig die Schnittstelle leicht auf die effiziente Übertragung von Binärwerten umgestellt werden.

3.6.2 FBGenerator

Ein einfaches Beispiel für die Verwendung der TCP-Schnittstelle ist das Programm `FBGenerator`. Mit diesem Programm können verschiedene Testsignale erzeugt werden. Die Signale werden dann an `fbview` gesendet und dort als neues Objekt eingebaut. Bild 3.4 zeigt die graphische Oberfläche mit den diversen Einstellmöglichkeiten.

Abbildung 3.4: Graphische Oberfläche für `FBGenerator`

3.6.3 Erweiterungen

Die Darstellung und Bearbeitung der Audiodaten kann durch zusätzliche Module erweitert werden. Dazu können entsprechende Java-Klassen dynamisch eingebunden werden. Die Liste der externen Module wird in der Konfigurationsdatei über die Eigenschaft `moduls` festgelegt. As Beispiel werden mit dem Eintrag

```
moduls=Diff:Mittel
```

zwei Klassen mit den angegebenen Namen `Diff` und `Mittel` eingebunden. Sie können über automatisch generierte Einträge im Kontextmenü der Audio-Darstellungen aufgerufen werden. Solche externen Klassen müssen das Interface XModul (Bild 3.5) implementieren. Dazu werden zwei Methoden benötigt: `setWaveView` um eine Referenz auf das aktuelle Darstellungsobjekt zu setzen und `process` für die eigentliche Verarbeitung. Wählt man den Befehl für das Modul aus, so wird ein Instanz der Klasse erzeugt und die beiden Methoden werden aufgerufen. Innerhalb der Methode `process` kann man über das `waveView`-Objekt auf das Audiosignal sowie einige Eigenschaften zugegriffen werden.

Wichtig ist, dass Java während der Laufzeit die zusätzlichen Klassen findet. Eine Möglichkeit ist, die Klassen zusätzlich in das Archiv `fbview.jar` zu packen. Einfacher ist es, die Anwendung mit einem Befehl in der Form

```
java -cp ".;fbview.jar" SpdcShow
```

```
public interface XModul {
    public void setWaveView( WaveView waveView );
    public void process();
}
```

Abbildung 3.5: Interface `XModul`

zu starten. Dabei wird direkt die Hauptklasse `SpdcShow` aufgerufen und das Archiv mit im Klassenpfad aufgeführt. In diesem Beispiel ist zusätzlich noch das aktuelle Verzeichnis angegeben, so dass Erweiterungsklassen auch hier gefunden werden.

Als erstes Beispiel betrachten wir eine Klasse `Mittel` zur Subtraktion des Mittelwerts im aktuellen Signal. Der Java-Code ist in Bild 3.6 wiedergegeben. In der Methode `process` wird zunächst eine Referenz `samples` auf das aktuelle Audiosignal angelegt. Die eigentliche Verarbeitung besteht aus zwei Schleifen zur Ermittlung und Subtraktion des Mittelwerts. Mit dem Aufruf der Methode `setSamplesHaveChanged` wird das Darstellungsobjekt über eine Änderung an den Signalwerten informiert. Abschließend wird mit `repaint` die Darstellung aktualisiert.

In der Erweiterung `Mittel` wird das vorhandene Signal überschrieben. Über die TCP-Schnittstelle ist es darüber hinaus möglich, neue Signale zusätzlich darstellen zu lassen. Als Beispiel wird in der `process`-Methode in Bild 3.7 die Differenz aufeinanderfolgender Werte bestimmt und als neues, internes Signal angelegt.

Über die Referenz auf das `WaveView`-Objekt können weitere Darstellungsdetails verändert werden. Als Beispiel wird mit den Zeilen

```
int[] x      = { samples.length / 2 };
String[] lab = {"Halbzeit" };
waveView.setXGrid( x, lab );
```

in der Mitte des Signals eine Markierung mit dem passenden Text gesetzt. Diese Beschriftung ist allerdings nur eine Eigenschaft des Darstellungsobjektes. Wird für das Audiosignal durch Vor- und Zurückblättern in der Dateiliste eine neue Darstellung erzeugt, so ist diese Markiert verloren. Als weiteres Beispiel wird durch

```
waveView.setSelectionTime(
   waveView.getWave().getDuration() * .25,
   waveView.getWave().getDuration() * .75 );
```

der Bereich in der Mitte des Signals ausgewählt. Schließlich kann ein externes Modul über eine Aufruf der Methode `actionPerformed` diverse Ereignisse auslösen. So wird durch

```
public class Mittel
implements XModul
{
   private WaveView waveView = null;

   public void setWaveView( WaveView waveView ) {
     this.waveView = waveView;
   }

   public void process() {
      short[] samples = waveView.getWave().getSignal();

      double m = 0;
      for(int j=0; j<samples.length; j++ ) {
          m += samples[j];
      }
      short mittel = (short) (m / samples.length) ;
      for(int j=0; j<samples.length; j++ ) {
          samples[j] -= mittel;
      }
      waveView.getWave().setSamplesHaveChanged();
      waveView.repaint();
   }
}
```

Abbildung 3.6: Klasse `Mittel` zur Subtraktion des Mittelwerts

```
public void process() {
   short[] samples = waveView.getWave().getSignal();
   String host   = "localhost";
   int    port   = 1958;

   short[] diff = new short[samples.length];
   for(int j=0; j<samples.length - 1; j++ ) {
       diff[j] = (short) ( samples[j+1] - samples[j]);
   }

   try {
       Socket server = new Socket(host, port );
       PrintWriter out =
             new PrintWriter(
               server.getOutputStream(), true );

       out.println( "NEW " );
       for( int i=0; i<diff.length; i++ ) {
          out.print( diff[i] + " " );
       }

       out.println( ";");
       out.close();
       server.close();
   } catch( IOException e ) {
       System.err.println( e );
   }
}
```

Abbildung 3.7: Erweiterung zur Berechnung des Differenzsignals

Tabelle 3.1: Objekt-Modell

Klasse	Zugriffsmethoden	Beschreibung
WaveView	Wave getWave()	Wave-Objekt
	setXGrid(int[] x)	Setzt Markierungen
	setXGrid(int[] x, String[] s)	zusätzliche Beschriftung
	setSelectionTime(float a, float e)	Auswahlbereich
Wave	short[] getSignal()	Feld mit Abtastwerten
	float getSampleRate()	Abtastrate
	double getDuration()	Dauer in Sekunden
	setSamplesHaveChanged()	Zähler für Änderungen am Signal

```
waveView.actionPerformed(
    new ActionEvent( this, 0, WaveView.cutSelection ) );
```

der ausgewählte Bereich gelöscht.

Kapitel 4

Netzwerk-Editor ShowGraph

Die Grammatiken zur Erkennung können für die HTK-Programme als extended Backus-Naur Form definiert werden. Allerdings müssen sie zur eigentlichen Erkennung noch in Erkennungsnetzwerke umgesetzt werden. Die HTK-Formate unterstützen jedoch nicht alle für die Anwendung mit VoiceXML gewünschte Möglichkeiten. Daher wurde ein Programm zur direkten Erstellung von Netzwerken entwickelt. Dieses Programm - ShowGraph - ermöglicht eine komfortable, graphische Bearbeitung von Netzwerken.

Als Beispiel ist in Abbildung 4 ein Netzwerk für die Eingabe von Schachzügen dargestellt. Zur besseren Übersicht und Modularität sind drei Teilnetze piece, file und rank definiert. Die Anwendung zeigt in einer Wortliste die verwendeten Wörter mit ihren Häufigkeiten. Mit einem Graph Tester können zufällige Sätze aus dem Netzwerk erzeugt werden. Außerdem erlaubt diese Komponente zu prüfen, ob vorgegebene Sätze von dem Netzwerk abgedeckt sind.

Kapitel 5

VoiceXML Interpreter

Zum Testen einfacher VoiceXML-Dialoge wurde ein entsprechender Interpreter realisiert. Die Anwendung besteht aus einer Anzahl von Java-Klassen. Spracheingaben und Ausgaben erfolgen über die Audio-Schnittstelle von Java. Für die Spracherkennung wird das HTK-Tool `HVite` eingesetzt. Die optionale Sprachsynthese erfolgt entweder mittels MBROLA [DPP+96] oder – allerdings nur in Englisch – FreeTTS [WLK02] . Bild 5.1 zeigt die graphische Oberfläche mit dem Ausgabefenster für die Dialogmeldungen und die Erkennungsergebnisse.

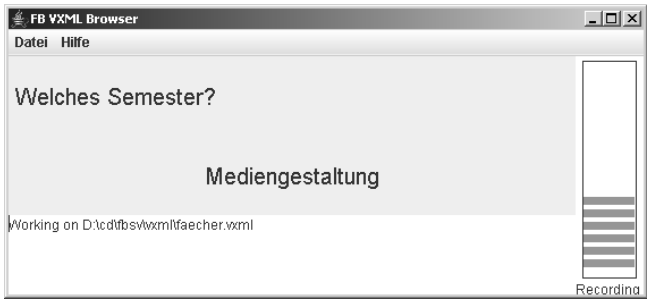

Abbildung 5.1: Graphische Oberfläche für `FBSV`

Das Hauptziel der Entwicklung war ein einfaches, leicht einsetzbares Programm zur Demonstration grundlegender Möglichkeiten von VoiceXML. Daher wurde Wert gelegt auf:

- Freie Verfügbarkeit
- Verwendung auch als Stand-Alone (ohne Server-Anbindung)
- Darstellung des Dialogablaufs in einem Ausgabefenster
- Umfangreiche Protokollierung
- Erweiterbarkeit

bbildung 5 zeigt FBSV mit einem Beispieldialog. Während eines Dialogs können optional die Äußerungen abspeichert werden. Diese Sprachdaten werden ebenfalls gesammelt und nach entsprechender Aufbereitung dem Trainingsvorrat hinzugefügt.

5.1 Beispiel Ergebnismeldung

Semantische Tags sind ein wichtiges Element, um in VoiceXML flexible Dialoge und insbesondere solche mit „gemischter Initiative“ zu realisieren. Im HTK-Format für Netzwerke sind semantische Tags zwar vorgesehen, aber das zur Spracherkennung eingesetzte HTK-Programm HVite verwendet sie nicht. Um dieses Problem zu lösen, wurde im Programm FBSV eine zweistufige Strategie realisiert. Zunächst erfolgt die Erkennung mit HVite. Für die erkannte Wortfolge werden dann die zugehörigen Tags ermittelt. Als weitere technische Schwierigkeit akzeptiert HVite keine Teilnetze. Daher muss aus dem allgemeinen Netzwerk zunächst mit dem HTK-Programm HBuild durch Expansion der Teilnetze ein entsprechendes Format erzeugt werden.

Die Vorgehensweise soll an einem Beispiel für die Eingabe von Sportergebnissen gezeigt werden. In Abbildung 6 ist ein Teilnetz für die Begegnungen eines Spieltages dargestellt. Das Teilnetz gegen enthält optionale Formulierungen wie spielt gegen oder gewinnt gegen. Wie an dem ausgewählten Knoten Wiesbaden beispielhaft zu sehen ist, tragen die Vereine Tags für heim und gast.

In VoiceXML spezifiziert dann der folgende Abschnitt den Dialog:

```
<form id="hessenliga">
<grammar src="grammar\hessenliga.slat" />
<initial name="initial">
<prompt timeout="6s">
Bitte geben Sie Ihr Ergebnis aus der Hessenliga ein.</prompt> </initial>
<field name="heim">
<prompt> Bitte geben Sie zunächst die Mannschaften an. </prompt>
<grammar src="grammar\hessenliga.slat" /> </field>

<field name="hp">
<prompt>Wie hat <value expr="heim"/>
gegen <value expr="gast"/> gespielt? </prompt>
<grammar src="grammar\ergebnis.slat" /> </field>

<field name="bestaetigung">
<prompt>    Ich habe verstanden:
<value expr="heim"/> spielt gegen <value expr="gast"/>
<value expr="hp"/> zu <value expr="8-hp"/>.
Ist das richtig? </prompt>
```

```
<grammar src="grammar\janein.slat" /> </field>
```

Der Dialog beginnt mit einer allgemeinen Eingabe-Aufforderung. Der Benutzer kann in einer Eingabe das vollständige Ergebnis angeben (*Wiesbaden gegen Fulda Drei zu Fünf*). In diesem Fall werden alle Felder gefüllt und der Dialog wird mit der Bestätigung fortgesetzt. Fehlen noch Angaben, so werden diese gezielt abgefragt. Bei der vorgesehenen Sportart sind in jedem Spiel genau 8 Punkte zu vergeben. Diese Bedingung wird ausgenutzt, um die Punkte der Gastmannschaft automatisch aus der Punktezahl der Heimmannschaft zu berechnen. Dadurch wird der Aufbau der Grammatik etwas vereinfacht. So genügt es beispielsweise bei einer Eingabe Unentschieden das Tag hp für Heimpunkte auf den Wert 4 zu setzen.

Die Grammatiken mit den semantischen Tags sind durch die Dateiendung .slat markiert. Zur Erkennung werden daraus Netzwerke ohne Tags und ohne Teilnetze generiert und in Dateien mit dem gleichen Namen und der Endung .lat abgelegt. Diese Dateien werden dann in der Erkennung mit HVite verwendet. Aus den erkannten Wortfolgen werden dann in einem zweiten Schritt die semantischen Tags ermittelt.

Literaturverzeichnis

[DPP+96] Dutoit, T. ; Pagel, V. ; Pierret, N. ; Bataille, F. ; Van der Vrecken, O.: The MBROLA Project: Towards a Set of High-Quality Speech Synthesizers Free of Use for Non-Commercial Purposes. In: *ICSLP 96*. Philadelphia, 1996, S. 1393–1396

[Dra99] Draxler, C.: Specification of Database Interchange Format / SpeechDat-Car. 1999. – Forschungsbericht

[Eul05] Euler, S.: Java Tools for Teaching Speech Recognition. In: *ICASSP–2005*. Philadelphia, 2005

[MLD+00] Moreno, A. ; Lindberg, B. ; Draxler, C. ; Richard, G. ; Chouki, K. ; Euler, S. ; Allen, J.: Speechdat–Car. A large speech database for automotive environments. In: *LREC*. Athen, 2000

[WLK02] Walker, Willie ; Lamere, Paul ; Kwok, Philip: *FreeTTS - A Performance Case Study*. Sun Microsystems, 2002

Index