Erstellung eines webbasierten Sprachlehrsystems mit Spracherkennung

Projektarbeit von Stefan Westerfeld

29. Oktober 2009

Zusammenfassung

Diese Projektarbeit beschreibt die Erstellung einer im Webbrowser lauffähigen Anwendung, die das Erlernen der deutschen Sprache zum Ziel hat. Hierbei werden dem Nutzer Aufgaben gestellt, die er dann mündlich - durch Sprechen in ein Mikrophon - beantworten soll. Das Ergebnis wird dabei serverseitig durch Spracherkennung überprüft.
Der Fokus der Arbeit liegt hierbei nicht so sehr auf den eigentlichen Aufgaben, sondern mehr auf der Verwendung und Integration geeigneter Technologien, um Lernen im Browser mit Spracherkennung zu ermöglichen. Zum Einsatz kommen hierbei Java, ein Java Applet, PHP, Prolog und HTK.

Inhaltsverzeichnis

1. Einleitung

Der Prozess des Erwerbs einer Fremdsprache besteht aus mehreren Teilen. Einerseits muss der Lernende den Wortschatz der neuen Sprache kennen lernen, andererseits aber auch die grammatisch richtige Verwendung dieses Wortschatzes. Weiterhin ist die Aussprache der neuen Sprache mit eventuell anderen Phonemen zu erlernen. Schließlich findet der Spracherwerb auch nicht getrennt vom Kennenlernen der Kultur des entsprechenden Landes statt. Wie wichtig welche Teile und Ziele im Lernprozess sind, hängt nicht zuletzt vom Schüler und dessen Wünschen ab.

Die Idee, eine Sprache teilweise durch die Nutzung eines Computers mit entsprechender Software zu erlernen, ist nicht neu. Es gab beispielsweise Mitte der 90'er Jahre für den Commodore Amiga Programme zum Lernen von Vokabeln. Wenn man einen Teilaspekt – beispielsweise Vokabeltraining - durch Software abbilden kann, stellt sich die Frage, welche anderen Teile ebenfalls prinzipiell in Form eines Computerprogrammes realisiert werden können. Im Rahmen des MSS-Projektes können wir einen Teil dieser Frage beantworten, indem wir ein eigenes Sprachlehrsystem konstruieren.

Die grundlegende Aufgabenstellung des MSS-Projektes war die Entwicklung eines Systems, welches einen Nutzer, dessen Muttersprache nicht Deutsch ist, beim Erlernen der deutschen Sprache unterstützt. Die Abkürzung MSS steht hierbei für Multimodale Sprachlehrsysteme. Multimodal ist dabei so zu verstehen, dass das System mehrere Modalitäten in der Interaktion mit dem User verwendet, also in diesem Projekt sowohl als normale Webapplikation im Browser *optisch* dargestellt wird, als auch über die Erkennung gesprochener Sprache vom Nutzer *akustisch* mit dem Benutzer interagiert.

Im folgenden Kapitel wird zunächst umrissen, welche Anforderungen dabei gestellt waren, und das System wird von der Nutzerseite her vorgestellt. Es folgt ein Überblick über die Architektur, in dem zu sehen ist, aus welchen Teilen das System besteht, und wie diese interagieren.

Im Kapitel Implementation wird jede Komponente einzeln beschrieben. Dabei werden die theoretischen Grundlagen der Spracherkennung und einige Designentscheidungen besprochen.

2. Das System im Überblick

2.1 Anforderungen

Ein Ziel war, ein System zu entwickeln, welches Spracherkennung verwendet, um mit dem Nutzer zu interagieren. Die Anwendung sollte dabei als Webanwendung konzipiert werden, sodass der Endbenutzer lediglich einen Webbrowser benötigt, um die Anwendung zu nutzen. Dies ist deshalb vorteilhaft, da man im Allgemeinen davon ausgehen kann, dass ein Webbrowser bereits auf jedem System verfügbar ist. Daher ist keine weitere Installation eines Spracherkenners beim Endbenutzer notwendig.

2.2 Leistungsumfang

Im Folgenden wird der Ablauf einer Übung skizziert. Die Software präsentiert dem Nutzer zunächst ein Bild, in dem ein Objekt markiert ist. Dieses geschieht, indem der Nutzer eine bestimmte Webseite aufruft. Unter dem Bild befindet sich eine Leiste mit Buttons. Durch Betätigen eines Start-Knopfes wird die Aufnahme von Sprache gestartet. Nach einer festgelegten Zeit von 5 Sekunden endet die Aufnahme.

Die hierbei vom Nutzer zu erfüllende Aufgabe ist es, das Objekt, welches markiert ist, sprachlich zu

benennen. Eine mögliche Antwort könnte "die rote Tasse" lauten. Die aufgenommenen Sprachdaten werden vom System des Endusers von einem Java Applet auf den Server übertragen. Dort wird mittels Spracherkennung und einer einfachen Grammatik erkannt, ob die Antwort korrekt war.

Je nach dem, wie diese Überprüfung ausfällt, wird dem Benutzer eine Seite dargestellt, auf der erklärt wird, was er falsch gemacht hat. Benennt er beispielsweise „die rote Tasse“ als „der rote Tasse“, liefert das System eine Seite, die „Du hast den falschen Artikel verwendet, es sollte "die" heißen, nicht "der"!“ beinhaltet.

2.3 Architektur

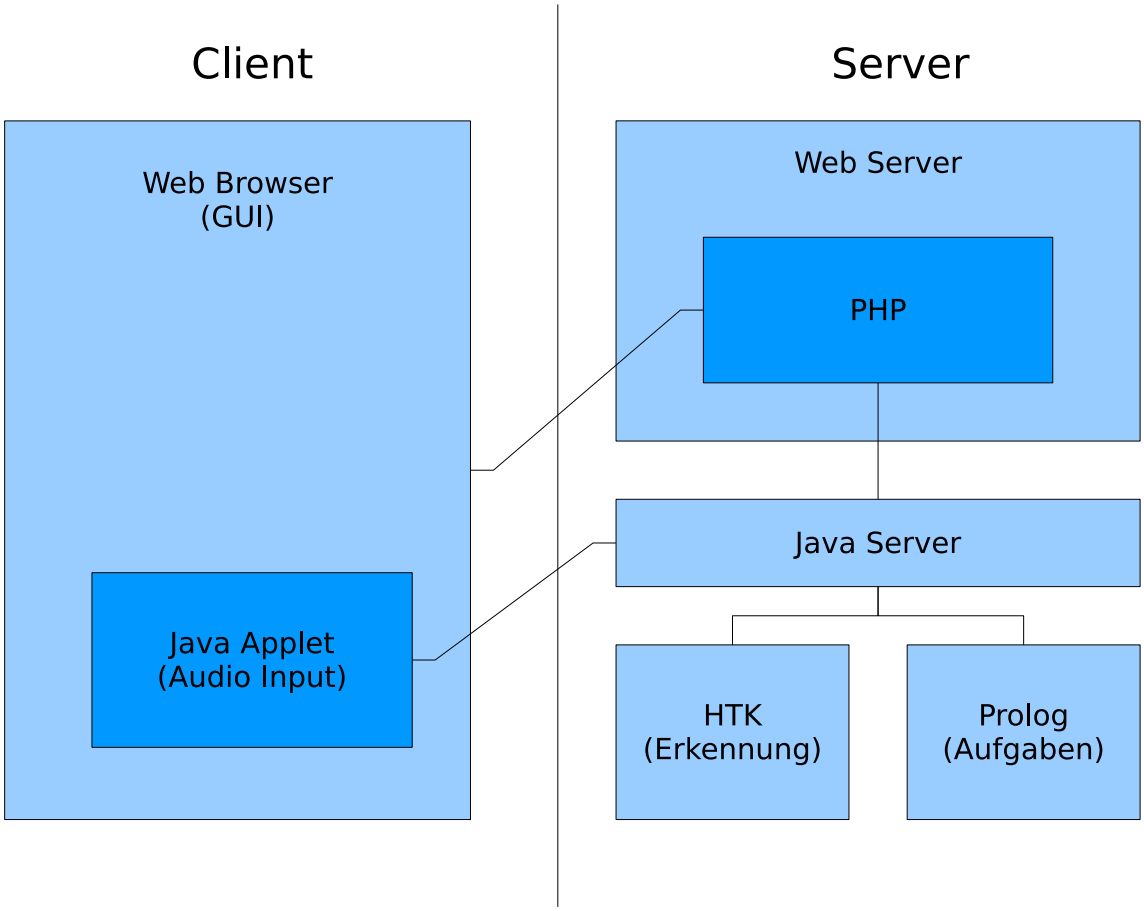

Abbildung 1: Architektur der Anwendung im Überblick

Die Anwendung ist so konzipiert, dass sie im Webbrowser lauffähig ist. Als Webbrowser kommt jeder Browser infrage, der neben der Darstellung von HTML auch noch Java Applets unterstützt. Von der Softwarearchitektur her gibt es eine klare Trennung zwischen dem Teil der Anwendung, der auf dem Client läuft, und dem Teil der Anwendung, der auf dem Server läuft. Die beiden Teile kommunizieren hierbei über TCP/IP, wodurch die Möglichkeit besteht, dass Client und Server räumlich getrennt und lediglich über Internet verbunden sind.

Es gibt hierbei, wie man in der Grafik sieht, unterschiedliche Kommunikationswege: einerseits gibt es die Kommunikation zwischen dem Webbrowser und dem Webserver; hierbei werden Webseiten und Grafiken über das http-Protokoll ausgetauscht.

Andererseits werden die vom Java Applet aufgezeichneten Sprachdaten über eine eigene TCP-Verbindung zu einem auf dem Server laufenden Java Server gesendet.

Auf dem Server werden vom Java Applet an den Java Server gelieferte PCM-Daten konvertiert und mit HTK wird der vom Schüler gesprochene Text in einen geschriebenen Text umgewandelt. Der Java Server übernimmt hierbei die Koordination zwischen dem im Webserver laufenden PHP einerseits und HTK und Prolog andererseits.

3. Die Implementation

3.1 PHP steuert die Oberfläche im Browser

PHP ist eine in Skriptsprache, die häufig auf Webservern eingesetzt wird, um dynamisch HTML-Seiten zu generieren. Der Client bekommt von PHP nur normale Webseiten geliefert, sodass jeder Webbrowser diese anzeigen kann, ohne etwas von der Existenz von PHP zu bemerken.

Das in dieser Anwendung zum Einsatz kommende PHP Skript „`sprache.php`“ („`src/Sprache/Gui/sprache.php`“) besitzt prinzipiell zwei Modi: einen, in dem Aufgaben gestellt werden, und einen, in dem eine Auswertung einer Antwort passiert.

Wenn der Benutzer mit seinem Webbrowser die Seite `sprache.php` aufruft, verbindet sich das Skript mit dem Javaserver, indem es eine TCP-Verbindung zum Server aufbaut, und erfragt von dort eine Aufgabenliste. Diese sieht beispielsweise so aus:

```
objekt#o1#Benenne das markierte Objekt!#bild-ananas.jpg#die-ananas.wav
objekt#o2#Benenne das markierte Objekt!#bild-apfel.jpg#der-apfel.wav
objekt#o3#Benenne das markierte Objekt!#bild-banane.jpg#die-banane.wav
objekt#o4#Benenne das markierte Objekt!#bild-birne.jpg#die-birne.wav
[…]
```

Jede Aufgabe hat einen Aufgabentyp (in dieser Version der Anwendung immer „`objekt`“), eine eindeutige „`id`“, einen Aufgabentext, ein Bild, und eine zugeordnete Wave-Datei, in der der Lösungstext von einem deutschen Sprecher vorgelesen wird (in dieser Version der Anwendung wird das Feld nicht verwendet).

Das PHP-Skript wählt zufällig eine Aufgabe aus und gibt eine Webseite aus, die die Aufgabenstellung, das Bild und das Javaapplet enthält. Im Browser sieht der User dadurch eine Seite, die in Abbildung 2 zu sehen ist.

Der zweite Modus, den das PHP-Skript hat, dient der Auswertung. Wenn vom Benutzer eine Spracheingabe aufgenommen wurde, veranlasst das Applet den Browser, eine neue Seite aufzurufen, die wiederum auf „`sprache.php`“ verweist, allerdings mit den zusätzlichen Parametern Aufgabennummer und einer von PHP generierten Nummer, unter der die vom Applet aufgenommen Sprachdaten gespeichert werden. Die vom Browser zur Auswertung geladene URL sieht also beispielsweise so aus:

`http://localhost/~stefan/mss/html/sprache.php?auswertung=1645506621&anr=o14`

Wie auch im ersten Modus baut das PHP-Skript eine TCP-Verbindung zum Java Server auf. Es sendet einen Befehl, um zu ermitteln, ob die Aufgabe richtig gelöst wurde, also die Spracheingabe vom Spracherkenner auf den richtigen Ergebnistext abgebildet wurde, oder nicht. Falls nicht der richtige Text erkannt wurde, sendet der Javaserver eine in deutsch abgefasste Fehlerdiagnose, zum Beispiel:

Erkannter Text: Ananas
Es fehlt der Artikel "die"!

Wie genau der Text erkannt wird, ist im Kapitel über HTK beschrieben, die Erstellung der Diagnose findet sich im Prolog-Kapitel.

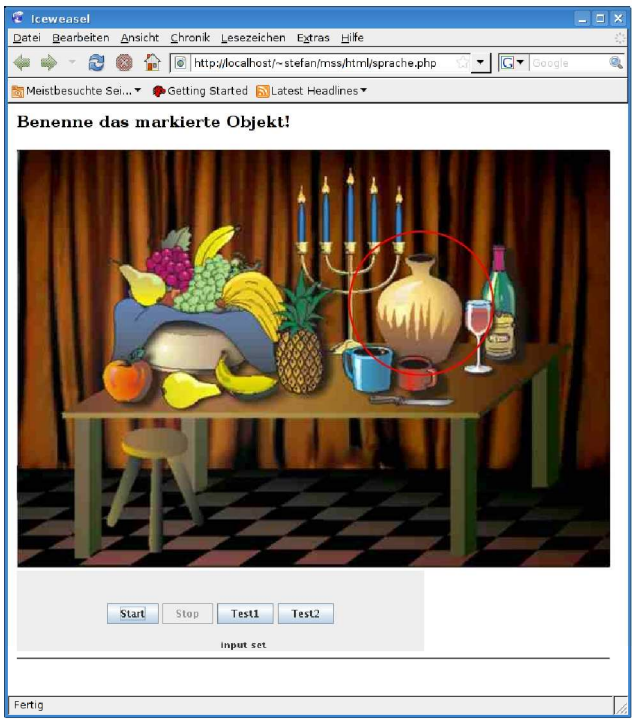

Abbildung 2: Webbrowser mit Javaapplet zur Spracheingabe

3.2 Aufnahmen mit dem Java Applet

Um die Spracheingabe im Webbrowser zu ermöglichen, wird ein Java Applet verwendet. Zu Beginn des Projektes war bereits der Sourcecode eines Applets vorhanden, welches mittels des Java Media Frameworks Sprache aufnehmen und diese über RTP an einen Server senden können sollte. Nach einigen vergeblichen Versuchen, dieses Applet – so wie es war - zum Laufen zu bringen, wurde das Applet im Laufe des Projektes angepasst und weiterentwickelt. Dabei wurde insbesondere auf die Verwendung des Java Media Frameworks verzichtet, sodass das veränderte Applet nur noch auf den normalen Java Sound Schnittstellen aufsetzt (`javax.sound.sampled.*`). Die Datenübertragung mittels RTP wurde durch eine reine TCP-Verbindung ersetzt.

Das Applet nimmt Daten vom Mikrophon auf und sendet die Ausgabe an den Java Server. Java Applets können, da sie in Java geschrieben sind, nahezu beliebige Komponenten für Webbrowser bereitstellen. Es handelt sich hierbei grundsätzlich um normale Java-Programme. Allerdings werden die Applets vom Browser in einer Sandbox ausgeführt, um den Nutzer vor schädlichen Javaapplets zu schützen. Beispielsweise dürfen Applets im Gegensatz zu Programmen keine Dateien lesen, die der Benutzer lokal installiert hat.

Eine solche Einschränkung ist auch, dass in der Standardeinstellung Java-Applets nichts vom Mikrophon aufzeichnen dürfen. Dies soll das heimliche Abhören von Gesprächen, die der Nutzer führt, während ein Javaapplet läuft, unterbinden. Da eine Aufnahme vom Mikrophon bei unserer Anwendung erwünscht ist, muss sie in der Datei `~/.java.policy` explizit erlaubt werden:

```
grant
{
  permission javax.sound.sampled.AudioPermission "record";
};
```

Die Aufnahme der Sprachdaten erfolgt mit einer Samplerate von 16kHz, in 16bit, mono. Dazu ein Zitat aus Tony Robinsons Speech Analysis Seiten [ROB98]:

> *„Signals must be filtered prior to sampling. Theortically the maximum frequency that can be represented is half the sampling frequency. In practice a higher sample rate is used to allow for non-ideal filters. [....] Telephone speech is sampled at 8 kHz. 16 kHz is generally regarded as sufficient for speech recognition and synthesis.“*

Wir verwenden also die allgemein als ausreichend angesehene Aufnahmequalität.

Die grafische Darstellung der Bedienelemente des Applets wurde mit Java Swing realisiert. Durch Verwendung dieses in Java standardmässig verfügbaren Toolkits kann die Oberfläche auf jedem Betriebssystem verwendet werden, auf dem Java läuft.

3.3 Ein Java Server koordiniert die Komponenten

Damit die Anwendung funktioniert, muss neben dem Webserver (mit PHP Modul) auf dem Server der Java Server gestartet werden. Dieser ist dann über einen TCP Port erreichbar. Dieser TCP Port ist üblicherweise 9000 und wird beim Starten des Java Servers festgelegt:

```
$ ./AudioReceiver 9000
[...]
AudioReceiver 0.1 listening on port 9000 ready.
[...]
```

Das auf dem Port 9000 angebotene Protokoll ist textbasiert, kann also auch von Hand über `telnet` erreicht werden:

```
$ telnet localhost 9000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 AudioReceiver 0.1 ready.
HELP
214 commands supported: DATA FAKE1 FAKE2 HELP PHP PHPLIST QUIT
```

Die Kommados `DATA`, `FAKE1` und `FAKE2` bilden hierbei die Schnittstelle zum Java Applet, die Kommandos `PHP` und `PHPLIST` bilden die Schnittstelle zur PHP Komponente.

Wenn das Java Applet durch einen Klick des Users in den Aufnahmemodus gebracht wird, baut es eine Verbindung zum Java Server auf. Es sendet dann „`DATA`“ und überträgt danach die aufgenommenen Daten, 16kHz, 16bit, mono, little endian raw. Der Java Server ergänzt einen Wave Header und schreibt eine gültige Wavedatei auf die Platte. Er konvertiert diese in eine MFC-Datei mit Merkmalen. Dies wird weiter unten im HTK Kapitel erklärt. Schließlich wird der Spracherkenner verwendet, um den gesprochenen Text (also etwa „die rote Tasse“) zu erkennen.

Damit der Spracherkenner nicht jedes Mal neu gestartet werden muss, wenn ein Satz erkannt werden soll, wurden in HVite ein Batch Modus eingebaut (der mit `HVite --batch` aktiviert wird). Dadurch ist es möglich, dem Spracherkenner immer nur den Namen einer neuen .mfc Datei zu übergeben, wenn eine Äußerung erkannt werden soll.

Die anderen beiden Kommandos, die das Java Applet verwendet (`FAKE1` und `FAKE2`), werden ähnlich verarbeitet. Allerdings tritt anstelle der übertragenen Audiodaten eine bereits auf dem Server verfügbare, fertig aufgenommene Datei. Diese Funktionalität ist lediglich für Tests sinnvoll.

Nach dem Versenden der aufgenommenen Daten (oder dem Verwenden der Testdaten) wird vom Java Applet die Seite mit der PHP Auswertung angefordert. Dieser schon im PHP-Kapitel beschriebene Modus der PHP Komponente erfragt vom Javaserver die Auswertung der Aufgabe. Das von PHP verwendete Kommando ist `PHP`, im Gegensatz zu `PHPLIST`, welches verwendet wird, um die Aufgabenliste zu erhalten.

Der Javaserver entscheidet nicht selbst über die Auswertung, sondern startet die Prologkomponente, um die Antwort auf die Aufgabe zu ermitteln. Diese übernimmt es, bei einem falschen Artikel beispielsweise darauf hinzuweisen, welcher Artikel richtig gewesen wäre. Hier ein Beispiel, nachdem ein Bild mit einer roten Tasse präsentiert wurde:

Antwort des Schülers: das rote Tasse

Diagnose (Prolog): Du hast den falschen Artikel verwendet, es sollte "**die**" heißen, nicht "das"!

3.4 Die Spracherkennung mit HTK

3.4.1 Merkmalsextraktion

Der erste Schritt, um aufgenommene Sprachsignale in Text umzuwandeln, ist die Merkmalsextraktion. Diese wird mit dem Programm `HCopy`, welches Teil von HTK ist, durchgeführt. Hierbei wird folgende Konfiguration verwendet:

```
# Wave -> MFCC config file
SOURCEFORMAT    = WAV
TARGETKIND      = MFCC_0_D_A
TARGETRATE      = 100000
WINDOWSIZE      = 200000
NUMCHANS        = 26
NUMCEPS         = 12
SAVECOMPRESSED  = T
SAVEWITHCRC     = T
USEHAMMING      = T
PREEMCOEF       = 0.97
CEPLIFTER       = 22
ENORMALIZE      = F
```

Das Spektrum von stimmhaften Sprachsignalanteilen weist in den tiefen Frequenzen mehr Energie auf als in hohen Frequenzen. Diese Eigenschaft hängt mit der Sprachproduktion beim Menschen zusammen. Um den Effekt zu kompensieren und damit die Erkennungsergebnisse zu verbessern, verwenden wir einen „pre-emphasis“-Filter (Vorverstärkungsfilter). Das Signal, gegeben als Folge der Abtastwerte s[n], wird hierbei mit folgender Formel gefiltert:

$$s'[n] = s[n] – k * s[n-1]$$

wobei als k die Einstellung `PREEMCOEF` verwendet wird, also k=0.97. Dieser Filter verstärkt im Signal die hohen Frequenzen und senkt tiefe Frequenzen ab. Über die Z-Transformationen kann man den Frequenzgang berechnen:

$$H(z) = 1 – k * z^{-1}$$

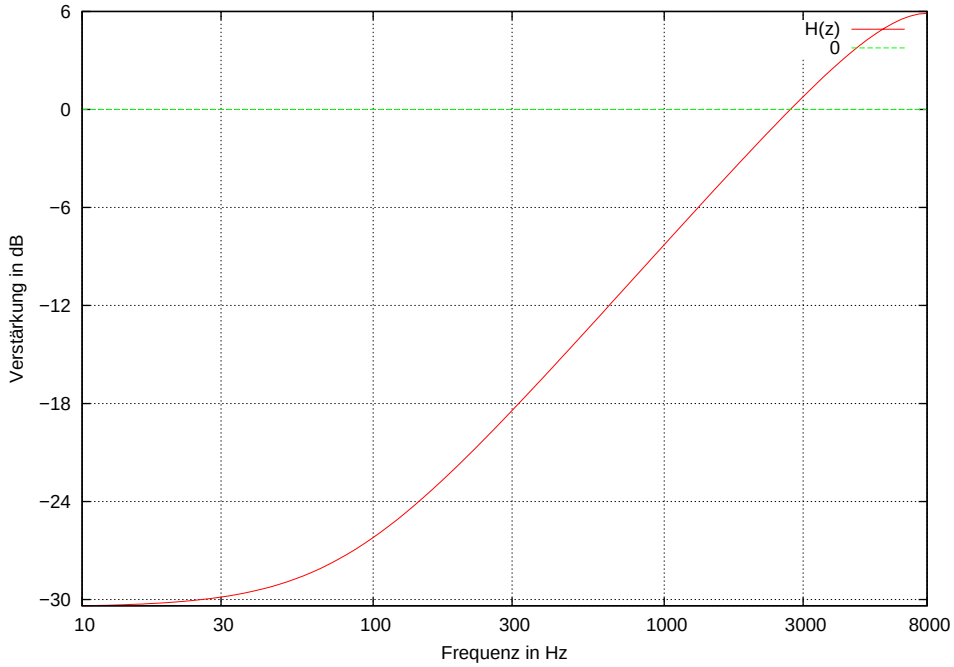

Abbildung 3: Frequenzgang des Vorverstärkungsfilters

Nach dem Anwenden des Vorverstärkungsfilters wird das gefilterte Sprachsignal in einzelne Blöcke mit der Länge von 20ms (`WINDOWSIZE`) unterteilt, wobei sich die Blöcke so überlappen, dass alle 10 ms (`TARGETRATE`) ein neuer Block anfängt.

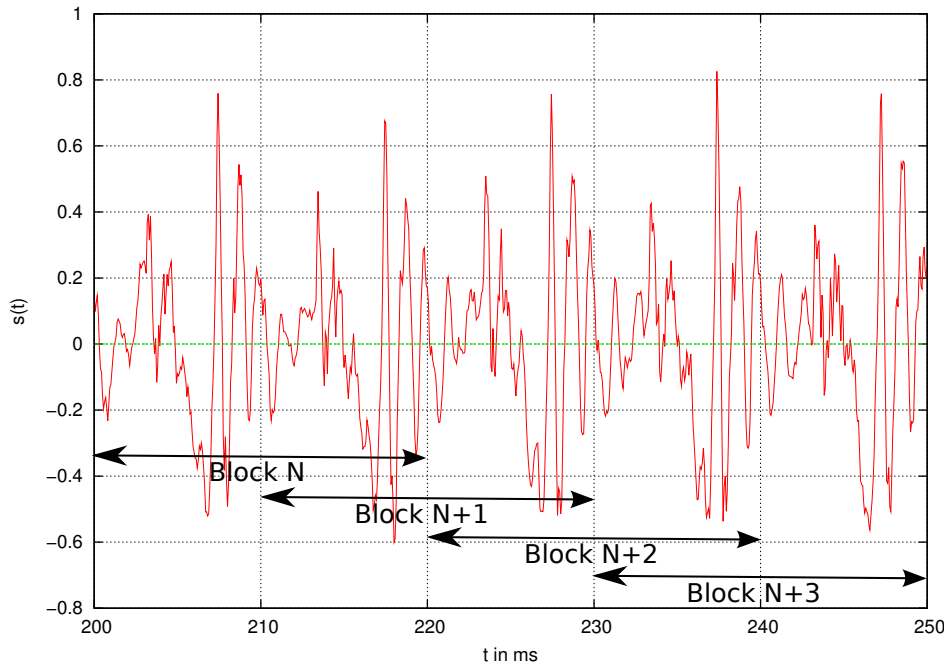

Abbildung 4: Einteilung des Sprachsignals in überlappende Blöcke

Für jeden dieser Blöcke wird ein Spektrum ermittelt, indem der Block erst mit einem Hamming

Fenster (`USEHAMMING=T`) multipliziert wird und das Spektrum dann mittels einer Fourier-Transformation, die durch eine FFT implementiert ist, berechnet wird. Durch das Hammingfenster werden die Auswirkungen des Ausschneidens eines Blocks aus dem Gesamtsignal, welche sich durch Sprünge am Anfang bzw. Ende des Blockes äußern, vermindert. Diese Art der Fourier-Transformation wird auch als Kurzzeitfouriertransformation (STFT = short time fourier transform) bezeichnet.

Um der Wahrnehmung des Menschen näher zu kommen, wird die Mel-Skala verwendet, um das Spektrum in Frequenzbereiche zu unterteilen. Die Einheit Mel, definiert durch die Formel

$$Mel(f) = 2595 \cdot \log_{10}(1 + \frac{f}{700})$$

ist so gestaltet, dass eine subjektiv als Verdoppelung der Tonhöhe wahrgenommener Frequenzunterschied einer Verdoppelung des Wertes in mel entspricht. Bei Tönen kleiner als 500 Hz entspricht eine Verdopplung der Frequenz einer Verdoppelung des Wertes in mel. Bei höheren Frequenzen ist der Zusammenhang nicht mehr linear. Dort werden deutlich größere Frequenzabstände als „doppelt so hoch“ empfunden.

In der Abbildung 5 sieht man, dass beim Zusammenfassen des FFT Spektrums in den tiefen Frequenzen schmalere Bänder zu einem Wert zusammengefasst werden, während es bei höheren Frequenzen deutlich breitere Bänder sind. Diese nach oben breiter werdenden Bänder sind in der Mel-Skala gleich breit. Man sieht auch, dass sich die Frequenzbänder überlappen.

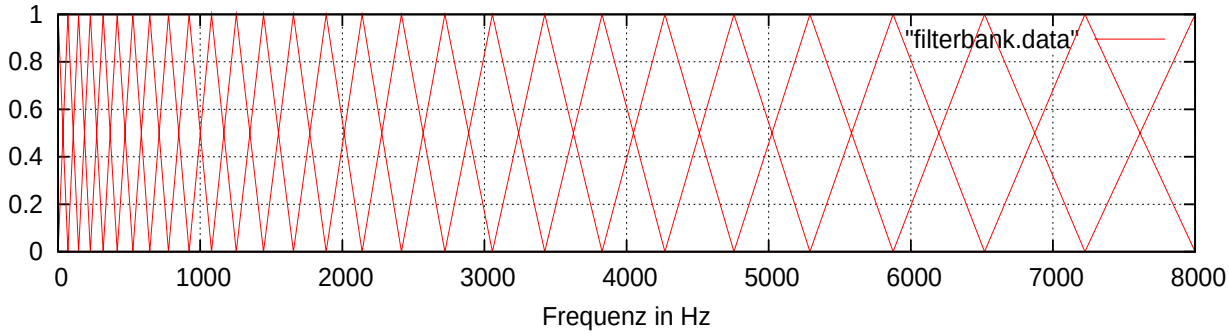

Abbildung 5: Zusammenfassen von Frequenzbereichen des Spektrums nach der Mel Skala

Nach dem Erstellen eines Vektors aus den Werten des Spektrums wird von diesem Vektor das Cepstrum berechnet. Dazu wird dieser logarithmiert, danach wird die Diskrete Cosinus Transformation (DCT) auf den Vektor angewandt. Nach der Berechnung des Cepstrum wird dieses noch durch Liftering (Filterung des Cepstrums) nachbearbeitet.

Die Erklärung, die im HTKBuch [HTK09] für den Liftering-Schritt gegeben wird, ist, dass es praktisch ist, wenn die Cepstral-Koeffizienten eine ähnliche Größe haben; dies wird durch die Skalierung

$$c_n' = (1 + \frac{L}{2}\sin(\frac{\pi n}{L}))c_n$$

erreicht, in unserem Fall mit L = 22 bei 12 Cepstral Koeffizienten.

Als letzter Schritt wird der Merkmalsvektor noch um die Delta- und DeltaDelta-Koeffizienten erweitert. Diese geben eine Art erste Ableitung (Steigung) des entsprechenden Merkmals und eine Art zweite Ableitung (Beschleunigung) des Merkmals an. Da es sich bei den Merkmalen um eine Folge von Messwerten handelt, ist die erste Ableitung (und die zweite Ableitung) nicht definiert, um diese zu berechnen müsste eine Funktion statt einer Folge vorliegen.

Die Berechnung der Delta-Koeffizienten an einer bestimmten Stelle verläuft deshalb wie folgt: zunächst werden insgesamt 5 Punkte aus der Messwertfolge extrahiert, also außer dem mittleren Wert, zu dem die Steigung gehört, noch zwei Werte davor, und zwei Werte danach. In diese wird mit linearer Regression eine Gerade durch die Messpunkte gelegt. Die Steigung dieser Gerade ist der gesuchte Delta-Koeffizient.

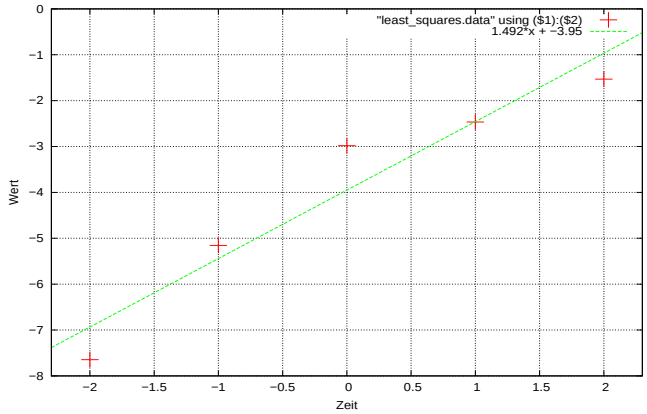

Abbildung 6: Berechnung eines Delta-Koeffizienten aus fünf Messwerten

Die DeltaDelta-Koeffizienten werden genauso berechnet, nur dass die zugrunde liegende Messwertfolge durch die zuvor berechneten Delta-Koeffizienten gegeben ist.

3.4.2 Phonmodellierung mit Hidden Markov Modellen

Im letzten Abschnitt wurde die Merkmalsextraktion beschrieben, die aus der vom Javaapplet aufgenommenen und an den Server übertragenen Audiodatei eine lange Liste von 39-dimensionalen Vektoren macht. Das Programm `HCopy` wird dazu mit den oben beschriebenen Parametern aufgerufen. In diesem Abschnitt soll nun beschrieben werden, wie aus den Merkmalen der gesprochene Inhalt als Text rekonstruiert werden kann.

Um diese Aufgabe zu lösen, wird zunächst eine Liste aller vorkommenden Wörter benötigt, in der die Aussprache jedes Wortes erfasst ist. So klingt beispielsweise das „e“ am Ende des Wortes „Tasse“ anders als das „e“ in dem Wort „der“. Um solche Unterschiede im Erkennungsprozess auseinanderhalten zu können, wird ein Aussprachewörterbuch verwendet. Jedes Wort wird auf eine Liste von Phonen abgebildet, die die Aussprache des Wortes darstellen. Hier ein Auszug aus dem Aussprachewörterbuch, in dem auch die beiden Beispiele vorkommen:

Wort	Aussprache
...	...
das	d a s sp
dem	d e: m sp
den	d e: n sp
der	d e:6 sp
die	d i: sp
...	...
gelbe	g E l b @ sp
gelben	g E l b @ n sp
gelber	g E l b P6: sp
gelbes	g E l b @ s sp
...	...
Tasse	t a s @ sp
...	...

Das hierbei zum Einsatz kommende phonetische Alphabet ist SAMPA [SAM96], wobei die Abbildung von Wörtern auf deren Aussprache eng verwandt mit der in Wörterbüchern für Fremdsprachen verwendeten Notation ist. Allerdings ist bei Verwendung von SAMPA nur der normalen ASC-II Zeichensatz für die Transkription notwendig, sodass beispielsweise das Wort Tasse „t a s ə“ als „t a s @“ repräsentiert wird.

Das am Ende jedes Wortes stehende „sp“ steht für „short pause“; dadurch ist modelliert, dass zwischen den einzelnen Worten eines Satzes eine kurze Pause stehen kann, aber nicht muss. Um zu erklären, wie aus einer Liste von Merkmalsvektoren erkannt werden kann, welche Worte gesprochen wurden, wird zunächst erklärt, wie der Klang eines einzigen Phones (etwa das „a“ in Tasse) in einem stochastischen Modell beschrieben wird.

Die stochastische Modellierung bietet sich deshalb an, weil jedes „a“-Phon, welches gesprochen wird, unterschiedlich ist. Nicht nur gibt es Unterschiede zwischen verschiedenen Sprechern, sowie beispielsweise zwischen Männern und Frauen. Zusätzlich wird der Klang des „a“-Phons wesentlich vom Kontext, also den davor und danach auftretenden Phonen geprägt. Aber selbst innerhalb des selben Wortes des selben Sprechers gibt es noch Unterschiede, jedes Wort wird jedes Mal etwas anders ausgesprochen.

Daher bietet sich die stochastische Modellierung jedes Phones mit Hidden Markov Modellen an. Ein Hidden Markov Modell besteht aus:

1. Einer Menge von Zuständen S = { s_1, s_2, s_3, … s_n }
2. Einer Übergangswahrscheinlichkeit $a_{ij} = p(q_t = s_j | q_{t-1} = s_i)$
3. Einer Anfangsverteilung, $\pi_i = p(q_1 = s_i)$
4. Eine Beobachtungswahrscheinlichkeit für jeden Zustand: $b_j(o) = \sum_{k=1}^{M} c_{jk} \cdot N(o, \mu_{jk}, U_{jk})$

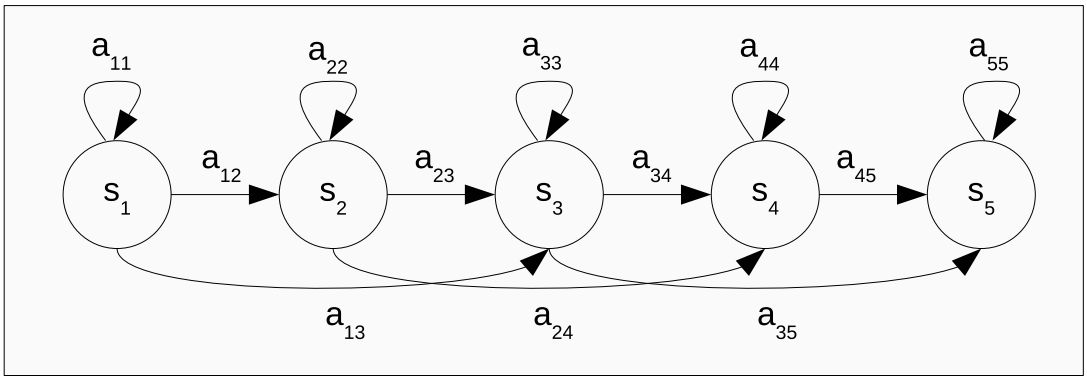

Abbildung 7: Ein Hidden Markov Modell mit fünf Zuständen

Die oben stehende Grafik illustriert ein Hidden Markov Modell mit fünf Zuständen. Die Pfeile bedeuten, dass man von einem Zustand mit einer durch a_{ij} gegebenen Wahrscheinlichkeit in den Folgezustand gelangen kann. Einige Zustandsübergänge (zum Beispiel ein Sprung zurück von Zustand s_4 in s_2) sind nicht möglich, haben also eine Wahrscheinlichkeit von $a_{ij} = 0$.

Bei der Modellierung von Sprache könnte das Hidden Markov Modell beispielsweise ein „a“ Phon modellieren. Zu jedem der fünf Zustände gehört dann eine Funktion $b_j(o)$, die über eine Mischverteilung von mehrdimensionalen Normalverteilungen beschreibt, wie die für diesen Zustand typischerweise beobachteten Merkmalsvektoren aussehen. Diese Verteilungen sind für jeden Zustand eines Modelles unterschiedlich; damit wird modelliert, dass ein „a“ am Anfang, in der Mitte und am Ende anders klingt (bzw. die erwarteten Merkmalsvektoren sich für diese Zustände unterscheiden).

Für den verwendeten Spracherkenner sind die Dinge etwas komplizierter. Der Klang eines „a“-Phones wird durch die davor und danach auftretenden Phone beeinflusst. Ein „a“ klingt also anders, wenn es zwischen einem „t“ und einem „s“ steht, wie in „Tasse“, als wenn es zwischen einem „n“ und einem „n“ steht, wie in „Banane“. Eine Technik, um diesen Kontext mitzumodellieren, ist es, einfach unterschiedliche Modelle zu verwenden, also ein Modell für "„a“ zwischen „t“ und „s“" („Tasse“) und ein anderes Modell für "„a“ zwischen „n“ und „n“" („Banane“). Diese Modelle, die den Kontext jedes Phones mitmodellieren, nennt man Triphone. Bei unserem Spracherkenner gibt es also ein eigenes Hidden Markov Modell für „t-a+s“ („a“ in „**Tas**se“) und ein eigenes Modell für „n-a+n“ („a“ in Ba**nan**e). Die Notation „t-a+s“ bezeichnet ein „a“, vor dem sich ein „t“ befindet, und nach dem ein „s“ folgt.

Wie kann man nun die Äußerung „die Tasse“ vor der Äußerung „die Banane“ unterscheiden? Gegeben sei zunächst die Merkmalsvektorfolge, die aus der Aufnahme der Sprachdaten eines Sprechers mit einem Mikrophon errechnet wurde. Wir nehmen zunächst an, dass nur die beiden Äußerungen möglich sind. Nun kann man für jedes Wort dieser Äußerungen eine phonetische Repräsentation im Aussprachewörterbuch nachschlagen. Eine Wortsequenz wie „die Tasse“ kann durch aneinanderfügen der Modelle der einzelnen Wörter produziert werden. Man findet:

- die → d i: sp
- Banane → b a n a n @ sp
- Tasse → t a s @ sp

Um den Kontext mitzumodellieren, werden wie oben beschrieben Triphone verwendet, wobei an den Wortgrenzen („sp“ = short pause) kein linker bzw. rechter Kontext für ein Triphon vorhanden ist, insofern werden dort Biphone verwendet. Also:

- die → **d**+i: d-**i:** sp
- Banane → **b**+a b-**a**+n a-**n**+a n-**a**+n a-**n**+@ n-**@** sp
- Tasse → **t**+a t-**a**+s a-**s**+@ s-**@** sp

Am Beispiel des Wortes „Tasse“ (die anderen Wörter werden analog behandelt) ist hier eine graphische Darstellung der Konstruktion eines Gesamtmodells für das Wort angegeben:

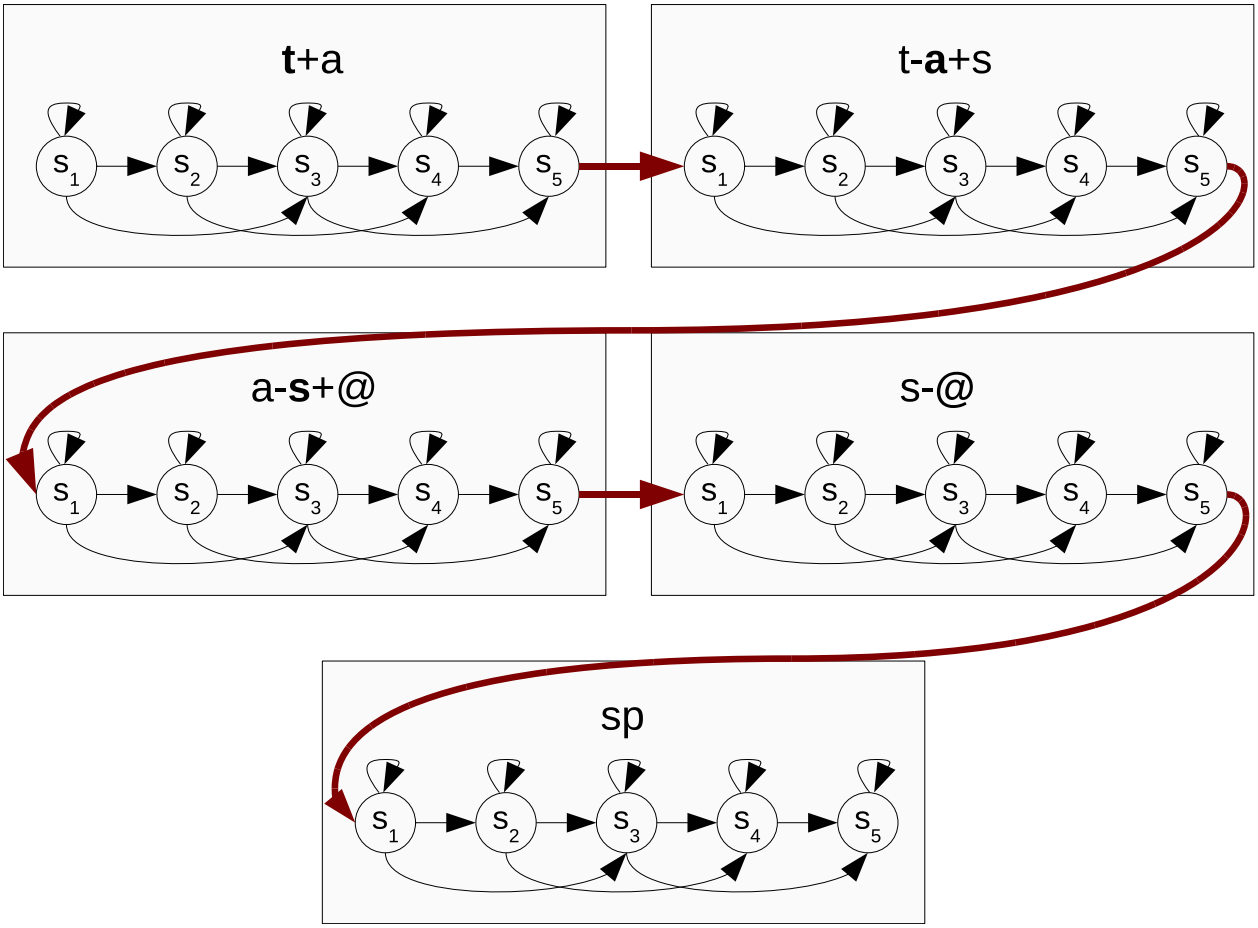

Abbildung 8: Konstruktion eines Hidden Markov Modells für Tasse

Durch diesen Verkettungsschritt ensteht ein Hidden Markov Modell für „Tasse“ aus den Einzeltriphonen. Aus den drei Wortmodellen lässt sich wiederum ein Gesamtmodell für die gewünschten Wortfolgen erzeugen:

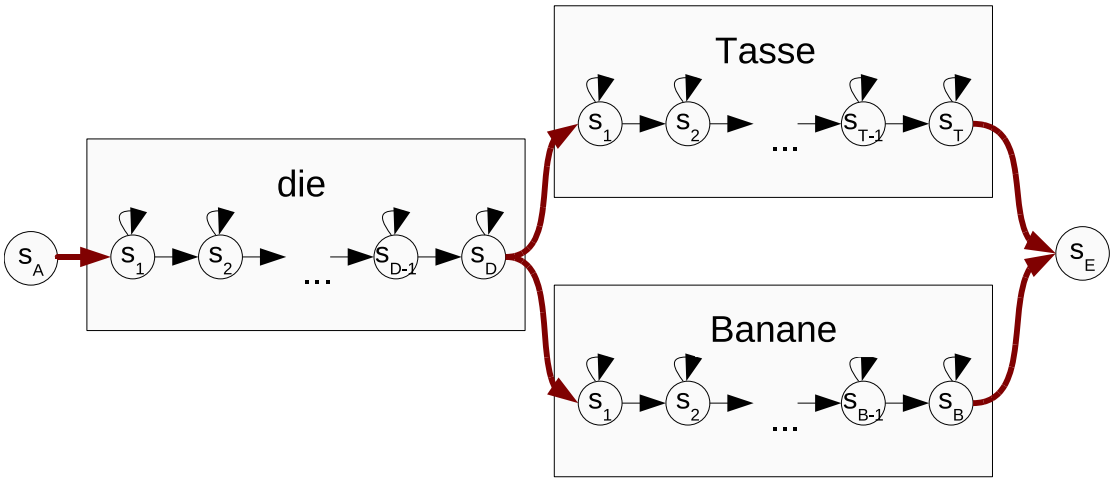

Abbildung 9: Ein Hidden Markov Modell für zwei Wortfolgen

Wie man in Abbildung 9 sieht, erhält man durch Zusammensetzen der Einzelmodelle ein kombiniertes Hidden Markov Modell, welches die Äußerungen „die Tasse“ und „die Banane“ beschreibt. Bei einer gegebenen Folge von Merkmalsvektoren kann dieses Modell verwendet werden, um zu entscheiden, welche der möglichen Äußerungen am wahrscheinlichsten ist. Dazu suchen wir den wahrscheinlichsten Pfad, der vom Startzustand (S_A) in den Endzustand (S_E) führt.

Geht dieser wahrscheinlichste Pfad durch den oberen Zweig des Modells, entscheiden wir, dass die Merkmalsfolge dem Text „die Tasse“ entspricht, geht der wahrscheinlichste Pfad durch den unteren Zweig des Modells, entscheiden wir, dass die Merkmalsfolge dem Text „die Banane“ entspricht.

Gegeben seien die Beobachtungsdaten: $O = (o_1, o_2, o_3, \ldots, o_T)$
und ein Hidden Markov Modell: HMM

Gesucht ist die beste Zustandsfolge: $q = (q_1, q_2, q_3, \ldots, q_T)$

Zu berechnen ist also q als:

$$\underset{q}{argmax}\, P_q(O|HMM) = \pi_{q_1} \cdot b_{q_1}(o_1)\, a_{q_1 q_2} b_{q_2}(o_2), \ldots, a_{q_T - 1 q_t} b_{q_T}(o_T)$$

Dies lässt sich naiv so implementieren, dass man jede mögliche Zustandsfolge getrennt überprüft, also für jedes mögliche q die Wahrscheinlichkeit P_q berechnet, die beschreibt, wie gut die Beobachtungsdaten zu dieser Zustandsfolge in diesem HMM passen. Der naive Algorithmus ist allerdings nicht effizient genug, da es sehr viele mögliche Zustandsfolgen gibt. Bei N Zuständen und einer Beobachtungslänge von T gibt es N^T Möglichkeiten für q.

Glücklicherweise lässt sich die gesuchte Zustandsfolge mit dem Viterbi Algorithmus wesentlich effizienter finden. Die Darstellung dieses Algorithmus findet sich zum Beispiel in [RAJ93]. Für unsere Zwecke war es ausreichend, daß HTK mit `HVite` eine effiziente Implementation des Viterbi-Algorithmus (also der Suche nach dem besten Pfad wie oben beschrieben) enthält, sodass wir diesen nur durch ein Aussprachewörterbuch, Triphone und die Grammatik, die im nächsten Abschnitt beschrieben wird, parametrisieren mussten.

Um Triphon-Modelle für eine Sprache (wie Deutsch) zu erhalten, gibt es Trainingsverfahren. Man braucht Aufnahmen vieler Äußerungen vieler Sprecher, zusammen mit den Wörtern, die gesprochen wurden, und deren Aussprachen. Für das MSS-Projekt lagen fertige Triphon - Hidden Markov Modelle vor, die bereits auf einer ausreichend großen Menge von Trainingsdaten trainiert worden waren.

3.4.3 Das Grammatikmodell

In 3.4.2 wurde beschrieben, was notwendig ist, um mit HTK zwischen verschiedenen gesprochenen Äußerungen zu unterscheiden. Die einzige Komponente, die für den Spracherkenner noch zu spezifizieren ist, ist, welche Wortfolgen als Eingabe zugelassen sind. Im Kontext dieser Übung wird immer nach einem Objekt, welches auf einem Bild zu sehen ist, gefragt. Dabei gehen wir davon aus, dass der Schüler nicht mit einem ganzen Satz auf eine Aufgabe antwortet, etwa „ich sehe eine rote Tasse“, sondern nur das gefragte Objekt benennt, also mit „eine rote Tasse“ antwortet. Um eine Antwort grammatikalisch zu modellieren, verwenden wir folgende Regel:

NP → [Artikel] [Adjektiv] Objekt

Artikel ist hierbei ein bestimmter oder unbestimmter Artikel, also aus { der, die, das, ein, eine }. Der Artikel wird von der Grammatik als optional behandelt.

Adjektiv ist ein Farbadjektiv, aus { blaue, blauen, blauer, blaues, braune, …, rotes }. Die Grammatik behandelt auch das Adjektiv als optionalen Bestandteil.

Objekt ist der Name des gefragten Objektes, also aus { Ananas, Apfel, Banane, … Weintrauben }. Dieses ist *nicht* optional.

Hier einige Beispiele für Antworten, die der Schüler geben könnte:

(1) die rote Tasse

(2) die blaue Tasse

(3) die Tasse

(4) eine rote Tasse

(5) *das blaues Tasse

(6) *das Ananas

(7) *das braunes Ananas

(8) ?die blaue Ananas

(9) *Ananas

Die Grammatik lässt absichtlich Äußerungen zu, die im Deutschen sprachlich falsch (markiert mit *) oder inhaltlich falsch (markiert mit ?) sind. Da wir davon ausgehen, dass der Sprecher gerade deutsch lernt, versuchen wir mögliche Fehler in der Grammatik zu antizipieren, um bei sprachlich oder inhaltlich falschen Äußerungen diese vom Spracherkenner unkorrigiert verschriftlichen zu lassen.

Im nächsten Kapitel wird beschrieben, wie diese Fehler vom System behandelt werden.

3.5 Prolog für die eigentlichen Aufgaben

Die Prologkomponente hat zwei Funktionen. Einerseits speichert sie die möglichen Aufgaben, die

das System dem Schüler stellen kann. Dabei handelt es sich im wesentlichen um die folgende Liste:

```
aufgabe(objekt,o1,ananas,'Benenne das markierte Objekt!','bild-ananas.jpg','die-ananas.wav').
…
aufgabe(objekt,o14,vase,'Benenne das markierte Objekt!','bild-vase.jpg','die-vase.wav').
```

Auf der anderen Seite wird ein Modell der Sprache verwendet, um die Antwort auf gestellte Aufgaben (beispielsweise „o14“) prüfen, und, wenn nötig, verbessern zu können. Bleiben wir beim sprachlichen Wissen für Aufgabe „o14“. Um die Antwort prüfen zu können, wird mit einem Prädikat gespeichert, dass der Genus von Vase im deutschen weiblich ist (möglich ist 'm' = *Maskulinum* - 'männlich', 'n' = *Femininum* – 'weiblich' oder 'n' = *Neutrum* - 'keines von beiden').

```
n(vase,f).
% … Liste des Genus der anderen Objekte
```

Die deutschen bestimmten und unbestimmten Artikel liegen auch nach Genus markiert vor:

```
% artikel: bestimmte:
art(der,b,m).
art(die,b,f).
…
% ... unbestimmte:
art(ein,u,m).
art(eine,u,f).

…
```

Schließlich ist auch vermerkt, welche verschiedenen flektierten Formen das Adjektiv (zum Beispiel „braun“) annehmen kann, und dass die Vase braun ist, aber der Schüler diese Angabe weglassen darf:

```
adj(braun,brauner,braune,braunes,braunen).
eigenschaft(vase,braun,optional).
```

Mit diesen Informationen kann das in Prolog implementierte Prädikat `np` die vier korrekten Nominalphrasen, die zu der Frage „Bennenne das markierte Objekt!“ mit markierter Vase passen, berechnen, und zwar:

```
?- np(vase,X,[]).
X = [die, vase] ;                       „Die Vase.“
X = [die, braune, vase] ;               „Die braune Vase.“
X = [eine, vase] ;                      „Eine Vase.“
X = [eine, braune, vase] ;              „Eine braune Vase.“
false.
```

Für den Fall, dass ein Fehler vorliegt, wird auf der dritten Argumentstelle eine Fehlerliste generiert, die sich mit einem weiteren Prädikat in einen Fehlertext umsetzen lässt. In der folgenden Tabelle werden einige Möglichkeiten gezeigt.

Antwort des Schülers	Diagnoseliste	Fehlermeldung für den Schüler
„Ein Vase.“	[artikel(ein, eine)]	Du hast den falschen Artikel verwendet, es sollte "**eine**" heissen, nicht "ein"!
„Die grüne Vase.“	[farbe(gruen, braun)]	Du hast die falsche Farbe verwendet, das Objekt ist **braun**, nicht gruen!
„Das rote Vase.“	[artikel(das, die), farbe(rot, braun),	Du hast den falschen Artikel

	farbe_flektion(roten, rote)]	verwendet, es sollte "**die**" heissen, nicht "das"! Du hast die falsche Farbe verwendet, das Objekt ist **braun**, nicht rot! Du hast die falsche Form fuer die Farbe verwendet, es sollte "**rote**" heissen, nicht "roten"!
„Vase.“	[artikel_notwendig(die)]	Es fehlt der Artikel "**die**"!

4 Schlussfolgerungen

Die im MSS-Projekt entwickelte Architektur stellt eine Möglichkeit dar, wie das Erlernen einer Sprache durch Software unterstützt werden kann. Dabei ist es für den Schüler ausreichend, einen Java-fähigen Webbrowser zu benutzen, eine Installation zusätzlicher Software ist nicht notwendig. Um auf Fehler, die der Schüler beim Lösen der Aufgaben macht, einzugehen, haben wir in der Grammatik falsche Äußerungen zugelassen (etwa „der rote Tasse“) und eine Prolog-basierte Diagnosekomponente entwickelt, die bei falschen Äußerungen Verbesserungsvorschläge macht. In gewisser Weise ist die Spracherkennung also robust gegenüber Fehlern.

Allerdings kann man sich Fehler vorstellen, die nicht auf der inhaltlichen oder grammatischen Ebene liegen. Ein Beispiel wäre ein französischer Schüler, der statt „das rote Haus“ nur „das rote ..aus“ sagt, weil er das „H“ nicht so wie ein deutscher Sprecher aussprechen kann. In weniger extremen Fällen wird ggf. ein Vokal etwas anders ausgesprochen als von einem deutschen Sprecher.

Die von uns gewählte Lösung bietet keine systematische Behandlung solcher Fehler, die man vielleicht unter dem Stichwort „Aussprachevarianten“ oder „Aussprachefehler“ führen könnte. Sowohl Experimente mit nicht nativen Sprechern als auch weitere Entwicklungen – etwa die Erweiterung des vom Spracherkenner verwendeten Aussprachewörterbuches – könnten das System weiter verbessern.

Zum Verbesserungspotential gehört auch die Sammlung der Aufgaben. Um einen relevanten Lernerfolg zu erzielen, sind sicher mehr als die von uns bereitgestellten 14 Aufgaben nötig. Einerseits ist die Anzahl der Aufgaben schlicht zu klein. Sicherlich wäre es auch wünschenswert, wenn die Aufgaben einem didaktischen Gesamtkonzept folgen würden. Die Prologkomponente könnte beispielsweise auf Sätze wie „die rote Tasse steht neben der Ananas“ erweitert werden, aber auch Sätze wie „der Mann trinkt Milch“ wären denkbar. Die grammatischen Strukturen müssten in die Grammatik des Spracherkenners eingebaut werden, und die neuen Wörter im Aussprachewörterbuch.

Andererseits wäre es sehr sinnvoll, wenn bei jeder Aufgabe die korrekte, von einem deutschen Muttersprachler aufgenommene Lösung, vom Computer vorgesprochen werden könnte. Durch diese Beispiele könnte der Schüler lernen, indem er den Muttersprachler nachahmt.

Zusammenfassend lässt sich sagen, dass mit dem MSS-Projekt eine funktionsfähige Lösung für die Integration von Spracherkennung, Web und Aufgabenrepräsentation entwickelt wurde, wobei Raum für weitere Verbesserungen vorhanden ist.

Literaturverzeichnis

ROB98: Robinson, Tony, Speech Analysis, 1998,
http://mi.eng.cam.ac.uk/~ajr/SpeechAnalysis/SpeechAnalysis.html
HTK09: Steve Young, Gunnar Evermann, Mark Gales et al, The HTK Book, 2009
SAM96: , Sampa for German, , http://www.phon.ucl.ac.uk/home/sampa/german.htm
RAJ93: Rabiner, Juang, Fundamentals of Speech Recognition, 1993