Studienarbeit

optimierte Wörterbucherstellung, Spracherkennung in Automobilen

Daniel Kiecza

23. Juni 1997

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit besteht aus zwei Teilen:

Im ersten Teil soll ein Programm vorgestellt werden, das es gestattet, Wörterbücher[1], wie sie in heutigen Spracherkennern vorkommen, komfortabel über eine graphische Benutzerschnittstelle zu erstellen und zu bearbeiten. Das Programm soll neben der direkten Eingabe der Phonemfolgen weitere Möglichkeiten zur Findung von Aussprachen bieten: z.B. Suche in bestehenden Wörterbüchern oder Vorschlag einer Phonemfolge aufgrund einer einfachen Graphem-zu-Phonem-Umsetzung.
Natürlich sollen die Vorgänge, soweit möglich, automatisiert werden.

Der zweite Teil beschreibt die Grundlagen für den Aufbau eines Spracherkenners, der im Auto zum Einsatz kommt, um Anweisungen des Fahrers entgegenzunehmen. Neben der Darlegung des zugrundeliegenden Erkenners und der verwendeten Datenbasis werden erste Ergebnisse präsentiert.

Als Verbindungsglied zwischen den beiden Teilthemen dieser Arbeit soll der praktische Einsatz des Programms *DictCreator* für den Aufbau des Erkenners aufgezeigt werden.

[1] Diese Dateien enthalten die Zuordnungen der dem Erkenner bekannten Wörter zu einer Folge von Phonemen, z.B.: **Abteilung** ↦ **A P T A I L U NG**

2 Wörterbucherstellung mit *DictCreator*

2.1 Die Idee

Für viele Entwickler von Spracherkennungssystemen ist die Erstellung des Wörterbuchs, also der Datei, die die Abbildung von Wörtern auf entsprechende Phonemfolgen enthalt, ein Greuel.
Die Idee des Programms *DictCreator* besteht nun gerade darin, für diesen Zweck einen komfortablen Editor bereitzustellen. Da Spracherkennungssysteme auf vielen verschiedenen Plattformen entwickelt werden, ist es nur zweckmäßig, diesen Wörterbucheditor in einer plattformunabhängigen Sprache zu programmieren. Aus diesem Grund ist *DictCreator* in Java geschrieben.

DictCreator soll es erlauben, ein unbekanntes Wort in Teilwörter zerlegen zu können, die Aussprachen für die einzelnen Teilwörter separat zu bearbeiten und das Ergebnis dann zusammenzusetzen. Aber neben der reinen Editierfunktion soll das Programm auch Möglichkeiten bereitstellen, Aussprachen selbständig zu finden. Dazu gehört beispielsweise die Suche in bestehenden Wörterbüchern oder das Vorschlagen von Aussprachen über Graphem-zu-Phonem-Abbildungen.
Im Zeichen der effizienten Erstellung eines Wörterbuchs muß das Programm natürlich seine eigenständige Aussprache-Findung weitestgehend automatisiert durchführen können.

Da die Entwicklung von Spracherkennern in anderen Sprachen sicherlich neue Dateiformate mit sich bringen wird, ist ein vorrangiges Entwurfsziel des Programms *DictCreator* die Festlegung einfacher Schnittstellen, die es erlauben, Dateifilter für neue Dateiformate einfach in das System zu integrieren. Das betrifft sowohl die Filter für Ein-/Ausgabedateien als auch die Filter für bestehende Wörterbücher.

2.2 Methoden zur Aussprache-Findung

In *DictCreator* werden drei Verfahren benutzt, um selbständig Aussprachen zu finden: Eine einfache *Suche*, ein *rekursives Auftrennverfahren* sowie eine *Graphem-zu-Phonem-Umsetzung*. Diese Verfahren sollen im folgenden näher erläutert werden.

1. **Suche:**
   Die Suche ist das einfachste der drei Verfahren. Hierzu werden lediglich sämtliche der vom Anwender ausgewählten Wörterbücher nach dem Wort durchsucht; und zwar in der Reihenfolge ihrer Plazierung in der Auswahlliste. Sobald das gesuchte Wort in einem der Wörterbücher gefunden wird, wird die Suche beendet und die gefundene Aussprache zurückgegeben.

2. **rekursives Auftrennen:**
   Diese Methode basiert auf der oben beschriebenen Suche. Bei ihr wird versucht, das Wort systematisch aufzutrennen, um die so gewonnenen Teilwörter in den Wörterbüchern zu suchen.
   Betrachten wir die Vorgehensweise an einigen Beispielen:

   (a) **zusammengesetzte Wörter:**
   Gesucht werde das Wort *Autobahnraststätte*. Es sei angenommen, daß die gewählten Wörterbücher die Aussprachen zu *Autobahn*, *Rast*, und *Stätte* kennen.
   Der Suchalgorithmus sucht jetzt in den bekannten Wörterbüchern der Reihe nach die Wörter:
   Autobahnraststätte,
   Autobahnraststätt,
   Autobahnraststät,
   Autobahnraststä,
   Autobahnrastst,
   Autobahnrasts,
   ⋮
   Wird der Algorithmus fündig, was hier bei *Autobahn* das erste Mal der Fall ist, merkt er sich die Aussprache zu diesem Teilwort und startet das Verfahren (rekursiv) erneut für den verbleibenden Rest

des Wortes (hier also *raststätte*)[2]. Ist dieser Rest das leere Wort, so ist das Verfahren beendet.
Wird für den verbleibenden Rest eine Aussprache gefunden, so ist das Verfahren ebenfalls beendet: Die gefundenen (Teil-)Aussprachen können zusammengesetzt und zurückgegeben werden. Im vorliegenden Fall bringt die erneute Anwendung des Verfahrens auf den Rest folgendes Fortschreiten:
raststätte,
raststätt,
raststät,
raststä,
rastst,
rasts,
rast, → Der Algorithmus wird fündig! Wir merken uns die Aussprache zu *rast* und starten das Verfahren wieder (rekursiv) mit *stätte*. Dieses Wort findet der Algorithmus sofort; wir sind hiermit fertig, und können die aus den „aufgesammelten" Aussprachen zusammengesetzte Gesamt-Aussprache zurückgeben.

(b) **zusammengesetzte Wörter mit Verbindungs-S:**
Betrachten wir den folgenden Fall: gesucht werde das Wort *Erdungskabel*. Bekannt seien die Aussprachen zu *Erdung* und *Kabel*. Nachdem das Wort *Erdung* gefunden und abgetrennt worden ist, werden wir für den Rest des Wortes *skabel* nicht mehr fündig: das Verbindungs-S **stört**! Aus diesem Grund prüft der Algorithmus das Restwort – nach vergeblicher Suche – auf ein führendes S. Ist dieses vorhanden, wird es abgetrennt und auf den Rest (hier also *kabel*) wieder das Verfahren angewandt. Auch hier werden wir jetzt fündig und können, die Aussprache des Verbindungs-S als einfaches S-Phonem annehmend, die Aussprachen zusammensetzen und zurückliefern.

(c) **Pre-/Postfixe:**
Nehmen wir an, wir suchen die Aussprache zu dem Wort *Herrlichkeit*. Unsere ausgewählten Wörterbücher sollen die Aussprache zu dem Wort *herrlich* kennen. Im vorliegenden Fall kommt der letzte Trumpf von *DictCreator* zur Anwendung. Das Programm

[2]Der Algorithmus sucht ohne Beachtung der Groß-/Kleinschreibung

kennt eine Liste von, in der jeweiligen Sprache üblichen Pre- und Postfixen[3] mit ihren zugehörigen Aussprachen. Unter anderem enthält die Postfixliste das Postfix *keit* $\mapsto$ *K AI T*. Nach dem Abtrennen des Wortes *herrlich* prüft das Programm den Anfang und das Ende des übrigen Wortes *keit* – nach erfolgloser Suche in den Wörterbüchern – auf Übereinstimmung mit einem der bekannten Prefixe. Im Erfolgsfall wird das Prefix abgetrennt, und die Suche mit dem Rest gestartet.
Sonst wird analog mit den Postfixen verfahren. Im konkreten Fall finden wir also letztlich das Postfix *keit* in unserer Postfixdatei, und können die zusammengesetzte Aussprache zurückgeben.

Wenn wir bis hierher nicht fündig geworden sind, so steigen wir in der Rekursion wieder eine Stufe nach oben und setzen dort das Verfahren fort (Verkürzung des Wortes und erneute Suche)[4].

(d) **Problemfall 1:**
Natürlich liefert dieses Verfahren nicht garantiert die gewünschte Lösung. Wir nehmen an, die bekannten Wörterbücher enthalten Aussprachen zu *Gefahr* $\mapsto$ *G E F AH R* und zu dem französischen *en* $\mapsto$ *O NG*. Wir suchen mit dem Auftrenn-Verfahren nach dem Wort *Gefahren*. Der Algorithmus wird bei der Verkürzung fündig bei dem Wort *Gefahr*. Dieses wird folglich abgetrennt. Für den Rest *en* finden wir ebenfalls (sofort) eine Aussprache, nämlich die des französischen Wortes. Die Aussprachen werden zusammengesetzt und, fälschlicherweise, als Lösung zurückgegeben.

(e) **Problemfall 2:**
Eine weitere Problemklasse bilden die zusammengesetzten Wörter, die mit gleichen Konsonanten aufeinanderstoßen (z.B. Hakennase). Dieser Doppelkonsonant wird beim Sprechen im Normalfall zu einem Phonem zusammengezogen, das Verfahren erkennt jedoch zwei Phoneme. Aus diesen Gründen werden die mit diesem Verfahren gefundenen Aussprachen als *unsicher* klassifiziert und müssen manuell verifiziert bzw. korrigiert werden.

[3]Diese stehen in den Dateien dictionary/data/prefixes.dat sowie dictionary/data/postfixes.dat

[4]Verkürzt wird nur bis zu einer Mindestlänge von drei Buchstaben.

3. **Vorschlagen:**
   Hier wird eine schrittweise *Graphem-zu-Phonem*-Umsetzung des Wortes vorgenommen. Dazu werden in einer Datei[5] Buchstabenfolgen Phonemfolgen zugeordnet, wobei die Buchstabenfolgen nach absteigender Länge sortiert sind. Falls eines der Grapheme auf den Wortanfang paßt, wird es vom Wort abgetrennt, die zugehörige Phonemfolge gemerkt und für den Rest des Wortes das Verfahren erneut gestartet. Die so ermittelten Phonemfolgen werden zusammengesetzt und als gefundene Aussprache zurückgegeben. Da sämtliche Buchstaben des Alphabets in der Abbildungsdatei enthalten sind, ist ein Terminieren des Verfahrens[6] bei Buchstabenfolgen gesichert. Damit das Verfahren aber auch bei unsinnigen Eingaben (Sonderzeichen etc.) terminiert, wird das zu untersuchende Wort nach erfolglosem Schleifendurchgang um eine Stelle (am Wortanfang) verkürzt.

   Die Zuverlässigkeit dieses Verfahrens steht und fällt mit der Qualität der Abbildungsdatei. Aufgrund der nicht eineindeutigen Abbildung eines geschriebenen Wortes auf die zugehörige Phonemfolge, zumindest in der deutschen Sprache, kann eine 100-prozentige Zuverläßigkeit wohl jedoch nicht erreicht werden. Aus diesem Grund werden die Vorschläge dieses Ansatzes als unsicher gekennzeichnet.

[5] dictionary/data/maptophone.dat

[6] Es wird eine while-Schleife der Form `while not Word is Empty` verwendet.

2.3 Programmbeschreibung

In diesem Abschnitt wollen wir in groben Zügen die Bedienung des Programms beschreiben.

2.3.1 Die graphische Benutzerschnittstelle

Die Arbeitsfläche (siehe Bild 1) besteht aus drei Hauptteilen:
Der obere Teil zeigt einen Ausschnitt der zu bearbeitenden Datei. Jede Zeile in diesem Bereich besteht aus einem kleinen Quadrat, sowie zwei Textelementen. Das Quadrat trägt (per Farbe) den Bearbeitungsstatus des betreffenden Wortes[7]. Mögliche Werte sind:

- weiß: Die Aussprache zu diesem Wort ist unbekannt.
- grau: Die zugehörige Aussprache ist unsicher.
- schwarz: Die Aussprache ist bekannt.

Die beiden Textelemente auf der rechten Seite dieses Quadrats enthalten das Wort sowie – falls vorhanden – seine zugehörige Aussprache.

Begrenzt wird der obere Teil der Arbeitsfläche durch einen Block von Navigationsknöpfen, die es erlauben, sich durch die Datei zu bewegen[8].

Der mittlere Teil der Arbeitsfläche bildet die eigentliche Bearbeitungsfläche. Er besteht aus einer Zeile, die das aktuell bearbeitete Wort, seine Aussprache (soweit vorhanden) und einen *accept*-Knopf zeigt. Dazu kommen fünf Zeilen, die die momentanen Teilwörter des aktuellen Wortes, ihre zugehörigen Aussprachen sowie verschiedene Aktionsknöpfe enthalten. Zur praktischen Benutzung dieser Komponenten siehe Kapitel 2.3.5.

Der untere Teil der Arbeitsfläche stellt eine Suchfunktion bereit, mit deren Hilfe in den ausgewählten, bestehenden Wörterbüchern Aussprachen zu beliebigen Wörtern gesucht werden können. Mehr darüber in Kapitel 2.3.5.

[7] Durch Mausklick auf das Quadrat kann der Status verändert werden!

[8] Ihre Bedeutung ist von links nach rechts: *zum Dateianfang, eine Seite nach oben, einen Eintrag nach oben, einen Eintrag nach unten, eine Seite nach unten* und *zum Dateiende.*

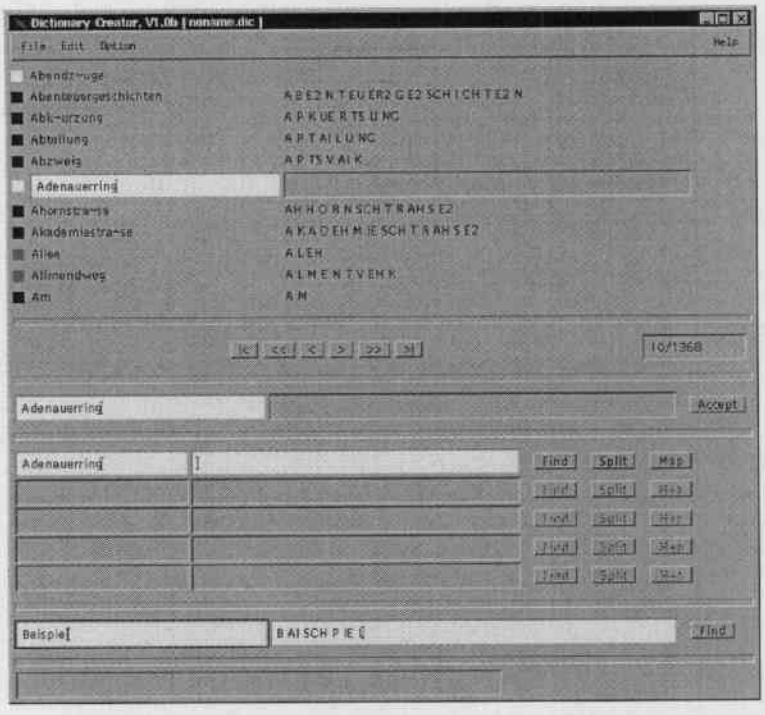

Abbildung 1: Die graphische Benutzerschnittstelle von *DictCreator*

Die Menüleiste besteht aus vier Menüs. Diese heißen im Einzelnen: **File**, **Edit**, **Option** und **Help**. Gehen wir kurz auf die interessanten Aspekte ein. Der Eintrag *Automode* im Menü **Edit** öffnet einen Dialog, der zur Automatisierung der Aussprache-Findung dient. Die Automatisierung wird in Abschnitt 2.3.4 näher beschrieben.
Das Menü **Options** enthält einen Eintrag *Dictionary Files*. Die Anwahl dieses Eintrags führt zu einem Dialogfenster, mit dessen Hilfe man die (bestehenden) Wörterbuchdateien auswählen kann, in denen nach Aussprachen gesucht werden soll. Der Abschnitt 2.3.3 beschreibt diese Auswahl näher. Zusätzlich findet man im Menü **Options** einen Eintrag *Options*, unter dem sich ein Dialogfenster zur Festlegung genereller Optionen öffnet (siehe Bild 2). Dazu gehören: *Breite des Kontextbereichs*, *Anzahl Teilwörter*, *Beachtung der Groß-/Kleinschreibung bei der Suche* sowie der *Ort*, an dem die Wörterbücher gehalten werden sollen (also Speicher oder Festplatte). Desweiteren kann man in diesem Dialog die Wort-Anzeige filtern; dabei lassen sich die unbekannten,

die unsicheren und die bekannten Wörter separat an- und abwählen.

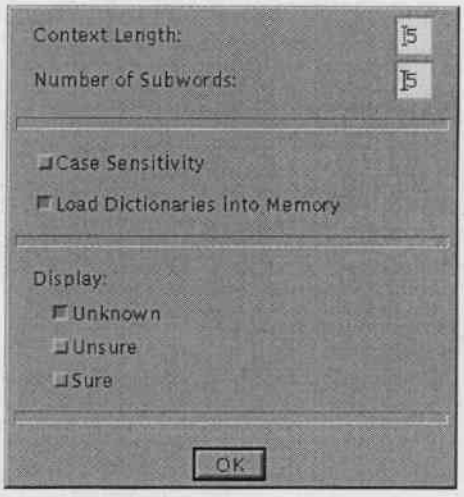

Abbildung 2: Optionen Dialog

2.3.2 Ein-/Ausgabedateien

Die zu bearbeitende Datei wird über den **Datei öffnen** Dialog eingeladen. *DictCreator* ermittelt dabei automatisch den passenden Importfilter, in dem es der Reihe nach die dem Programm bekannten Filter prüfen läßt, ob sie das Format lesen können. Wird kein passender Filter gefunden, gibt das Programm eine entsprechende Fehlermeldung aus. Momentan kennt *DictCreator* drei Importfilter:

DictCreator:
Dieses programmeigene Format erlaubt die Speicherung sämtlicher Informationen, einschließlich der Statusindikatoren.

Janus:
Mit diesem Filter können Janus-Standard-Wörterbücher eingelesen werden, um z.B. fehlende Wörter zu ergänzen.
Wichtig: dieses Format speichert nicht die Statusindikatoren.

pure vocabulary:
Lädt einfache „ein Wort pro Zeile"-Vokabulardateien.

Zur Integration neuer Filter sei auf Abschnitt 2.4.2 verwiesen, in dem die verwendete Schnittstelle ausführlich beschrieben wird.

2.3.3 Verwendung bestehender Wörterbücher

Wie bereits erwähnt, können aus bestehenden Wörterbüchern bekannte Aussprachen extrahiert werden. Über das Menü *Options/Dictionary Files* gelangt man zu einem Dialog, der die verwendeten Wörterbücher auflistet. Da in den Wörterbüchern in der Reihenfolge gesucht wird, wie sie in dieser Liste erscheinen, bietet der Dialog die Möglichkeit, die Reihenfolge der Einträge zu verändern[9].

Natürlich können Wörterbücher aus der Liste wieder entfernt (**Remove**-Knopf) und neue hinzugefügt werden. Das Hinzufügen geschieht über den **Add**-Knopf, der einen Dateidialog zur Auswahl einer Wörterbuchdatei öffnet.

DictCreator kennt zur Zeit folgende Formate[10]:

- Janus-Standard Wörterbuch
- CELEX-Wörterbuch
- GSST-Wörterbuch

2.3.4 Automatisierung

Mit *DictCreator* lassen sich Teile der Wörterbucherstellung automatisieren. Dies geschieht über den **Automode**-Dialog (siehe Bild 3). Zur Automatisierung müssen drei Parameter angegeben werden.
Zuerst die Methode der Aussprache-Findung. Die in *DictCreator* bekannten sind in 2.2 genauer erläutert.

[9]Dies geschieht über die beiden Knöpfe **Up** und **Down**, die einen Eintrag nach oben bzw. unten schieben.

[10]Es ist zu beachten, daß die Dateien bei allen drei Formaten lexikalisch sortiert vorliegen müssen, da Aussprachen mittels Binärsuche gesucht werden!

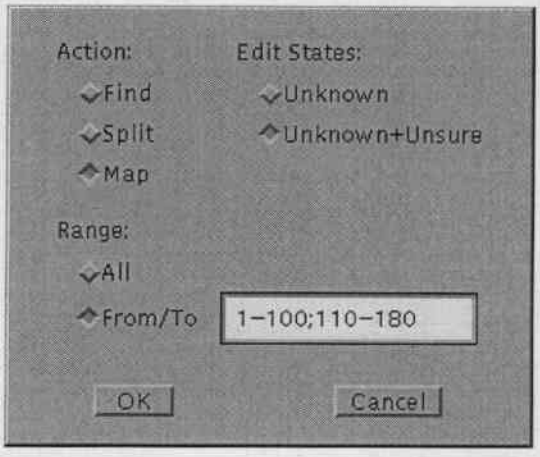

Abbildung 3: Der Automatisierungs-Dialog

Zweitens muß der Status der Wörter angegeben werden, die bearbeitet werden sollen. Zur Auswahl stehen einerseits *nur Wörter mit unbekannter Aussprache* und andererseits *Wörter mit unbekannter oder unsicherer Aussprache.*
Drittens muß der Bereich angegeben werden, der bearbeitet werden soll. Entweder die gesamte Datei (*All*), oder nur einzelne Bereiche (*From/To*). Diese Bereiche werden, wie in Bild 3 dargestellt, durch mit Semikoli getrennte Zahlenbereiche angegeben, wobei die Zahlen den Zeilennummern der Wörter entsprechen.

Während des Ablaufs der automatischen Aussprache-Findung wird ein kleines Dialogfenster angezeigt, das den Benutzer über das momentan bearbeitete Wort informiert. Zusätzlich enthält es einen Knopf, der den Abbruch der automatischen Suche ermöglicht.

2.3.5 Praktische Benutzung

In diesem Abschnitt sollen ein paar Tips gegeben werden, wie man das Programm effektiv zur Wörterbucherstellung benutzen kann.

Nachdem die Vokabulardatei eingeladen ist und die Wörterbücher, in denen gesucht werden soll, ausgewählt sind, wird in einem ersten Schritt der auto-

matische Modus mit der Methode *Find* benutzt. So werden alle bekannten Aussprachen aus den gewählten Wörterbüchern zusammengesammelt.

Danach wendet man den automatischen Modus mit der Methode *Split* an, die durch rekursives Auftrennen versucht, Aussprachen zu finden. Die so gefundenen Aussprachen werden als unsicher markiert, da das Auftrennen nicht in jedem Fall das korrekte Ergebnis liefert[11]

Jetzt wendet man ein letztes Mal den automatischen Modus an: Dieses Mal mit der Graphem-zu-Phonem-Umsetzung.

Im nächsten Schritt geht man mit den Navigationsknöpfen durch die Datei. Dabei können über den Anzeigefilter die sicheren Aussprachen ausgeblendet werden, so daß bequem die unbekannten Aussprachen ergänzt bzw. die unsicheren Aussprachen verifiziert und ggf. verbessert werden können.

Dazu ein paar Worte zur Bedeutung der einzelnen Textelemente:

Das Texteingabefeld am linken Rand in der Zeile mit dem *accept*-Knopf dient dem manuellen Auftrennen eines zusammengesetzten Wortes mittels der Leertaste.
Die entstehenden Teilwörter (maximal fünf) werden in den Zeilen darunter angezeigt. Dort können die Aussprachen zu den Teilwörtern einzelnen bearbeitet werden. Dies geschieht entweder manuell, oder über einen der drei Knöpfe, die sich am rechten Rand in der Zeile mit dem Teilwort befinden. *Find* sucht das Teilwort in den ausgewählten Wörterbüchern, *Split* versucht die Aussprache des Teilworts mit der Trennmethode zu finden und *Guess* versucht, eine Aussprache über die Graphem-zu-Phonem-Umsetzung zu raten.
Die Aussprachen der Teilwörter werden automatisch zusammengefügt und in dem Textfeld links neben dem *accept*- Knopf angezeigt. Ist diese Gesamt-Aussprache in Ordnung, kann sie durch Drücken auf den genannten Knopf in die Datei übernommen werden.

Es ist zu beachten, daß man die Teilwörter selbst auch editieren kann; dies ist praktisch, um etwa ein Plural-s zu entfernen oder ein ähnliches Wort zu suchen. Dadurch wird aber nicht die Schreibweise des Wortes innerhalb der Datei verändert.

[11] Enthält ein Wörterbuch z.B. die Wörter *Gefahr* $\mapsto$ *G E2 F AH R* und *en* $\mapsto$ *O NG*, so liefert die Suche nach *Gefahren* möglicherweise fälschlich *G E2 F AH R O NG*

In der manuellen Editierphase sollte man reichlich Gebrauch von der Möglichkeit machen, beliebige Wörter in den Wörterbüchern suchen zu können (die Textfelder zwischen Teilwörter-Bereich und Statusanzeige). So kann man leicht nach ähnlichen Wörtern suchen und die Aussprache dann zusammensetzen. Sucht man z.B. eine Aussprache für das Wort *glove*, so wird man – vorausgesetzt dieses Wort ist in den bestehenden Wörterbüchern unbekannt – etwa nach *dove* suchen und, im Erfolgsfall, leicht die Aussprache für *glove* vervollständigen.

2.4 Schnittstellen für Dateifilter

2.4.1 Allgemeines

Wie wir eingangs erkannt hatten, wird die allgemeine Verwendung[12] von *DictCreator* die Notwendigkeit neuer Dateifilter – sowohl die Ein-/Ausgabedateien, als auch die Wörterbücher betreffend – mit sich bringen. Zu diesem Zweck ist eine flexible, einfache Schnittstelle notwendig, an die sich alle verwendeten Filter halten. In den folgenden beiden Abschnitten wollen wir die Schnittstellen genauer betrachten, wie sie in *DictCreator* zum Einsatz kommen.
Praktisch wird die Einhaltung der Schnittstelle so realisiert, daß in einer Java-Interface-Klasse die Methoden der Schnittstelle festgelegt werden, die dann von den Filtern implementiert werden müssen.

2.4.2 Ein-/Ausgabeformate

Die Schnittstelle für die Ein-/Ausgabedatei-Filter sind folgendermaßen aufgebaut:

```
package fileio.inoutformats;

public interface InOutFormatI
{
    public boolean checkFormat(File name);
    public String getFormatName();

    public Vector[] loadFile(File name);
    public boolean saveFile(File name, Vector[] data);
}
```

Beschreiben wir zuerst die Bedeutung der Schnittstellenmethoden und geben dann abschließend einige Hinweise zur Integration neuer Formate.

checkFormat:
Als Parameter erhält diese Methode einen Dateinamen. Zurückgegeben

[12] etwa die Verwendung mit anderen Sprachen

werden soll ein Wahrheitswert, der angibt, ob diese Datei unserem[13] Format entspricht. Benutzt wird diese Methode von *DictCreator*, um automatisch den richtigen Filter für eine vom Anwender gewählte Datei benutzen zu können.

getFormatName:
Rückgabewert dieser Methode ist eine String-Variable, in der der Name unseres Formats abgelegt ist. Dieser Name wird beim Speichern der Datei zur Auswahl des Formats benötigt.

loadFile:
Diese Methode dient dem Einladen der ausgewählten Datei, deren Name als Parameter übergeben wird. Zurückgegeben werden muß ein Vektor-Feld (Array) der Größe drei. Diese Vektoren enthalten mit aufsteigendem Index: Zustandsindikatoren, Wörter, Aussprachen. Werden bei unserem Format keine Indikatoren gespeichert, so müssen wir (günstige) Dummywerte eintragen. Günstig heißt: Bei nichtvorhandener Aussprache wird der Indikator auf null (*unbekannt*) gesetzt; bei sicheren Aussprachen wählen wir eine zwei und anderenfalls, im Zweifel eine eins (*unsicher*).

saveFile:
Hiermit speichern wir eine bearbeitete Datei in unserem Format unter dem angegebenen Namen *name*. Der zweite Parameter übergibt der *saveFile*-Methode die Daten, die zu speichern sind; auch hier als Vektor-Feld der Länge drei: Indikatoren, Wörter, Aussprachen. Was davon in unserem Format gespeichert wird, bleibt uns überlassen.

Bei der Integration neuer Filter muß lediglich dafür gesorgt werden, daß die beschriebene Schnittstelle von unserer Java-Klasse implementiert wird. Dann müssen wir unseren neuen Filter nur noch in das Verzeichnis fileio/inoutformats/ kopieren, wobei der Dateiname die Konvention InOutFormat*.class erfüllen muß. Ab dem nächsten Programmstart wird dieser Filter automatisch als solcher erkannt und bei Bedarf verwendet.

[13] Wenn hier von *unserem* Format die Rede ist, so ist damit jenes Format gemeint, das wir mit unserem Filter zugänglich machen wollen.

2.4.3 Wörterbuchformate

Die Schnittstelle für die Wörterbuchformat-Filter hat das folgende Aussehen:

```
package dictionary.dictformats;

public interface DictFormatI
{
    public boolean checkFormat(File name);
    public String getFormatName();

    public boolean openDictionary(File name, boolean memory);
    public boolean closeDictionary();

    public String findPronounciation(String word,
                                     boolean caseSensitive);

    public String getID();
}
```

Auch hier zuerst eine Beschreibung der Schnittstellenmethoden und abschließend einige Hinweise zur Integration neuer Formate.

checkFormat:
Als Parameter erhält diese Methode einen Dateinamen. Zurückgegeben werden soll ein Wahrheitswert, der angibt, ob diese Datei unserem Format entspricht. Benutzt wird diese Methode von *DictCreator*, um automatisch den richtigen Filter für ein vom Benutzer gewähltes Wörterbuch benutzen zu können.

getFormatName:
Rückgabewert dieser Methode ist eine String-Variable, in der der Name unseres Formats abgelegt ist. Momentan wird diese Methode nicht benutzt. Sie ist für mögl. Erweiterungen vorgesehen.

openDictionary:
Benutzt wird diese Methode, um den Zugriff auf ein Wörterbuch zu initiieren. Übergeben wird ein Dateiname sowie ein Wahrheitswert, mit dem festgelegt wird, ob das Wörterbuch bevorzugt in den Speicher

geladen werden soll. Von Bedeutung ist diese Variable nur, wenn unser Filter sowohl die Möglichkeit bietet, das Wörterbuch in den Speicher zu laden als auch auf der Festplatte zu belassen. Wird nur eine der beiden Möglichkeiten angeboten, kann *memory* ignoriert werden.

closeDictionary:
Sämtliche Datenstrukturen (Hashtabellen, Dateihandles, etc.) eines ausgewählten Wörterbuchs werden mit dieser Methode aus dem Speicher gelöscht. Der Zugriff auf die Datei ist beendet.

findPronounciation:
Diese Methode bildet den Kern aller Wörterbuchformat-Filter. Sie bildet eine Schnittstelle zur Suche nach Ausprachen in unserem Wörterbuch. Wie die Suche organisiert wird, bleibt uns überlassen (Hashtabelle, Binärsuche, etc.). Es muß lediglich dafür gesorgt werden, daß im Erfolgsfall die Aussprache des gesuchten Wortes und im Nichterfolgsfall ein leerer String zurückgegeben werden. Eingangsparameter sind zum einen der Suchstring und zum anderen ein Wahrheitswert, der festlegt, ob die Groß-/Kleinschreibung bei der Suche berücksichtigt werden soll, oder nicht. Diese Funktionalität wird für die Suchmethode des rekursiven Auftrennens benötigt.

getID:
Hiermit soll eine eindeutige Identifikation der ausgewählten Wörterbücher ermöglicht werden. Man wird im allgemeinen den vollständigen Pfad des Wörterbuchs zurückliefern.

Bei der Integration neuer Filter gilt dasselbe wie bei den Ein-/Ausgabedatei-Filtern: Es ist dafür zu sorgen, daß die beschriebene Schnittstelle von unserer Java-Klasse implementiert wird. Dann muß der neue Filter in das Verzeichnis dictionary/dictformats/ kopiert werden, wobei der Dateiname die Konvention DictFormat*.class erfüllen muß. Ab dem nächsten Programmstart wird dieser Filter automatisch als solcher erkannt und bei Bedarf verwendet.

3 Spracherkennung in Automobilen

3.1 Die Datenbasis

Die Datenbasis, die dem hier vorgestellten Spracherkennungssystem zugrundeliegt, soll vorab kurz beschrieben werden.[14] Sie besteht aus 100 Sprechern, davon 32 weiblich und 68 männlich. Das Durchschnittsalter beträgt 27,8 Jahre.
Zur Aufnahme standen 4 verschiedene Fahrzeuge zur Verfügung: *ein BMW 540, ein VW Passat TDI, ein Renault Laguna* und *ein Audi A6*. Die Sprachdaten wurden während der Fahrt mittels eines DAT-Recorders aufgezeichnet. Es kamen dabei mehrere Mikrophone zum Einsatz, die an verschiedenen Stellen im Fahrzeug platziert wurden. In dieser Arbeit wurden die Aufnahmen der Mikrophonkanäle eins und zwei verwendet. Das Mikrophon des Kanal eins – dem sog. Closetalk-Kanal – befand sich direkt vor dem Mund des Fahrers (Headset). Die Aufnahmen des zweiten Kanals entstammen einem Mikrophon, das oberhalb der Windschutzscheibe auf der Fahrerseite angebracht worden war (am A-Holm).

ID	Bedeutung
A	isolierte Buchstaben
L	Buchstabenfolgen
I	isolierte Ziffern
C	Ziffernfolgen
W	Kommandowörter
E	Sätze, die Kommandowörter enthalten
P	Eigennamen
S	phonemreiche Sätze

Tabelle 1: Äußerungsklassen der Datenbasis

Die Gesamtzahl der gespendeten Äußerungen beträgt 13662; also 130-140 Äußerungen pro Sprecher. Dabei sind die Äußerungen in acht festgelegte

[14] Wir verwenden hier die Datenbasis, die dem Projekt VODIS (*Voice Operated Driver Information System*) zugrundeliegt.

Klassen eingeteilt: Siehe hierzu Tabelle 1.

Zusätzlich zu den eigentlichen Sprachdaten wurden wesentliche – also die Aufnahme beeinflussende – Umgebungsparameter mitprotokolliert. Dazu gehören u.a. *Radio, Fenster, Lüftung, Fahrgeschwindigkeit, Straßenbelag, Wetter.*

Zur Entwicklung des Spracherkennungssystems wurde diese Datenbasis, wie üblich, in drei Mengen aufgeteilt. Die Trainingsmenge (80 Sprecher), die Kreuzvalidierungsmenge (10 Sprecher) und die Testmenge (10 Sprecher)[15]. Bei der Einteilung wurden folgende Bedingungen berücksichtigt. Die Mengen müssen sprecherdisjunkt sein. Die Geschlechtsverteilung in den Mengen soll etwa gleich sein. Ebenso soll die Verteilung der Hauptstörquellen (*Radio, Lüftung, Fenster*) gleichmäßig sein.
Ein weiteres, wesentliches Kriterium kann bei der Einteilung jedoch nicht erfüllt werden: Die Disjunktheit der Äußerungen. Dies muß an der Einteilung in acht Äußerungsklassen und dem insgesamt relativ beschränkten Vokabular[16] scheitern.

Sprecher	100
Äußerungen	13662
Vokabular	1207 Wörter
Perplexiät	13.55
OOV	1.323 %

Tabelle 2: Datenbasis – Eckdaten

Die Perplexität dieses Systems liegt bei 13.55; die *out-of-vocabulary-rate* ist 1.323 %. Eine Zusammenfassung der Eckwerte zur Datenbasis kann in Tabelle 2 gefunden werden.

[15] Auf die genaue Bedeutung dieser Mengen soll nicht näher eingegangen werden.
[16] Insgesamt enthält die Datenbasis einen Vokabularumfang von 1207 Wörtern.

3.2 Der Einsatz von *DictCreator*

Das Vokabular der Datenbasis umfaßte 1207 Wörter. Bei der Erstellung des Wörterbuchs mit *DictCreator* wurde folgendermaßen vorgegangen:

Als Grundlage zur Erstellung dienten zwei bereits bestehende Wörterbücher – eines aus dem GSST-System (ca. 7800 Wörter) und ein CELEX-Wörterbuch (rund 320.000 Einträge).

Die normale Suche in den beiden Wörterbücher lieferte Ausprachen zu 944 der 1207 Wörter – eine Erfolgsquote von rund 78.2 %.

Im nächsten Schritt wurde die Auftrennsuche auf die restlichen 263 Wörter angewandt (ebenfalls über beiden Wörterbüchern). Dies lieferte weitere 134 Aussprachen (absolut: 11.1 %). Bei den hier erkannten Wörtern handelt es sich durchweg um komponierte Nomen (z.B. *Abenteuergeschichten* oder *Akademiestraße*).

Anzahl	Typ
57	Straßen-/Ortsnamen (auch Fragmente)
41	Vor-/Nachnamen
13	Nomen, die den bestehenden Wörterbüchern unbekannt sind. darunter: *Bußgeldes, Cassette, Eszett, Pictogramm*
5	apostrophierte Wörter im Einzelnen: *ein hab is nich un*
4	Spracherkenner-spezifische Wörter
4	sonstige Eigennamen
3	Akronyme
2	Firmennamen

Tabelle 3: Struktur der nach den Suchvorgängen unerkannten Wörter

Die Untersuchung der bisher unerkannten Wörter ergab das in Tabelle 3 dargestellte Bild. Man stellt fest, daß ca. 80 % dieser Wörter Eigennamen verschiedener Natur sind. Natürlich kann die Suche in den bestehenden Wörterbüchern zu diesen Eigennamen keine Aussprache liefern, da jene Wörterbücher nur *Standard*-Nomen enthalten.

bee	B EH	**tz**	TS	**ach**	A X	**amm**	A M	**wen**	W E N
hov	H OH F	**th**	T	**bast**	B A S T	**itz**	I TS	**lly**	L IE
beh	B EH	**dt**	T	**oeh**	OEH	**chs**	K S	**ich**	I CH
ai	AI	**err**	E R	**rch**	R CH	**rr**	R	**ver**	F E R
ais	AI S	**wer**	V E R	**uest**	UE S T	**ph**	F	**rin**	R I N
nk	NG K	**je**	J E	**ay**	AI	**ien**	IE E2 N	**len**	L E N
eh	EH	**ann**	A N	**nch**	N CH	**ier**	IE E2 R	**hen**	H E N

Tabelle 4: Erweiterung der Graphem-zu-Phonem-Tabelle

Dies bedeutet gleichzeitig, daß die Suche (normal plus rekursiv), die Eigennamen außen vor gelassen, eine beachtliche Trefferrate von 98% erzielt.

Von Interesse war als Nächstes, wie zuverlässig die Graphem-zu-Phonem-Umsetzung der restlichen 129 Wörter arbeitete und wo hierbei die Probleme lagen.
Wie erwartet, lieferte dieses Verfahren, relativ gesehen, weitaus schlechtere Ergebnisse als die vorausgegangenen Suchvorgänge. In Zahlen heißt das, daß 79 der 129 Wörter – mehr oder weniger – falsch umgesetzt wurden; eine Erfolgsrate von etwa 38.8 %. Die Untersuchung der Probleme ergab folgendes Ergebnis:

Den größten Teil der falsch umgesetzten Wörter machten Wörter aus, die auf *-e* enden; darunter die Straßeneigennamen (*-straße*). Bei diesen Wörtern wurde für dieses *e* nicht das richtige *schwa*-Phonem *E2* verwendet, sondern fälschlicherweise das Phonem *E*. Zur Beseitigung dieses Fehlers könnte man beispielsweise das Wortende als zusätzliches Zeichen einführen und „e gefolgt von Wortende“ in *E2* umsetzten. Bei einer allgemeinen Berücksichtigung des Wortendes bei der Umsetzung, würden weitere 26 Wörter richtig erkannt. Insgesamt würde dies im vorliegenden Fall die Erkennungsrate von den festgestellten 38.8 % auf 58.9 % steigern. Eine Erweiterung der Graphem-zu-Phonem-Tabelle um die in Tabelle 4 gezeigten Einträge würde die Rate um weitere 34 richtig umgesetzte Wörter auf 85.3 % steigern.

An die Grenzen seiner Möglichkeiten stößt dieses Umsetzungsverfahren in folgenden Fällen:

- Akronyme (hier 4 Stück)
- Fremdwörter (im Detail: *Joy, Capri, Life, Kiecza*)

Ideen zur allgemeinen Verbesserung der Graphem-zu-Phonem-Umsetzung werden in Kapitel 4 angedacht.

3.3 Ergebnisse

In diesem Abschnitt sollen Erkennungsresultate präsentiert werden, die mit den beiden entwickelten Erkennungssystemen erzielt wurden. Es wurde für jeden der beiden Mikrophonkanäle getrennt ein Erkenner trainiert und auf Daten des jeweils gleichen Kanals getestet. Die Ergebnisse sind in Tabelle 5 zu finden.

Die erste Spalte der Tabelle gibt die Äußerungsklasse an. In der zweiten Spalte steht die Anzahl der Äußerungen. In den Spalten drei und vier sind die Erkennungsleistungen (in Prozent) zu den Kanälen eins und zwei zu finden.

Klasse	Anzahl	K 1 (%)	K 2 (%)
A	320	53.3	45.5
L	54	82.3	73.5
I	108	78.8	72.2
C	82	95.6	87.9
W	686	95.4	89.3
E	28	96.4	88.2
P	50	60.4	49.1
S	50	97.3	89.9
Total	1378	88.7	81.2

Tabelle 5: Ergebnisse

Bemerkenswert ist die durchweg (um durchschnittlich 8 %) bessere Erkennungsrate auf dem ersten Kanal im Vergleich zum zweiten Kanal. Dies ist

jedoch einsichtig, da erstens das Sprachsignal wesentlich schwächer ist am Mikrophon des zweiten Kanals und zweitens die Störgeräusche im Falle des zweiten Kanals **zwischen** dem Sprecher und dem Mikrophon liegen. Einige Möglichkeiten zur Weiterentwicklung des Systems werden in Kapitel 4 diskutiert.

4 Zusammenfassung und Ausblick

DictCreator:

Wie aufgezeigt wurde, ermöglicht das Programm *DictCreator* eine effizientere und komfortablere Erstellung von Wörterbüchern, als es etwa mit gewöhnlichen Texteditoren möglich ist. Dies wird vor allem durch das einfache Bearbeiten von Teilworten eines gegebenen Wortes sowie die diversen – automatisierbaren – Aussprache-Findungsverfahren erreicht.

Zukünftige Verbesserungen des Programms können verschiedene Aspekte betreffen. Beispielsweise wäre es denkbar ein komplexeres Verfahren zum rekursiven Auftrennen zu entwickeln, das ein Wort in die bekannten und unbekannten Teilwörter auftrennt, und der Benutzer so nur die unbekannten Teilwörter bearbeiten müßte.
Die Graphem-zu-Phonem-Umsetzung bietet natürlich auch Möglichkeiten zur Verbesserung. Denkbar ist eine Erweiterung der bestehenden Umsetzungstabelle, die Hinzunahme der Wortgrenze als eigenständiges Zeichen oder beispielsweise ein Werkzeug, das anhand eines bestehenden Wörterbuchs eine Umsetzungstabelle erzeugt.

Allgemein könnte es Vorteile bringen, nicht nur das Wortende als Kontext zu verwenden, sondern prinzipiell Kontexte einzusetzen. Als Beispiel denke man an die beiden Wörter *holen* und *Holz*. Im ersten Fall – also **ol** mit Kontext **e** – wird das *o* in ein langes *OH* umgesetzt; im zweiten Fall – hier haben wir als Kontext **z** – ist das *o* kurz: *O*.

Man wird jedoch auch durch den Einsatz eines solchen Kontexts keine hundertprozentige Zuverlässigkeit errreichen. Dies verhindern die im vorigen Kapitel erwähnten Problemfälle – also Fremdwörter, Eigennamen und Akronyme. Hinzu kommen aber noch Wörter, die sich zwar in der Aussprache unterscheiden, nicht aber in der Schreibweise, als Beispiel seien hier *weg* und *Weg* genannt.

Zusätzliche Verbesserung könnten evtl. anders geartete Umsetzungsverfahren bringen. Beispielsweise könnte man von der hier verwendeten *first-fit*-Strategie[17] Abstand nehmen und eine *best-fit*-Stragie verwen-

[17] Das bedeutet, daß das Verfahren, den ersten passenden Umsetzungseintrag aus der Tabelle gleich verwendet

den: Man erstellt **alle** – mit der gegeben Umsetzungstabelle – möglichen Aussprachen und entscheidet sich für jene mit der besten Bewertung. Dazu muß jeder Eintrag um eine Bewertung erweitert werden. Denkbar wäre z.B. der Kehrwert der Länge des Graphems. Ein anderer interessanter Ansatz zur Umsetzung könnte der Einsatz eines Neuronales Netzes sein.

Natürlich wäre selbst an eine Kombination des Auftrennens und der Umsetzung zu denken. Man könnte durch das Auftrennen bekannte Teilwörter in Aussprachen umsetzen und die nicht bekannten über das Graphem-zu-Phonem-Verfahren erhalten.

Die Entwicklung von Spracherkennungssystemen in anderen Sprachen wird natürlich ebenfalls Erweiterungen des Programms bedingen; hier sei etwa an neue Wörterbuchformate gedacht.

Letztlich wäre es sogar möglich, *DictCreator* für andere Abbildungen als die Zuordnung eines Wortes zu seiner Aussprache zu benutzen; es sei hier an die Zuordnung eines Wortes zu seiner morphologischen Zerlegung gedacht.

Spracherkenner:

Die präsentierten Ergebnisse des vorliegenden Erkennungssystems können als gut für die gegebene (eingeschränkte) Datenbasis bezeichnet werden. Wie die Aufschlüsselung der Resultate nach Äußerungsklassen aber zeigt, sind es vor allem die kurzen Äußerungen, also etwa die Buchstaben, die die Gesamterkennungsrate verschlechtern. Hier kann z.B. Abhilfe geschaffen werden, indem ein spezieller Buchstabiererkenner in das System integriert wird, der zur Erkennung von Buchstabiersequenzen verwendet wird. Das Problem hierbei wird sein, die Zeitpunkte zu erkennen, bei denen zwischen dem normalen und dem Buchstabiererkenner hin- und hergeschaltet werden soll. Hier könnte etwa ein für diesen Zweck zugeschneidertes Neuronales Netzwerk zur Anwendung kommen.

Ein Vergleich der beiden Mikrophonkanäle weist auf eine andere Möglichkeit zur Verbesserung des Systems hin: Weiterentwicklung der Signalvorverarbeitung des Sprachsignals. Dies werden beispielsweise Methoden zur Geräuschreduktion sein.

Langfristig Ziel ist es, das starre Kommandogerüst der verbalen Bedienung zu verlassen, um dem Benutzer spontane Anfragen an das System zu ermöglichen.