EMU Speech Database System – Praxisorientierte Weiterentwicklung der Funktionalität, Benutzerfreundlichkeit und Interoperabilität sowie die Aufbereitung des Kiel Corpus als EMU-Sprachdatenbank

Tina John

München 2012

EMU Speech Database System

Praxisorientierte Weiterentwicklung der Funktionalität, Benutzerfreundlichkeit und Interoperabilität sowie die Aufbereitung des Kiel Corpus als EMU-Sprachdatenbank

Dissertation
zur Erlangung des Doktorgrades
der Philosophie an der Ludwig-Maximilians-Universität
München

vorgelegt von
Tina John

aus Kronshagen

2012

Erstgutachter: Prof. Dr. Jonathan Harrington
Zweitgutachter: PD Dr. Christoph Draxler

Tag der mündlichen Prüfung: 13.02.2012

für meine Familie
– hinterm Regen scheint die Sonne uns wieder –

Hinterm Regen scheint die Sonne, hörst du
hinterm Regen reisst der Himmel auf,
hinterm Regen sieht das Leben besser aus.

Hinterm Regen scheint für dich wieder Licht,
du musst nur kämpfen und glauben
nur stillstehen darfst du nicht!

Tefla & Jaleel

Inhaltsverzeichnis

Quellcodes und Dateiinhalte

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Bsp.	Beispiel
d. h.	das heißt
etc.	et cetera
ff.	und folgende
ggf.	gegebenenfalls
iE	im Erscheinen
inpr	im Druck
Jhdt.	Jahrhundert
sog.	so genannte
u. a.	unter anderen
u. U.	unter Umständen
usw.	und so weiter
vgl.	vergleiche
z. B.	zum Beispiel

Zusammenfassung

Gegenstand dieser Dissertation ist die Weiterentwicklung des EMU Speech Database Systems, einer *open source* Software für den Einsatz in Lehre und Forschung auf dem Gebiet der Phonetik sowie der Linguistik. Die Besonderheit dieser Arbeit ist der interdisziplinäre Ansatz[1], da der Entwickler gleichzeitig Benutzer der Software ist. Der Forscher der Phonetik und Linguistik als Benutzer konnte somit als Entwickler in der technischen Weiterentwicklung direkt die eigenen Ansprüche an das geisteswissenschaftliche Forschungswerkzeug umsetzen.

Ausgangspunkt für die Weiterentwicklung war ein bereits vorhandenes Softwaresystem, das den Forscher beim Aufbau und damit verbunden bei der Etikettierung, der Abfrage sowie der Analyse von Sprachdatenbanken unterstützt. Es stellte als primäre Schnittstelle zum Anwender eine grafische modular erweiterbare Oberfläche zur manuellen sowie skriptbasierten Etikettierung von Sprachsignalen bereit. Weiterhin bot das EMU-System die Möglichkeit, Etikettierungen sehr vielseitig zu strukturieren und stellte gleichzeitig eine Abfragesprache für diese Etikettierungen zur Verfügung. Darüber hinaus befolgte die existierende Software bereits das Prinzip der getrennten Verwaltung von Datendefinition und Daten.

Vor der Weiterentwicklung des EMU-Systems konnte die Datendefinition nur textbasiert durchgeführt werden. Wie die eigene Erfahrung zeigte, setzte dies ein ausführliches Studium des Benutzerhandbuches oder eine intensive Programmeinführung im Rahmen der Lehre voraus. Die eigentliche Forschungsarbeit wurde somit dem Studium der Software nachgestellt. Ähnliches gilt für die Verwendung der Abfragesprache.

Aus diesem Grund und mit dem Bestreben, die genannten funktionalen Eigenschaften des Systems den Forschern in benutzerfreundlicher Form zur Verfügung zu stellen und um für die wachsenden Funktionalitätsansprüche durch neue Forschungsfragen

[1] Um sowohl der technischen als auch geisteswissenschaftlichen Ausrichtung Rechnung zu tragen, steht diese Zusammenfassung entgegen streng geisteswissenschaftlicher Tradition am Anfang der Arbeit.

Lösungen anzubieten, wurde die Weiterentwicklung des EMU-Systems in Angriff genommen. Im Rahmen dieser Dissertation wurde die Benutzerfreundlichkeit durch grundlegende Neuerungen verbessert, die Funktionalität praxisbezogen deutlich erweitert und gänzlich fehlende oder lückenhafte Dokumentationen sowohl in Hinsicht auf die internen als auch externen Strukturen erstellt. Außerdem konnte die Interoperabilität zu anderen Programmen realisiert werden.

Eine Neuerung in Hinblick auf die Benutzerfreundlichkeit ist im Allgemeinen das Bestreben, grafische Oberflächen für alle neuen Funktionen zur Verfügung zu stellen und über eine spezielle Darstellung auf die Funktionen an sich aufmerksam zu machen. Für bereits vorhandene Funktionen, die nur textbasiert verwendet werden konnten, wurden in dieser Arbeit grafische Benutzerschnittstellen implementiert und die Verwendung somit vereinfacht.

Im Rahmen der Funktionalitätserweiterung wurden die *Scripting*-Möglichkeiten des Systems auf verschiedene Weisen erweitert. Dies geschah zum einen, indem weitere Funktionen zur bereits vorhandenen Skriptbibliothek hinzugefügt und die Stapelverarbeitung von Skripten realisiert wurde. Zum anderen wurden die bestehenden *Scripting*-Schnittstellen dem Benutzer zugänglicher gemacht und eine weitere hinzugefügt, die das direkte Ausführen von Skripten ermöglicht. Außerdem wurde die einst primäre grafische Schnittstelle des EMU-Systems komplett neu implementiert, um eine verbesserte Signaldarstellung und Signalbearbeitung zu gewährleisten und die Etikettierung selbst intuitiver und anderen Programmen ähnlicher zu gestalten.

Ebenfalls funktional aber auf Systemebene wurden die Organisation und der Umgang mit den unterschiedlichen Datendefinitionen grundlegend verändert. Ausgehend von einem vom System fest vorgegebenen Speicherort für Datendefinitionen wurde eine Anwendung entwickelt und umgesetzt, die es gestattet, Datendefinitionen an benutzerdefinierten Speicherorten zu verwalten. Diese Entwicklung erleichtert das Archivieren von Datenbanken sowie deren Transfer, sodass Datenbanken auch online verfügbar gemacht und leicht installiert werden können.

Bei der Implementierung neuer Funktionalität wurde jeweils das Bestreben nach Benutzerfreundlichkeit, also die grafische Aufbereitung und Dokumentation, berücksichtigt. Während die Dokumentation der Anwendung selbst eher benutzerorientiert multimedial umgesetzt wurde, wurden für Systeminterna schematische Darstellungen erzeugt. Sie wurden im Rahmen dieser Dissertation als Grundlage für die funktionalen Weiterentwicklungen benötigt und dienen auch in Zukunft Programmierern für die Weiterentwicklung als Orientierung.

Die Erweiterung der Interoperabilität wurde durch eine Funktionalität zum Datenbankenimport und -export über indirekte und direkte Schnittstellen zu den Programmen Praat, WaveSurfer sowie Articulate Assistant realisiert.

Neben den praxisorientierten Erweiterungen der Funktionalität, Benutzerfreundlichkeit und Interoperabilität sowie der Systemdokumentation wurde im Rahmen dieser Dissertation das *Kiel Corpus* in eine EMU-Sprachdatenbank überführt. In dieser Form ist es nunmehr in einer hierarchischen Etikettierungsstruktur aufgrund der Weiterentwicklungen des EMU-Systems sehr gut darstellbar, benutzerfreundlich abfragbar und damit einhergehend analysierbar. Wiederum aufgrund der Weiterentwicklungen besteht die Möglichkeit, es in dieser Form Forschern online zur Verfügung zu stellen und es leicht in EMU-Systemen zu installieren.

Weiterhin wurde in dieser Dissertation auf Grundlage der Schemata aus der Dokumentation diskutiert, wie das System in den nächsten Entwicklungsstufen effizienter gestaltet werden könnte. Vorgeschlagen wird die Überführung des EMU-Datenmodells in ein relationales. Bei geeigneter Modellierung wird nachweislich die Abfrage auf große Datenkorpora zeitlich effizienter und die Abfragemöglichkeiten werden erweitert.

Die Weiterentwicklung des EMU Speech Database Systems, die im Rahmen dieser Dissertation unternommen wurde, hat die Software für die Lehre und Forschung einsetzbar gemacht. EMU kann damit für sehr viel mehr Anwender einen wichtigen Arbeitsschritt innerhalb des eigenen Arbeitsprozesses[2] zur Lösung von Forschungsfragen darstellen.

[2] *workflow*

1. Einleitung

Die zeitgenössische Phonetik bezieht sich im Allgemeinen auf die Produktion, Übertragung und Perzeption, damit einhergehend auf die kognitive Verarbeitung von Sprache. Gegenstand der artikulatorischen Phonetik (Produktion) ist die Aktion und Koordination der Sprechorgane in der Laut- und Lautfolgenproduktion sowie die Auswirkungen verschiedener Faktoren auf diese Prozesse. Hierfür werden artikulatorische Messdaten erhoben, u. a. durch MRI[1] (Ramanarayanan, Bresch, Byrd, Goldstein und Narayanan, 2009), EPG [2] (Ambrazaitis und John, 2004) oder EMA[3] (Bombien, Mooshammer, Hoole, Rathcke und Kühnert, 2007). Ergebnisse aus der Artikulationsforschung können u. a. in der Sprechtherapie und in der automatischen Spracherkennung eingesetzt werden (Heracleous, Badin, Bailly und Hagita, 2011).

Die Übertragung bezieht sich auf das Schallsignal, das durch die Artikulation produziert wird. Es werden akustische Korrelate, d. h. die Abbildung oder Wirkung der Artikulation auf das Schallsignal wie in Ananthakrishnan und Engwall (2011) gesucht. Ein einfaches Beispiel sind Pausen, die leicht durch nullwertige Amplituden im Schallsignal detektierbar sind. In Gordon und Applebaum (2010) werden die akustischen Korrelate zur Betonung in türkischen Dialekten analysiert und mit erhöhter Grundfrequenz, Dauer und Intensität über der betonten Silbe gefunden. Gordon und Ladefoged (2001) beschreiben, wie sich verschiedene Stimmqualitäten wie z. B. Knarrstimme im Signal ausprägen. Diese Erkenntnisse können für die automatische Bestimmung verschiedener artikulatorischer Phänomene verwendet werden. So wurde in John und Harrington (2007) über eine Analyse des Kiel Corpus (Kohler, Pätzold und Simpson, 1995) anhand des Parameters Spektralneigung nachgewiesen, dass sich in der Produktion reduzierter phonologisch stimmhafter und stimmloser Plosive die zugrundeliegende Stimmhaftigkeit in verschiedenen zeitlichen Koordinationen von Knarrstimme widerspiegelt. Auch für die automatische Spracherkennung ist die Erforschung der Korrelate relevant, wie in Meyer, Brand und Kollmeier (2011)

[1] real-time **m**agnetic **r**esonance **i**maging
[2] **E**lektro**p**alato**g**raphie
[3] **E**lectro **M**agnetic **A**rticulometry

auf segmenteller Ebene der Phone oder in den Methoden zur Emotionserkennung (Batliner, Schuller, Seppi, Steidl, Devillers, Vidrascu, Vogt, Aharonson und Amir, 2011).

In der perzeptiven Phonetik werden die in der Akustik gefundenen Korrelate verschiedener Phänomene in der menschlichen auditiven Perzeption getestet. So können akustisch nachweisbare Unterschiede beim Hörer durch die physiologischen Gegebenheiten unterhalb der Wahrnehmungsgrenze liegen (Sanders, Chang, Hiss, Uchanski und Hullar, 2011) oder aufgrund des individuellen sprachabhängigen Phonemsystems von unterschiedlichen Hörern unterschiedlich wahrgenommen werden (John, 2004). Generell müssen aufgrund der quantalen Natur der Sprache Veränderungen in der Produktion nicht zwingend zu Veränderungen in der Akustik und auch nicht in der Perzeption führen (Stevens, 1989).

Speziell in der Phonetik werden zur Beantwortung von Forschungsfragen oder als Forschungsziel selbst in der Produktion sowie in der Akustik aber auch in der Linguistik Sprachdaten gesammelt. Diese Sammlung kann zufällig auftretendes oder von der Forschungsfrage abhängiges kontrolliertes und somit sehr unterschiedliches Sprachmaterial sein. Derartige Sammlungen werden als Sprachkorpora oder Sprachdatenbanken mithilfe von Sprachverarbeitungsprogrammen aufbereitet und analysiert. Sprachverarbeitungsprogramme stellen ein in der Literatur bisher wenig erwähntes, aber sehr wichtiges Standbein für die Erforschung phonetischer und auch linguistischer Phänomene generell dar. Es gibt verschiedene Umsetzungen von Datensammlungen und auch Programmen zur Verarbeitung.

In den Sprachwissenschaften wird der Begriff Sprachdatenbank für fast alles gebraucht, das eine Sammlung sprachlichen Materials enthält. Dieser Umstand ist aus der reinen Wortbedeutung der Konstituenten des Wortes nachvollziehbar. Aus technischer Sicht hat eine Sprachdatenbank jedoch einige grundlegende Eigenschaften mehr.

Beide Wörter <Sprachdatenbank> und < Sprachdatenbanksystem>, als Sprachverarbeitungsprogramm, stellen sprachwissenschaftlich betrachtet ein Kompositum aus mehreren Konstituenten dar.

Sprachdatenbank Der Kopf des Kompositums <Sprachdatenbank>, <bank> (Anhäufung) wird durch <Sprachdaten> semantisch näher bestimmt, wobei der Kopf <daten> wiederum durch <Sprach> näher bestimmt wird. Eine Sprachdatenbank ist somit eine Anhäufung von Daten (Angaben), die aus Sprache gewonnen worden

ist. <Sprach> ist eine Wortform des Lexems SPRACHE, das bei jedem erwachsenen Menschen intuitiv mit einem Bedeutungskonzept verbunden ist. Dennoch gibt es unterschiedliche Interpretationen zu diesem Wort. Für Pelz (2000:4) hat das Wort <Sprache> grundlegend zwei unterschiedliche Bedeutungen: „die Sprechfähigkeit des Menschen im Allgemeinen, die ihn von anderen Lebewesen unterscheidet“ oder „die Nationalsprache“. Somit kommt ein weiterer Aspekt von Sprache, nämlich das Sprechen hinzu.

Sprache Um diese verschiedenen Bedeutungen von Sprache wissenschaftlich handhabbar zu machen, schlugen de Saussure und Chomsky zu ihrer Zeit unterschiedliche Termini vor. Während in de Saussure (2001), einer von zahlreichen übersetzten Auflagen des Originalwerkes von de Saussure (1916), die Begriffe ‹ langue › und ‹ parole › dem französischen ‹ langage › gegenüber gestellt werden, beschränkt sich Chomsky (1965) auf ‘competence’ und ‘performance’. Während ‘competence’ nach Chomsky (1965:3ff.) “the speaker-hearer’s knowledge of his language” ist, sei ‘performance’ “the actual use of language in concrete situations”. Der individuelle und situative Gebrauch von Sprache entspricht zum Teil der ‹ parole › im Sinne de Saussures. Zusätzlich bezeichnet der Terminus die durch den Sprachverwendungsakt produzierte Äußerung. Der ‹ langue › Begriff bei de Saussure lässt sich jedoch nicht mit Chomskys ‘competence’, die eine individuelle Komponente trägt, gleichsetzen, denn ‹ langue › ist eine überindividuelle, soziale Erscheinung, die einer Gesellschaft gemeinsam ist. Das Vermögen des Menschen, als Kommunikationsmittel Sprache zu verwenden, drückt in diesem Paradigma der Terminus ‹ langage › aus. Allen Ansätzen, die versuchen das Konzept Sprache in Worte zu fassen, ist gemeinsam, dass sie eine Fähigkeit des Menschen ist, die individuell ist, weiterhin dass sie als Kommunikationsmittel eingesetzt wird, dem ein in der Gemeinschaft gemeinsamer Code[4] zugrunde liegt, welcher u. a. durch Schrift, Gesten und Sprechen wiederum individuell umgesetzt wird.

Sprachdatenbanken enthalten Daten von ‘performance’ bzw. ‹ parole › im Sinne von Äußerungen, die ein Sprecher auf Grundlage seiner ‹ langue › mit seiner ‘competence’ erzeugt hat. Anhand einer solchen Sammlung von mehreren Sprechern einer ‹ langue › können Analysen vorgenommen werden, die Aufschluss sowohl über die ‘performance’ selbst als auch über die ‘competence’ geben können. Diese Ergebnisse können über mehrere Sprecher verallgemeinert werden, um die Eigenschaften der

[4] Code meint eine Sprache wie Deutsch, dessen Semantik, Syntax, Morphologie, Phonologie, Phonetik usw. in einem gewissen Rahmen festgelegt ist.

‹ langue › zu bestimmen. Die Verallgemeinerung der Ergebnisse zu Untersuchungen mehrerer ‹ langues › können wiederum Aufschluss über die ‹ langage ›, die Fähigkeit des Menschen, geben.

Sprachdatenbanksystem Die Konstituente <system> im Wort <Sprachdatenbanksystem> kann mit mehreren Bedeutungen assoziiert sein. Es bezeichnet das Datenbankmanagementsystem (DBMS), das die Datenbanken verwaltet, oder es bezeichnet die Software, in der ein solches DBMS enthalten ist. Simpson (1998, S. 4) sieht den Nutzen von Sprachdatenbanken bzw. Korpora hauptsächlich „in der Forschung und Entwicklung sprachtechnologischer Anwendungen“ sowie in der Grundlagenforschung. Zu den sprachtechnologischen Anwendungen gehören die Spracherkennung, Sprachsynthese, Sprecherverifikation sowie Sprecher- und Sprachidentifikation.

Moderne Analysemethoden in der linguistischen und phonetischen Forschung, denen Wahrscheinlichkeitsmodelle zugrunde liegen, benötigen eine sehr umfangreiche Datenbasis, die aus Rohdaten aufbereitetet ist. Eine Überführung von Rohdaten in eine Datenbank eines angemessenem Datenbanksystems eröffnet die Möglichkeit, viele der im weiteren Verlauf benötigten Arbeitsschritte zu automatisieren, was den erheblichen Aufwand bei großen Datenmengen verringert. An diesem Punkt unterscheiden sich Korpora von Datenbanken.

Eine Datenbank ist in einem Datenbanksystem eingebunden. Neben der Datenbasis enthält ein solches System auch ein Datenbankmanagementsystem, welches Methoden für die Speicherung, Abfrage und Verarbeitung zur Verfügung stellt, die speziell auf die Datenbankstruktur abgestimmt sind. Ein Sprachkorpus stellt in diesem Rahmen nur eine nicht strukturierte Datenbasis dar, ohne die Möglichkeit der Anwendung speziell angepasster Methoden.

Die Datenbasis von Korpora oder Datenbanken in der Analyse gesprochener Sprache enthält in der Regel Daten der ‘performance’ in Form von diskretisierten Signalen wie Amplitudenwerten der Luftdruckschwankungen, EMA, EPG oder Daten in Form von abgeleiteten Signalen, wie dem Spektrum, der Grundfrequenz usw. In diesen Signalen sind verschiedene sprachliche Einheiten durch Etiketten markiert.

Etikettierung von Sprachdaten Nach Schultze-Berndt (2006) sollte eine linguistische Etikettierung, die auch für die Phonetik relevant ist, u. a. zunächst auf segmenteller Ebene die Sätze und Wörter, die Wortgrenzen, die Morpheme als kleinste

bedeutungstragende Einheit, die Phoneme als kleinste bedeutungsunterscheidende Einheit, die phonologischen Wortgrenzen und eine lautliche, also phonetische Transkription enthalten.[5] Letzteres kann je nachdem, ob eine umfangreiche phonologische Analyse der Sprache vorliegt, sehr eng oder weit sein, wobei mit einer engen Transkription auch allophonische Varianten als Information in die Sammlung einfließen. Morpheme, Sätze und Wörter können in Graphemen (Buchstaben) und der sprachabhängigen Orthographie etikettiert sein. Abbildung 1.1 zeigt ein etikettiertes Sprachsignal als Beispiel.

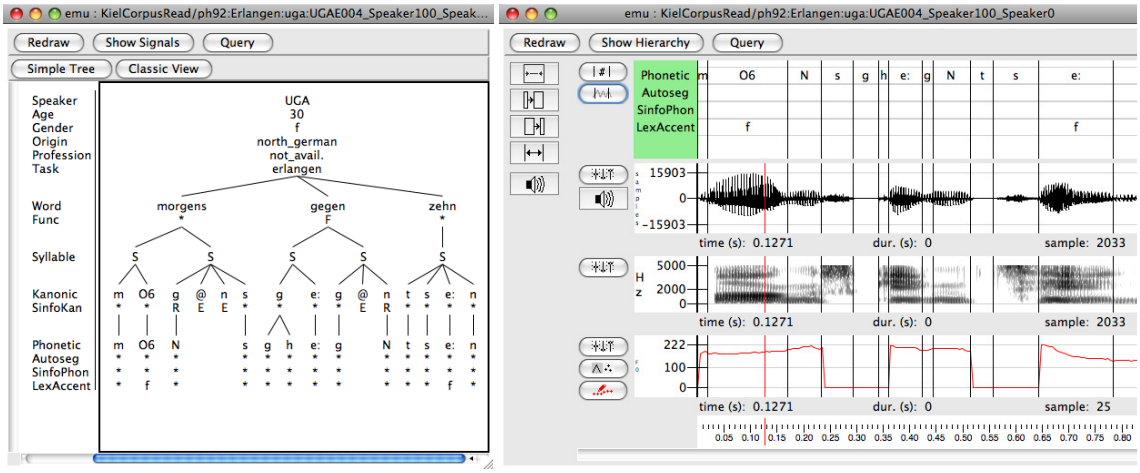


Abbildung 1.1.: Etikettiertes Sprachsignal im EMU Speech Database System.

Zu einer Etikettierung können weiterhin prosodische Merkmale gehören wie Tonhöhenverläufe, Akzente, Pausen, Sprechtempo und -rhythmus, die mit den segmentellen Einheiten (Wörtern, Phonen, etc.) verbunden sein sollten. Für die Markierung hat sich eine Vielzahl von Konventionen entwickelt. Ihnen ist gemeinsam, dass sie in der Regel verschiedene orthographische Zeichen mit gewissen Konventionen, wie beispielsweise Großschreibung für akzentuierte Silben, verwenden (Schultze-Berndt 2006). Wie auch die phonetische und phonemische Transkription, kann die Etikettierung von Prosodie phonologisch motiviert sein. Intonationsmodelle wie ToBI (Tone and Break Indizes) oder das KIM (Kieler Intonationsmodell) modellieren Phänomene wie Tonhöhenabfall oder -anstieg, die die Bedeutung des Gesagten beeinflussen und stellen Konventionen zur Zeichenverwendung für die Etikettierung dieser Phänomene auf (Pierrehumbert und Club, 1980; Beckman und Hirschberg,

[5]Die exakte Etikettierung von Sätzen, Wörtern, Morphemen und Phonemen kann nur vorgenommen werden, wenn die Sprache auf diesen Ebenen bereits untersucht ist.

1994; Peters, 1999; Kohler, 1995). Signale und Etikettierungen werden für den Aufbau der Datenbasis und der Analyse in einem System zusammen verarbeitet.

Da die Datenbasis in der Regel rechnergestützt verarbeitet wird und auch auf andere Systeme übertragbar sein soll, muss sowohl bei den Signalen als auch bei der Etikettierung auf international kompatible Formate bzw. Kodierungen der Daten geachtet werden. Zur Zeichensetzung in der Etikettierung bietet sich hierfür besonders für die phonetische Transkription und auch für die Phoneme das *International Phonetic Alphabet* (IPA, 1999) mit einer Unicode-Codierung (Allen und Consortium, 2007) an oder eine sich bereits durchgesetzte und verbreitete "computer readable" (Wells, 1997, S. 1) und „tastaturfreundliche ASCII-Kodierung[6] des IPA" (Ebert und Ebert, 2010, S. 176), das SAMPA[7] (Wells 1997) an. Gleiches gilt auch für die Transliteration. Für Signale sei hier das WAV-Format (IBM Corp. and Microsoft Corp., 1991; Microsoft, 1994; Microsoft, 2001) genannt.

Einsatz von Sprachdatenbanken Der Zusammenhang von Daten, Analysen und Ergebnissen ist in der Wissenschaft unumgänglich. Auch in der linguistischen und phonetischen Forschung stehen Ergebnissen Analysen von Daten voran. Die Form des verwendeten Materials ist zwischen den unterschiedlichen Wissenschaften, aber auch innerhalb einer Wissenschaft sehr verschieden. Während linguistische Analysen anhand geschriebener, sogar in Stein gemeißelter Texte durchgeführt werden können, kann der Phonetiker im 21. Jhdt. auf digital vorliegende Sprachaufnahmen zurückgreifen. Die Wahl und der Zeitpunkt des Einsatzes eines Datenbanksystems sollte nach Umsetzbarkeit und Komplexität abgewogen werden. Ein Sprachverarbeitungsprogramm, das Sprachaufnahmen im Sinne der 'performance' bzw. ‹ parole › voraussetzt, kann keine in Stein gemeißelten Texte oder Bilder dieser verarbeiten. Der Aufbau einer Datenbank, die nur einen einzigen 10 Sekunden langen Ausschnitt von ‹ parole › enthalten würde, ist nicht angemessen. Um die Umsetzbarkeit zu gewährleisten, sollte ein Datenbanksystem für den Benutzer und die Forschungsfragen angemessen entwickelt sein.

Unabhängig von der Komplexität der Daten benutzt nicht jeder Forscher in der Phonetik Sprachdatenbanksysteme. Trotzdem verwendet jeder von ihnen zur Ana-

[6] Das ASCII (*American Standard Code for Information Interchange*) Zeicheninventar (ISO/IEC 646, 1991) deckt überwiegend nur Zeichen ab, die auf englischen bzw. amerikanischen Computertastaturen zu finden sind. Daher sind sie international in rechnergestützten Systemen einheitlich vorhanden und verarbeitbar.

[7] *Speech Assessment Methods Phonetic Alphabet*

lyse von Sprachdaten rechnergestützte Verfahren. Derartige Verfahren werden von Programmen bereitgestellt. Praat (Boersma, 2002), WaveSurfer (Sjölander und Beskow, 2010) und das EMU Speech Database System (Bombien, Cassidy, Harrington, John und Palethorpe, 2006) sind drei freie Programme, die in der phonetischen Forschung u. a. verwendet werden. Wie sich in verschiedenen Workshops[8] zu Programmen in den letzten Jahren herausgestellt hat, stellt Praat die am häufigsten benutzte Software dar[9]. Das EMU-System unterscheidet sich grundlegend von den anderen Programmen, indem die Struktur der Datenbasis definiert sein muss und die Daten auf Grundlage dieser Definition zentral verwaltet werden.

Als Motivation für die Weiterentwicklung gerade des letztgenannten Systems, wird im folgenden Kapitel 1.1 zunächst aufgezeigt, dass anders als im EMU-System, die Verarbeitung in heutigen Sprachverarbeitungsprogrammen vorwiegend dateibasiert ohne systemeigene Organisation erfolgt. In Kapitel 1.2 wird daraufhin aufzeigt, welche Folgen die Unterschiede zwischen zentraler Datenorganisation und fehlender Datenorganisation haben können.

Eine strukturierte Datenbasis und eine zentrale Verwaltung von Daten sind Eigenschaften eines Datenbanksystems. Auch das EMU ist als Sprachdatenbanksystem im Beinamen tituliert. Datenbanksysteme, wie sie heutzutage verstanden werden, weißen jedoch weitere Eigenschaften auf, die das EMU-System ohne Zweifel bisher nicht erfüllt. Unter diesen Bedingungen kann aus streng informationstechnologischer Betrachtung heraus, die EMU-Software nicht als Datenbanksystem anerkannt werden[10]. Obgleich dieser Einschätzung lässt sich der zugrundeliegende Ansatz im EMU-System sehr gut mit Datenbanksystemkonzepten erklären. Aus diesem Grund wird der Unterschied zwischen einem Datenbanksystem aus streng informationstechnologischer Sicht und dem datenbanksystemähnlichen Ansatz des EMU-Systems im Rahmen dieser Arbeit nicht diskutiert. Das EMU-System wird entgegen der streng informationstechnologischen Sicht als Datenbanksystem behandelt.

EMU als Sprachdatenbanksystem hat eine lange Entwicklungsgeschichte hinter sich. Das EMU-System basiert auf dem MU+ (Harrington, Cassidy, Fletcher und McVeigh, 1993) System, das für die individuelle Verarbeitung des ANDOSL Kor-

[8] "Developing standards for phonological corpora" im Juli 2009 an der Universität Augsburg und "New Tools and Methods for Very-Large-Scale Phonetics Research" an der University of Pennsylvania im Januar 2011

[9] Diese Einschätzung basiert nicht auf repräsentativen statistischen Datenerhebungen und ist damit als vage zu kommentieren.

[10] persönliche Kommunikation mit PD Christoph Draxler, LMU München

pus (Millar, Vonwiller, Harrington und Dermody, 1994) am SHLRC der Macquarie University in Sydney, Australien entworfen und implementiert wurde. MU+ selbst basiert wiederum auf einer an der Edinburgh University entwickelten Erweiterung des statistischen Pakets S, dem in Watson (1989) vorgestellten Software System APS (Acoustic Phonetics in S). Seit 2002 wird das australische EMU-System unter gleichem Namen in Deutschland zunächst in Kiel und später in München unter anderem im Rahmen dieser Dissertation weiterentwickelt. Bis heute ist das EMU-System mit vielen unterschiedlichen Forschungsansätzen, und -methoden konfrontiert worden, die in der Entwicklung berücksichtigt werden mussten. Somit konnte sich das System als ein für sehr viele Forschungsfragen und -methoden einsetzbares Programm in der phonetischen Forschung neben anderen Programmen mit anderen Ansätzen etablieren.

> "EMU is best used for: automatic annotation at higher levels, database queries, and integration with statistical software. In addition, EMU can be used to handle cases of many-to-many mapping across levels that cannot be handled by other software." (Williams, 2008, S. 174)

Im Rahmen dieser Dissertation wurde in der Weiterentwicklung des Systems die Verbesserung der Benutzerfreundlichkeit, die Erweiterung der Funktionalität und die Interoperabilität in Angriff genommen. Weiterhin wurde ein sehr umfangreiches Sprachkorpus mit linearen Etikettierungen für die Verwendung als EMU-Sprachdatenbank mit der EMU-typischen hierarchischen Struktur der Etikettierungen aufbereitet.

Ausreichend detaillierte Dokumentationen sowohl interner implementierungsnaher als auch externer benutzernaher Strukturen als Grundlage zum Erreichen der genannten Ziele, standen vor dieser Dissertation nicht zur Verfügung. Somit war es unumgänglich, diese Beschreibungen als zusätzliches Ziel aufzunehmen.

Bereits vor der Weiterentwicklung war das EMU-System eine funktional mächtige Software. Die Erfahrungen im eigenem Studium sowie später in der Dozententätigkeit in der Hochschullehre zeigten jedoch, dass potentielle Benutzer, Studierende und Forscher der Phonetik und Linguistik erhebliche Schwierigkeiten mit der Verwendung der Software hatten. Auch Williams (2008) bestätigt diesen subjektiven Eindruck in ihrer objektiven Kritik[11] des EMU-Systems, wie aus dem folgenden Zitat hervorgeht.

[11] *review*

> "A steep learning curve is involved on initial exposure to EMU. It is not possible to use EMU without some kind of database template file, however simple, and so this feature must be mastered before continuing. By comparison, Praat can be used at a basic level very soon after installation, with a minimum of learning of the software. " (Williams, 2008, S. 173)

Das Abflachen dieser Lernkurve steht als allgemeines Ziel im Fokus dieser Dissertation. Hierfür soll die Arbeit einen Einblick in das EMU-Sprachdatenbanksystem im derzeitigen Entwicklungsstand (Sommer 2011) geben. Dabei werden die eruierten zugrundeliegenden Strukturen aufgezeigt, die Vorteile des datenbankähnlichen Ansatzes für den Benutzer verdeutlicht sowie die Weiterentwicklungen des Systems und die damit einhergehenden Vorarbeiten in den oben genannten Punkten dargelegt. Kapitel 1.3 stellt die Weiterentwicklungen des EMU-Systems zunächst anhand der Implementierungsstruktur in kompakter Form dar. Schließlich gibt Kapitel 1.4 einen Überblick über die Inhalte dieser Arbeit.

1.1. Datenbankfunktionalität von Sprachverarbeitungsprogrammen

Nach Elmasri und Navathe (2009) und Simpson (1998) werden Datenbanken u. a. im Bereich der Bildung eingesetzt. Im Rahmen eines Studiums der Linguistik oder Phonetik, in denen Sprachverarbeitung ein Teil der Lehre ist, wird mit Sprachdaten gearbeitet. Sprachdaten sind alphanumerisch (Buchstaben, Ziffern, Sonderzeichen) gespeicherte Informationen. Die Voraussetzungen für die Verwendung eines Datenbanksystems sind somit erfüllt.

Für die Aufgabe der Sprachverarbeitung wurde ganz unterschiedliche Software entwickelt, die im Folgenden Sprachverarbeitungsprogramme genannt werden. An dieser Stelle soll geprüft werden, ob die Sprachverarbeitungsprogramme datenbankbasiert arbeiten. Mit dem folgenden Zitat lässt sich bereits im Voraus die Hypothese aufstellen, dass eine Datenbankfunktionalität bei den meisten vorhandenen Programmen nicht bereitgestellt wird.

> „Leider greifen die meisten Etikettierungseditoren aber nicht auf Datenbanksysteme, sondern direkt auf Dateien im Dateisystem [...] zu.“ (Draxler, 2006, S. 217)

Die Eigenschaften eines Datenbanksystems werden in Kapitel 2.1 diskutiert und lassen sich in folgenden Punkten zusammenfassen. Es ist eine Software, die mithilfe eines Datenbankmanagementsystems (DBMS) mehrere Datenbanken zentral verwalten kann, wobei das DBMS die Benutzer und Programme von der physischen Datenspeicherung abschirmt. Eine Datenbank in diesem System besitzt eine mithilfe des Datenbankschemas strukturierte Datenbasis. Treffen diese Eigenschaften für ein Programm zu, wird es im Folgenden als datenbankbasiert arbeitend eingestuft.

In der Wissenschaft interessant sind in diesem Rahmen zunächst nur freie, nicht kommerzielle Programme. Wissenschaft sollte immer mit der Weitergabe des Wissens in der Lehre einhergehen. Um auch Studierenden der Wissenschaften die Verwendung zu ermöglichen, sollten kommerzielle Programme in Wissenschaft und Lehre weitgehend vermieden werden. In der heutigen Zeit stehen *open source* Software (Lakhani und von Hippel, 2003) oder kostenfreie Programme für fast alle Bereiche zur Verfügung, so auch mit EMU, Praat, WaveSurfer und Elan im Bereich der Phonetik und letzteres im Bereich der Linguistik (Bombien, Cassidy, Harrington, John und Pal-

ethorpe, 2006; Boersma, 2002; Sjölander und Beskow, 2000; Wittenburg, Brugman, Russel, Klassmann und Sloetjes, 2006).

Am Beispiel des am weitesten verbreiteten Sprachverarbeitungsprogramms Praat – *Doing Phonetics by Computer* (Boersma, 2002) kann belegt werden, dass ein solches System dateibasiert arbeitet. Ein kurzer Einblick in das Arbeiten mit Praat wird im Folgenden gegeben. Eine visualisierte Darstellung ist in den Videohilfen (John, 2010m) zu Kapitel 2.4 von Harrington (2010a) zu finden.

Um eine Signaldatei phonetisch zu etikettieren, d. h. Sprachsignalteile mit Etiketten zu versehen, muss zunächst eine Signaldatei über den Dateibrowser aus der lokalen Verzeichnisstruktur ausgewählt und in Praat geöffnet werden. Es entsteht innerhalb der Applikation ein Sound-Objekt, das im weiteren Verlauf anstelle der Signaldatei verwendet wird. Liegt für das Signal noch keine Etikettierungsdatei (*TextGrid*-Datei) vor, so kann in Praat das Sound-Objekt ausgewählt werden, um es im TextGrid-Editor zu öffnen. Praat sieht für die Etikettierung von Sprachsignalen eine beliebige Anzahl an Ebenen (Tiers) vor, auf denen Zeitintervalle des Signals mit Etiketten versehen werden können (Intervall-Tiers), aber auch Ebenen, auf denen nur einzelne Zeitpunkte mit Etikett versehen werden können (Point Tiers). Da für ein unetikettiertes Signal noch nicht festgelegt ist, auf welchen Ebenen etikettiert werden soll, können diese, bevor sich der TextGrid-Editor zum Etikettieren öffnet, definiert werden. An diesem Punkt angekommen, können Etiketten auf den Ebenen hinzugefügt, aber auch zusätzliche Ebenen eingefügt und gelöscht werden. Die Etikettierung auf den Ebenen wird während der Laufzeit des Programms in TextGrid-Objekten gespeichert. Für die permanente Speicherung muss dieses TextGrid-Objekt als Datei gespeichert werden, hierfür wird wieder über den Dateibrowser ein Verzeichnis zur Speicherung ausgewählt und der Datei ein Dateiname zugewiesen. Wurden zu speichernde Änderungen am Signal vorgenommen, müssen diese auf dem selben Weg gespeichert werden. Für eine Weiterverarbeitung müssen das Sound-Objekt und das TextGrid-Objekt, falls durch das Schließen des Programms nicht mehr im Programm vorhanden, wieder geladen werden, wofür die Dateien wieder über den Dateibrowser geöffnet werden müssen. Sie können nach der Bearbeitung wieder gespeichert werden. Dieses Vorgehen kann sich beliebig oft wiederholen. Welche Folgen ein solches Vorgehen haben kann, wird in 1.2 erläutert.

Eine fehlende zentrale Verwaltung der Informationen, wie der Etikettierungen und der Signale, ist zu erkennen, da Etikettierungen bereits in den elementarsten Eigenschaften, wie in der Benennung der Ebenen für jedes Sprachmaterial unterschied-

lich definiert sein können. Die Zusammengehörigkeit ist nicht zentral gesteuert, so kann eine Etikettierung und die Signaldateien, die der Etikettierung zugrunde liegen, an unterschiedlichen Orten und unter sehr unterschiedlichen Namen abgelegt werden. Dabei gibt es keine Instanz, die die Zusammengehörigkeit der Daten belegen kann. Selbst wenn die Zusammengehörigkeit von Etikettierungen und Signalen durch den Benutzer durch gleiche Namensgebung provoziert wird, gibt es keine Instanz, die aufzeigt, welches Material ein gemeinsames Korpus bildet. Hierfür gibt es aus Benutzersicht Strategien, beispielsweise das Speichern der korpusbildenden Daten innhalb eines Verzeichnisses. Derartige Strategien werden jedoch wiederum nicht auf Einhaltung überprüft, da sie nicht zentral verwaltet werden. Veränderungen am Etikettierungsschemas, d. h. an den Etikettierungskonvention bezüglich der verwendeten Ebenen, können jederzeit vorgenommen werden. Diese Schemata sind jedoch nur für eine Äußerung (die in separaten Dateien vorliegen) gültig. Auf diese Weise können Etikettierungen zu verschiedenen Sprachsignalen, die manuell bzw. konzeptuell zu einem Korpus zusammengefasst wurden, unterschiedliche Informationen enthalten. Diese Tatsache erschwert Korpusanfragen, da nicht zentral definiert ist, welche Informationen in welcher Form vorhanden sind.

Praat bietet *Scripting*-Schnittstellen an, womit zunächst durch kleine Skripte Datenmanagement betrieben werden kann. D. h. jeder Benutzer kann sich seine eigenen Funktionen schreiben, die Dateien aus der Datenbasis wählen, sie in Praat öffnen und die gewünschten Operationen auch gleichzeitig bzw. nacheinander für mehrere Dateien durchführen. Dieses Vorgehen ist beim Aufbau von Korpora üblich. Es werden Sprachdaten gesammelt und dann Programme implementiert, die auf dieses Korpus Operationen ausführen können. Ob Praat-Skripte oder derartige Korpusprogramme, sie sind abhängig von der Datenbasis, so dass sie nur spezifisch eingesetzt werden können. Skripte und Programme können selbstverständlich in Hinblick auf eine Allgemeingültigkeit auf unterschiedliche Daten implementiert werden. Diese Allgemeingültigkeit setzt jedoch an irgendeiner Stelle Konventionen und Konfigurationen voraus. Wird dieses Verfahren perfektioniert und die implementierten Funktionen arbeiten auf einem beliebigen Set an Daten, das einer gewissen Grundstruktur folgt, die wiederum durch Konfigurationen spezialisiert werden kann, so kann man in diesem Fall auch von einem System sprechen, das Daten zentral verwaltet. Dann arbeiten Funktionen des Skripts oder Programms, wie auch für Datenbanksysteme üblich, auf strukturierten definierten Daten. Die Eigenschaft von Datenbanksystemen, den Benutzer von den eigentlichen physischen Daten abzuschirmen, kann auch in Praat-Skripten oder Korpusprogrammen erfolgen, indem gezielt Skripte verfasst

werden, die nach Aufruf eine gewünschte Datei öffnen, ohne dass der Nutzer selbst einen Dateibrowser verwenden muss.

Wie argumentiert wurde, kann ein jeder Anwender eines Sprachverarbeitungsprogramms die Vorteile einer zentralen Verwaltung und einer strukturierten Datenbasis nutzen, indem er ein Datenbanksystem zur Verarbeitung wählt oder in anderen Programmen Skripte schreibt und verwendet. Das naheliegendste Skript hilft dem Nutzer zu allererst, den Vorgang des Heraussuchens einer Datei im Dateibrowser zu umgehen. Das ist nur ein Anfang auf dem Weg zum Datenmanagementsystem und mit einer strukturierten Datenbasis zu einem Datenbankmanagementsystem. Diese Schritte erfordern jedoch Kenntnisse und Erfahrungen in der Programmierung. Nutzer von Sprachverarbeitungsprogrammen sind jedoch die Forscher aus Linguistik und Phonetik und nicht die Programmierer. Daher stellt sich die Frage, und eigentlich stellt sie sich nicht, ob jeder Nutzer sein eigenes Datenmanagementsystem skriptbasiert entwirft oder auf ein bereits vorhandenes und vielseitig verwendbares Sprachdatenbanksystem wie das EMU Speech Database System zurückgreift.

1.2. Dateiorientiertes versus Datenbankorientiertes Arbeiten

> "There are many benefits gained in the conversion from several files [...] to a database [...]. One such benefit results from the significant improvement in performance that accrues from using the database [...]." (Bachman, 1973, S. 655)

Neben der erhöhten Leistungsfähigkeit, die die Verwendung einer Datenbank mit sich bringt, wie von Bachman (1973) im Zitat erwähnt, bietet die zentrale Verwaltung von Daten eine höhere Datenkonsistenz. In einer dateiorientierten Datenverarbeitung ohne zentrale Verwaltung werden wie im vorangegangenen Kapitel beschrieben wurde, aus dem Dateispeicher Eingabedateien in ein Programm eingelesen. Dort werden die Daten bearbeitet. Das Programm speichert die Daten in Ausgabedateien in den Datenspeicher. Die Ein- und Ausgabedateien werden durch die Programme verwaltet, somit sind die Dateistrukturen an die Datenstrukturen und die Verarbeitungsroutinen der einzelnen Programme angepasst. Es besteht somit eine logische Datenabhängigkeit.

Die Verknüpfung von Datenbeständen ist in einer solchen Datenverarbeitung schwierig, so dass die gleichen Informationen u. U. in mehreren Dateien und auch in unterschiedlichen Datenstrukturen gespeichert sein können. Somit ergibt sich eine mögliche Redundanz in den Datensätzen. Bei dem Zugriff durch verschiedene Programme auf Dateien können Datensätze verloren gehen, indem sie durch eine ungünstige Wahl des Datenspeicherbereichs andere Dateien überschreiben oder die Datenstrukturen so weit ändern, dass andere Programme die Dateien nicht mehr einlesen können. Aus dem alltäglichen Umgang mit Dateien sind derartige Vorfälle bekannt, wie das Speichern einer Datei in ein Verzeichnis, in dem eine Datei gleichen Namens bereits vorhanden war. Unter Umständen wird damit eine aktuellere Version der Datei überschrieben, was zum Datenverlust führt oder der Dateiinhalt war identisch, was eine Redundanz zeigt.

Im Gegensatz dazu steht die datenbankorientierte Datenverarbeitung. Die Verwendung eines Datenbanksystem bietet eine Integration und eine zentrale Verwaltung der Datenbestände. Die Datenstrukturen sind anders als bei der dateiorientierten Verarbeitung unabhängig von einzelnen Programmen, die die Daten verarbeiten sollen, außerdem sind die Daten selbst nach einem festen Schema strukturiert. Damit

verringert sich die Gefahr der Dateninkonsistenz und vereinfacht die Gewährleistung der Datenintegrität. Durch die Integration und die zentrale Verwaltung der Datenbasis funktioniert der Zugriff auf die Daten über vorgegebene Schnittstellen, was den Zugriff vereinfacht und das Organisieren von parallelen Zugriffen auf Daten verbessert. Im besten Fall bietet ein Datenbanksystem eine zentrale Regelung der Zugriffsrechte.

Folgende Schritte werden durchlaufen, wenn Informationen aus der Datenbasis aufgerufen und modifiziert werden: Ein Programm fragt Daten beim Datenbanksystem an. Dieses prüft die Zulässigkeit der Anfrage und gibt ggf. die Daten in geeigneter Form zurück. Die Zulässigkeit ist von den unterschiedlichen Zugriffsrechten beeinflusst. Das Programm kann nun die Daten ändern. Um die Modifikationen zu speichern, muss es eine Anfrage zum Verändern der Daten an das Datenbanksystem weitergeben. Dieses prüft wiederum, ob eine derartige Anfrage zulässig ist. Ist das der Fall, werden die modifizierten Daten auf Datenintegrität geprüft, d. h. ob sie einer im Datenbanksystem definierten Form entsprechen. Ist dieses wiederum gegeben, so werden die Veränderungen an der Datenbasis vorgenommen. Weder beim Einlesen noch beim Zurückgeben der Daten aus dem Datenspeicher kennt das Programm den physischen Speicherbereich der Daten. Im Datenbanksystem besteht wieder, anders als bei der dateibasierten Verarbeitung, keine logische Datenabhängigkeit.

Die erhöhte Leistungsfähigkeit der datenbankorientierten Datenverarbeitung resultiert aus der Anzahl der Dateizugriffe und wie durch die Daten navigiert werden kann. Im Datenbanksystem sind die Speicherbereiche und Datenstrukturen bekannt und der Zugriff kann optimiert werden, während die Datenstruktur von Dateien vom Programm detektiert werden muss und auch der Speicherbereich nicht vom Programm verwaltet wird. Man kann Datenbanken und Dateien nicht gänzlich voneinander trennen, wie auch “The distinction between a file and a database is not clearly established“ (Bachman, 1973, S. 655) betont, denn im Hintergrund auch bei Datenbanksystemen stehen Dateien zur Verfügung, die die Informationen außerhalb der Laufzeit des Systems persistent speichern.

Aber auch die datenbankorientierte Datenverarbeitung ist nicht von Nachteilen freizusprechen. Eine wirklich effiziente Datenbank benötigt eine sehr intensive Planung, einen sehr guten Datenbankentwurf und eine Datenbankwartung, die einen Datenbankadministrator erfordert.

Das EMU-System verfolgt einen datenbanksystemähnlichen Ansatz. Es hat als Datenbankadministratoren nur seine eigenen Nutzer zur Verfügung. Da die Nutzer

vorrangig Phonetiker und keine Informatiker sind, ist der Datenbankentwurfsprozess und die Datenbankwartung entsprechend leichter als in komplexeren Datenbanken größerer Firmen. Dennoch bietet es eine gewisse Integration der Datenbestände und kann somit die Daten zentral verwalten. Dadurch kann EMU eine datenbankbasierte Sprachdatenverarbeitung in der Phonetik leisten und macht die Vorteile einer datenbankorientierten Verarbeitung auch in der phonetischen Forschung möglich.

1.3. Das EMU Speech Database System: Struktur & Wandel

Das EMU Speech Database System wird seit den 90er Jahren des 20. Jhdt. als Software für den Aufbau von Sprachdatenbanken und für quantitative Analysen dieser entwickelt (Cassidy und Harrington, 1996; Cassidy und Harrington, 2001; Harrington, Cassidy, John und Scheffers, 2003; Bombien, Cassidy, Harrington, John und Palethorpe, 2006; John und Bombien, iE). Diesem Aspekt folgend, muss das System zunächst das Definieren von Datenbanken und damit einhergehend das Erzeugen, Modifizieren, Löschen und Abfragen der Datenbasis ermöglichen sowie eine Vielzahl an Routinen zur Analyse, die speziell an die Datenbasis angepasst sind, anbieten.

Für die Erzeugung der Datenbasis muss die Möglichkeit zur Einbindung von Zeitsignalen und zur Etikettierung von Sprachsignalen angeboten werden. In diesem Bereich ist die Entwicklung eines Datenmodells der Etikettierung, das alle Arten von Etikettierungen akzeptiert, ein Ziel vom System. Die Vielfalt an Etikettierungsstrukturen sollte weiterhin datenbankbasiert über die Abfragesprache des Datenbanksystems abfragbar sein. Für die Etikettierung von Sprachsignalen muss deren Darstellung und auch die Darstellung jeglicher Art von Zeitsignalen durch das System gewährleistet sein. Das EMU-System soll weiterhin die Möglichkeit bieten, mit jedem Dateiformat arbeiten zu können, soweit es die Beschränkungen der internen Repräsentation erfüllt, wie Cassidy und Harrington (2001) betonen. Insgesamt soll das System an die Anforderungen seiner Nutzer angepasst und anpassbar sein.

Das System, bestehend aus einer C++ Bibliothek für die grundlegende Datenbankfunktionalität in einer Tcl/Tk Umgebung (Ousterhout, 1994), die die einfache Erweiterung des Systems an neue Anforderungen ermöglichen soll (Cassidy und Harrington 1996), konnte bereits vor der Weiterentwicklung die genannten Anforderungen erfüllen. 2002 wurde der EMU Labeller mit dem integrierten Query Tool und *Scripting*-Möglichkeiten zur automatischen Etikettierung als einzige und daher auch primäre grafische Schnittstelle verwendet. Für den EMU Labeller können Erweiterungen in Tcl/Tk oder auch C/C++ mithilfe von EMU Labeller Modulen eingebunden werden, um die Funktionalität der Applikation zu erweitern. Nach Cassidy und Harrington (2001) sollen auf diesem Weg unterschiedliche Darstellungsmöglichkeiten verschiedener Daten implementiert werden. Als Beispiel wird das EPG-Plug-In genannt, das elektropalatographische Daten als Modell des Gaumenabdrucks an verschiedenen Zeitpunkten darstellt. Für Analyseroutinen, die speziell an Sprachdaten-

banken angepasst sind, stellt EMU eine Schnittstelle zur R Programming Language (R Development Core Team, 2011) zur Verfügung.

Die Anforderungen an das System wurden vom gesamten Entwicklerteam in den letzten Jahren erweitert, sodass von nicht funktionaler Seite aus betrachtet das System plattform- und hardwareunabhängig sein und Anschluss an möglichst viele existierende Systeme für die Sprachanalyse haben soll. Besonders im Rahmen dieser Dissertation wurde die Anforderung an die Benutzbarkeit vorhandener Funktionalität des Systems, die Bereitstellung weiterer nützlicher Funktionen sowie die Anforderung an die Interoperabilität hervorgehoben.

Abbildung 1.2 auf der nächsten Seite zeigt sowohl eine schematische Darstellung der Implementierungsstruktur des Systems im derzeitigen Entwicklungsstand als auch welche Teile im Zuge der Weiterentwicklung im Rahmen dieser Dissertation und durch andere Entwickler hinzugekommen sind bzw. überarbeitet oder neu implementiert wurden. Die Tabellen B.1(a) und (b) ergänzen die Abbildung mit Informationen zu den im Umgang verwendeten Namen der Applikationen[12] und Programmpakete sowie zu den jeweils beteiligten Entwicklern.

Wie aus der Abbildung hervorgeht, hat sich die EMU-Software in den letzten Jahren zu einem modularen System entwickelt. Neben dem EMU-core sind nun das Snack Toolkit (Sjölander, 2004) und `libassp` (Scheffers und Bombien, 2011) im System integriert. Ersteres findet Anwendung für die Darstellung und Modifikation von Signalen, während letzteres für das Durchführen von Signalanalysen, wie die Berechnung von Formanten, Grundfrequenz, Nulldurchgangsrate, rms-Energie, spektrale Analysen und mehr verwendet wird. Die `snackssff` Bibliothek ist eine Erweiterung des Snack Toolkits um die Verarbeitungsmöglichkeit von Signalen im Simple Signal Datei Format (Cassidy, 2011a), das im EMU-System vorrangig verwendet wird. Die Schnittstelle zur R Programming Language ist durch die Weiterentwicklung plattformunabhängig und leichter auf Systemen zu installieren.

Sämtliche Datenbankanwendungen sind heute über eine grafische Tcl/Tk Benutzerschnittstelle zugänglich, so dass nicht nur der EMU-core Tcl/Tk Befehle zur Verfügung stellen muss, sondern auch andere Bibliotheken wie `libassp`. Für `libassp` stellt `tclassp` Tcl/Tk Kommandos für die Nutzung in den GUIs[13] wie Tkassp bereit. Das Snack Toolkit ist bereits für die Verwendung in einer Tcl/Tk Umgebung aus-

[12] Unterschiede in der Namensgebung ergeben sich durch Arbeitstitel sowie durch die unterschiedlichen Anforderungen bzw. Möglichkeiten der Benennung während der Programmierung und im sprachlichen Gebrauch.

[13] Graphical User Interface

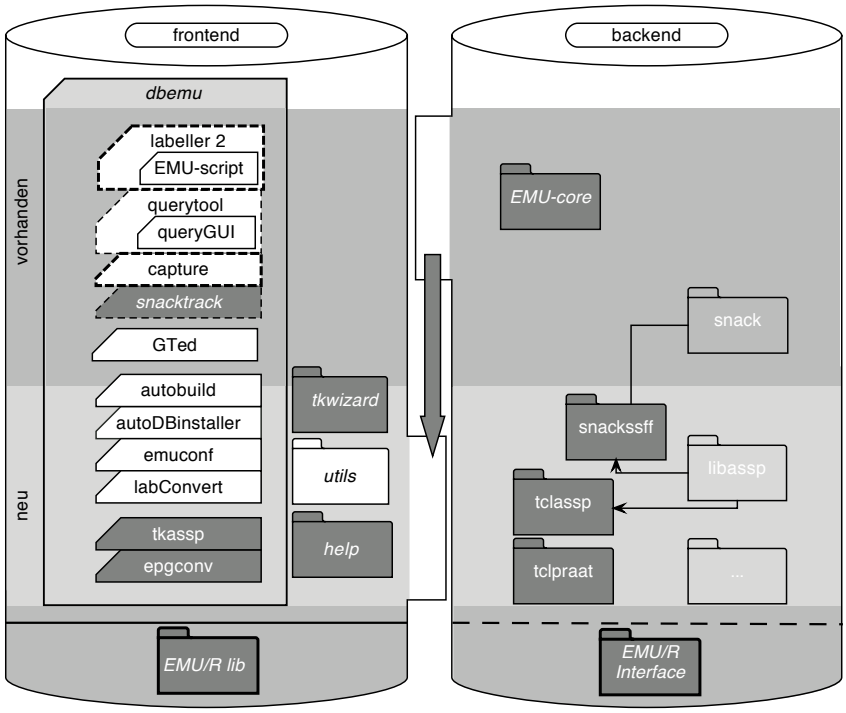


Abbildung 1.2.: Schematische Darstellung des modularen Aufbaus der Implementierung des EMU-Systems mit der Darstellung neuer und vor der Weiterentwicklung bereits vorhandener Funktionalität, weiterhin mit der Angabe vorhandener grafischer Benutzerschnittstellen zur Funktion (gestrichelt), vorhandener aber neu implementierter Teile (fett umrandet), mit der Unterscheidung der Arbeiten, die im Rahmen dieser Dissertation durchgeführt (weiß hinterlegt) bzw. die unterstützend mitentwickelt (kursiv) wurden sowie die hauptsächlich in der Verantwortung anderer Entwicklern lagen (dunkelgrau hinterlegt) als auch externe Programmpakete (gepunktet).

gelegt. Zuletzt stellt die EMU-Bibliothek `tclpraat`, Tcl-Kommandos zu Sendpraat (Boersma und Weenink, 2006) bereit. Sendpraat ist eine Bibliothek von Praat (Boersma, 2002), die es ermöglicht, Befehle an ein laufendes Praat-Programm zu senden. Das EMU-System verwendet diese Bibliothek zum Datenaustausch zwischen EMU und Praat.

Die linke Seite von Abbildung 1.2 zeigt gegenüber 2002 zahlreiche Erweiterungen im Bereich der Applikationen. Die EMU-Erweiterungen, die im Rahmen dieser Dissertation entstanden sind, sind zunächst zur Verbesserung der Benutzbarkeit grafische Benutzerschnittstellen zu Funktionen, die das EMU-System schon bereitstellte. Beispiele hierfür sind der Grafische Template Editor oder das queryGUI, ein GUI zur EMU Query Language. Überarbeitungen vorhandener Applikationen, wie des

Segmenters und die neue Implementierung des EMU Labellers sollen ebenfalls die Benutzbarkeit verbessern.

Das Bereitstellen neuer Funktionen wurde in dieser Dissertation durch die neuen Applikationen labConvert, den autoDBinstaller, den Konfigurationseditor und Script umgesetzt. labConvert konvertiert Etikettierungsdateien verschiedener anderer Tools in das EMU-Etikettierungsformat und andersherum, sowohl skriptbasiert als auch über ein GUI. Der autoDBinstaller ist eine grafische Benutzerschnittstelle zum Importieren von lokal oder auf Servern gespeicherten EMU-Datenbanken in das System. Über den Konfigurationseditor können Speicherorte der Datendefinitionsdateien konfiguriert werden, während Script als Teil des komplett neu implementierten EMU Labellers ein sofort ausführbares Skripten ermöglicht. Weiterentwicklungen am EMU-core gehen auf diese neuen Funktionen und auf Anpassungen an neue Betriebssystemanforderungen zurück. Durch die Weiterentwicklungen in der EMU/R Bibliothek ist das Paket umfangreicher an Funktionen und Dokumentationen aber auch effizienter in der Verarbeitung größerer Datenmengen.

Die von Cassidy und Harrington (2001) bereits erwähnten EMU Labeller Modules wurden bis heute um eine Vielzahl an Modulen ergänzt. Dazu gehören z. B. das EMA-Display, ein Modul zur Darstellung artikulatorischer Daten, die mithilfe eines elektromagnetischen Artikulographen aufgezeichnet wurden und in das EMU kompatible SSF-Format überführt wurden. Auch im Rahmen dieser Dissertation wurde ein Vielzahl EMU Labeller Module jedoch für ganz andere als die vorgesehenen Funktionalitäten implementiert, die mit dem derzeitig veröffentlichten System ausgeliefert werden.

Die Erweiterungsschnittstelle glitt in den letzten Jahren näher in den Kern der Funktionalität, die das EMU-System anbietet. Der modulare Aufbau des Systems soll sich bis zum Anwender bewähren. Jeder Anwender ist je nach individuellen Kenntnissen in der Informatik in der Lage, das EMU-System mithilfe eines EMU Labeller Modules für sich persönlich zu erweitern.

Auch die Interoperabilität des Systems wurde in dieser Dissertation erweitert. EMU bietet nun verschiedene Schnittstellen zu Praat und WaveSurfer an, was die Möglichkeit eröffnet, mit allen dieser Programme auf der selben Datenbasis sowohl dateibasiert als auch datenbankbasiert zu arbeiten. Mit der Interoperabilität zu Praat und WaveSurfer werden somit alle Funktionen der drei Programme auf einer Datenbasis aus einer ganz anderen Software nutzbar. Weiterhin können andere Datenbasen

in das EMU-System integriert werden. Korpora, die mit ganz anderen Systemen aufgebaut wurden, werden hierfür in EMU-Strukturen überführt.

1.4. Übersicht der Kapitel

Die Weiterentwicklungen des EMU Speech Database Systems werden in dieser Arbeit aufgezeigt. Da es sich um eine interdisziplinare Arbeit handelt, die die technische Umsetzung von Arbeitsmaterialien und Handwerkszeug für Geisteswissenschafler im speziellen Phonetiker und Linguisten darstellt, müssen beide Aspekte in dieser Arbeit Berücksichtigung finden. Während die technischen Aspekte separiert dargestellt werden, wird auf die Einführung phonetischer und lingiuistischer Terminologie soweit für das Nachvollziehen verschiedener Umsetzungen nicht notwendig, verzichtet. Ein Glossar im Anhang soll das Verständnis für die in den einzelnen Kapiteln eingeführte Terminologie in Kapiteln die diese Terminologie nicht explizit erneut aufgreifen aber verwenden unterstützen.

Um die theoretischen technischen Aspekte des datenbankbasierten Ansatzes, in diesem Rahmen näher zu bringen, ist es notwendig, einen Überblick über Datenbankkonzepte zu geben, da obgleich das EMU Speech Database System u. U. nicht als solches gesehen wird, Datenbankterminologie zur besseren Referierbarkeit verwendet wird. Dieser Überblick ist in einer Form darzustellen, die für potentielle Nutzer von Sprachdatenbanksystemen, nämlich vorrangig für Wissenschaftler aus der phonetischen Forschung mit weniger Erfahrung auf dem Gebiet der Informatik, ausreichend, aber nicht unnötig komplex ist, um das Prinzip und die in dieser Arbeit verwendete Notation und Terminologie zu verstehen.

Kapitel 2.1 nimmt dieses Vorhaben in Angriff und verzichtet auf eine vollständige Beschreibung aller Konzepte und beschränkt sich auf Konzepte, die in der späteren Beschreibung des Datenbankansatzes des EMU-Systems in Kapitel 3 verwendet werden. Dennoch greift es sehr viele Aspekte auf, die als sehr wichtig für das Verständnis der Konzepte angesehen werden. Somit konnte auf eine gewisse Ausführlichkeit nicht verzichtet werden. Ein umfassender Überblick ist jeweils in der angegebenen Literatur zu finden. So wird das Konzept und die Realisierung von Datenbanken (Kapitel 2.1.1) dargestellt bevor der Begriff Modell anhand von Datenmodellen erläutert und zwei verschiedene Datenmodelle für Datenbanksysteme vorgestellt (Kapitel 2.1.2) werden. Schließlich wird auf Datenbankmanagementsysteme (Kapitel 2.1.3) eingegangen.

Im Kapitel 2.2 wird die Standardsprache SQL, die für die Abfrage und Anfrage von Informationen aus Datenbanken mit zugrundeliegendem relationalen Datenmodell verwendet und für das EMU-System in einem Ausblick als verwendbar eingestuft

wird. Als letzte technische Grundlage wird eine kleine Übersicht über die Backus-Naur Form als Werkzeug für die Beschreibung formaler Sprachen gegeben, da sie für die formale Beschreibung verschiedener Teile des EMU-Systems in dieser Arbeit verwendet wird.

Die technischen Grundlagen sind für die detaillierte Darstellung der zugrundeliegenden Konzepte und Funktionsweisen des EMU-Systems in Kapitel 3 notwendig. In diesem Kapitel wird zunächst das Etikettierungsmodell (Kapitel 3.1), das im System den Etikettierungen von Signalen zugrundeliegt, ausführlich beschrieben. Dabei wird eine Notation für die Konzeptualisierung einer Etikettierungsstruktur vorgeschlagen und die Verwendung modellierbarer Strukturen in der Forschung dargelegt. Weiterhin wird der Datenbankansatz (Kapitel 3.2), nämlich das Konzept und die Realisierung von Datenbanken (Kapitel 3.2.1) und dem Datenbankmanagementsystem (Kapitel 3.2.3), diesmal jedoch mit der speziellen Ausrichtung auf das EMU-System erläutert. Da, wie sich zeigen wird, EMU keines der speziellen Datenbanksysteme ist, das die Standarddatenbanksprache verwenden kann, wird Syntax, Semantik und die Anwendung der EMU eigenen Abfragesprache EMU Query Language in Kapitel 3.3 betrachtet. In der Beschreibung der Grundstrukturen werden die in dieser Dissertation erstellten konzeptuellen Modellierungen und formalen Beschreibungen verschiedener Systemteile erläutert.

Die Modelle und formalen Beschreibungen dienten als Arbeitswerkzeuge in der Weiterentwicklung des Systems selbst, die in drei getrennten Kapiteln erläutert wird. Kapitel 4 wird zunächst einen Einblick in die Weiterentwicklungen der Dokumentationsstrategien des Systems gegeben. Es haben sich in den letzten Jahren im Zuge der Aufgabe, das System benutzerfreundlicher zu gestalten, neue Methoden herausgebildet, die kurz dargestellt werden. Der von anderen Programmen verschiedene Ansatz im EMU-System war in der Vergangenheit für potentielle Benutzer eine ungern überwundene Hürde.

In Kapitel 4.1 wird gezeigt, dass das System nun auf eine Weise dokumentiert worden ist, die den Benutzer in jedem seiner Schritte begleitet, so dass die Vorteile des datenbankbasierten Arbeitens mit dem EMU-System für jeden potentiellen Nutzer zugänglich sind. Ebenfalls zur benutzerfreundlicheren Verwendbarkeit von im EMU-System vorhandener Funktionalität werden die Arbeiten und das Ergebnis der Umsetzung einer grafischen Benutzeroberfläche zur EMU-Abfragesprache in Kapitel 4.2 und der Datenbankschemadeklaration in Kapitel 4.3 dargelegt.

Neben den Weiterentwicklungen der Benutzerfreundlichkeit wurde das EMU-System im Rahmen dieser Dissertation funktional erweitert. Die Arbeiten zu diesen Erweiterungen sind in Kapitel 5 zusammengestellt und umfassen benutzerdefinierte Templatepfade, die Darstellung- und Etikettierungsmöglichkeiten sowie das Schneiden von Signalen als auch Erweiterungen in den *Scripting*-Möglichkeiten des Systems. Zuletzt wird die Entwicklung der Funktion zur Installation und dem Transfer von EMU-Datenbanken gezeigt.

Neben der detaillierten Darstellung des EMU-Sprachdatenbanksystems in Kapitel 3 werden die beiden Programme Praat und WaveSurfer im Rahmen der Interoperabilität zum EMU-System in Kapitel 6 kurz vorgestellt. Dabei werden die Etikettierungsstrukturen und Datenverarbeitungsroutinen berücksichtigt, aber es wird keine umfassende Beschreibung der Programme vorgenommen. Sie wird nur als Grundlage sowohl für den Vergleich als auch für die Möglichkeiten und Probleme für die Interoperabilität der beiden Programme zum EMU-System verwendet.

Im Kapitel wird außerdem die Entwicklung verschiedener Methoden gezeigt, wie Schnittstellen zu den beiden Programmen zur Verfügung gestellt werden konnten und zu welchen unterschiedlichen Ergebnissen sie führen. Hierbei werden Informationsverluste, die durch die Methoden und durch die Unterschiede der Programme im Umgang mit Etikettierungen bedingt sind, als auch die neuen Möglichkeiten, die dabei entstehen, offengelegt. Es wird gezeigt werden, dass durch die Weiterentwicklung Praat, EMU und WaveSurfer sowie auch weitere Programme über die vorgestellten Schnittstellen kommunizieren können und somit die Funktionalitäten aller einzelnen Programme durch die Weiterentwicklung auf nur einer Datenbasis nutzbar sind.

Die Arbeiten für eine umfangreiche Konvertierung von Signal- und Etikettierungsdateien eines umfangreichen Korpus zur Integration in eine EMU-Datenbank, um die Verwendung aller Datenbankvorteile für das Korpus zu ermöglichen, werden in Kapitel 7 anhand des Kiel Corpus detailliert beschrieben. Dieses Kapitel soll zeigen, dass zum einen die Überführung eines dateibasierten Korpus in eine Sprachdatenbank möglich ist und das zum anderen eine strukturierte Etikettierung unter Verwendung eines Etikettierungsschemas, dem ein Modell zugrundeliegt, konsistenter und übersichtlicher in der Darstellung sein kann.

Die aus der Überführung entstandene EMU-Datenbank in Kapitel 7 wird bereits in den vorangegangenen Kapiteln als Beispieldatenbank verwendet, so dass der Nutzen und die Anwendbarkeit des Korpus als Datenbank mit jedem Kapitel gezeigt wird. Die Datenbank weist eine übersichtliche Struktur auf (Kapitel 3.1) und sie ist um-

fangreich abfragbar (Kapitel 3.3). Die im Rahmen dieser Dissertation erzeugte Kiel Corpus Datenbank kann außerdem durch die von EMU gegebene neu entwickelte Interoperabilität (Kapitel 6) auch mit Praat oder WaveSurfer verwendet werden.

Die vorgestellten bereits umgesetzten Weiterentwicklungen besonders im Frontend des Systems werden schließlich durch einen Ausblick in die Weiterentwicklung in Bezug auf das Backend in Kapitel 8 ergänzt. In diesem Kapitel wird auf Grundlage eines älteren Experimentes die Abfragemöglichkeit von EMU-Datenbanken mit der Standarddatenbanksprache als Alternative diskutiert. Die Verwendung von SQL sollte aufgrund des zugrundeliegenden Modells effizienter sein als die EMU eigene Abfragesprache und vorhandene Einschränkungen in den Abfragemöglichkeiten selbst sollten aufgehoben werden. Unmittelbar damit verbunden ist die Überführung des aktuell verwendeten Datenbankmodells im EMU-System in ein alternatives Modell, dem relationalen Datenbankmodell. Diese Überführung wird im Kapitel evaluiert, diskutiert, und es werden Überarbeitungsanregungen formuliert.

2. Technische Grundlagen

Dieses Kapitel stellt eine Zusammenstellung der technischen Grundlagen dar, die dazu dienen soll, die verwendete Terminologie in dieser Arbeit nachvollziehen zu können. Es folgt ein Einblick in die Datenbankkonzepte, da sich mithilfe dieser Konzepte die Struktur und Funktionsweise des EMU Speech Database Systems in Kapitel 3 sehr gut erklären lassen. Die Abfragesprache SQL wird kurz vorgestellt, da sie als mögliche Abfragesprache in der weiteren Entwicklung des EMU-Systems in Zukunft in Kapitel 8 diskutiert wird. Weiterhin wird die als Werkzeug zur Beschreibung formaler Sprachen verwendbare Backus-Naur Form vorgestellt, denn in dieser Form sind verschiedene Sprachen im EMU-System im Rahmen dieser Dissertation formal beschrieben worden.

2.1. Datenbanksystem

In diesem Kapitel werden Datenbankkonzepte dargestellt, um einem Phonetiker, der ein Sprachdatenbanksystem wie EMU verwenden möchte, die grundlegenden Strukturen näher zu bringen. Für die genauere Darstellung sei auf Einführungsbücher (Elmasri und Navathe, 2009; Mertens, Bodendorf, König, Picot, Schumann und Hess, 2004; Biethahn, Mucksch und Ruf, 2007) verwiesen, die auch für die Zusammenfassung hier verwendet wurden. Die Literatur wird an unterschiedlichen Stellen diskutiert, wenn Betrachtungsweisen auseinandergehen. Die Diskussionen sollen vermeiden, dass in dieser Arbeit irgendeine Stellung bezogen wird, denn die Konzepte sollen nur dargestellt werden und stehen hier selbst nicht zur Diskussion.

> "In the early days of our field, data was intimately tied to the application programs that used it. Now we see that we want to break that tie. We want data that is independent of the application programs that use it – that is, data that is organized and structured to serve many applications and many users. What we seek is the data base." (Bachman, 1973, S. 653)

Datenbanksysteme (DB-Systeme) werden heutzutage in unterschiedlichen Formen in ganz unterschiedlichen Bereichen, die mit elektronischer Datenverarbeitung in Verbindung stehen, eingesetzt. Sie spielen in fast allen Bereichen, in denen Computer eingesetzt werden, eine entscheidende Rolle. Beispiele, in denen Datenbanken zum Einsatz kommen, sind u. a. die Bereiche Medizin, Recht und Bildung, aber auch traditionellere Datenbankanwendungen wie im Reisebüro, in der Bank oder in computergestützten Bibliothekskatalogen. Das Hauptziel eines Datenbanksystems ist es, Informationen effizient zu speichern, zentral zu verwalten und dem Menschen in komfortabler Form zugänglich zu machen. (Elmasri und Navathe, 2009)

Ein Datenbanksystem setzt sich aus den Komponenten Datenbank und Datenbankmanagementsystem (DBMS) zusammen, das die Routinen der zentralen Verwaltung übernimmt. Die Datenbankkomponente selbst enthält strukturierte Daten, die Informationen speichern, und die Strukturbeschreibung der Daten selbst (Mertens et al. 2004). Die Strukturierungsmöglichkeiten und die wirkliche Struktur der Daten unterliegen einem Datenmodell. Die Art des Datenbanksystems richtet sich nach diesem Datenmodell.

Ein Datenmodell gibt einen formalen Rahmen für die Beschreibung von Datenstrukturen und Operationen auf die Datenstrukturen vor. Eine solche Beschreibung wird als Datenbankschema bezeichnet. Die gespeicherten Informationen müssen in der Datenbank genauso strukturiert sein, wie es das Datenbankschema beschreibt (Kemper und Eickler, 2006).

Über die Strukturbeschreibungen in der Datenbankkomponente kann ein Benutzer Anfragen an das DBMS stellen, das die Zugriffsberechtigungen prüft, den Ort der gespeicherten Information ermittelt und dem einzelnen Benutzer die angefragten Informationen zur Verfügung stellt. Je nach Nutzer und Problemstellung wird nur ein kleiner Teil des Datenbestands als Information benötigt, die das DBMS extrahiert. Weiterhin soll der Benutzer über das DBMS je nach Autorisierung die Datensätze abfragen, verändern, einfügen und löschen können (Biethahn et al. 2007).

Für die Nutzung von Datenbanksystemen muss von unterschiedlichen Benutzergruppen ausgegangen werden, die unterschiedliche Anforderungen und Zugriffswünsche an das System haben. Zu den Benutzergruppen gehören die Endbenutzer, Anwendungsprogrammierer sowie die Datenbankadministratoren. In dieser Reihenfolge steigt der Bedarf an Informationen, die das System an den Benutzer zurückgibt. Derartige unterschiedliche Sichtweisen auf das System werden Sichten genannt (Elmasri und Navathe, 2009).

Sie müssen bereits während der Datenmodellierung berücksichtigt werden. Hierbei sind Sichten aus drei Modellierungsebenen zu unterscheiden. Interne Sichten enthalten die physischen Details, wie die Daten im Speicher abgelegt sind, externe die anwendungsbezogene logische Organisation der Daten und konzeptuelle Sichten enthalten die reine Information, die in unterschiedlichen konzeptuellen Sichten weiter eingeschränkt werden kann. Beim Zugriff auf Information ist der Endbenutzer in der Regel nicht an Datenverwaltungsdetails interessiert, sondern an der Information, die verwaltet wird. Jedoch ist auch nicht jede Information interessant. So ist die Information über die Körpergröße einer Person, deren Daten in der Datenbank vorliegen, überflüssig bei der Abfrage der Postanschrift der Person. Ein Praktikant, der Werbepost versenden soll und dafür Zugriff auf die Daten bekommt, sollte nicht aufgrund der Abfrage der Anschrift auch den Kontostand geliefert bekommen.

Während ein Datenbanksystem die Datenbank und das DBMS die Daten und das Datenbankschema verwaltet, stellen die Datenbankanwendungen eine Schnittstelle zwischen Benutzer und DBMS her. Derartige Programme bieten meist grafische Benutzerschnittstellen für das Abrufen und das Verändern von Datensätzen in der

Datenbank. Beispiele hierfür sind Formulare, wie sie für die Anmeldung zu Konferenzen oder zur Buchung von Hotelzimmern eingesetzt werden. Weiterhin können sie die Stapelverarbeitung (*batch processing*) unterstützen, d. h. die nicht interaktive sequentielle Bearbeitung von Aufgaben, wie z. B. das Konvertieren von Bildern oder Zeitsignalen in ein bestimmtes Format.

Im Entwurfsprozess einer Datenbank wird zunächst eine Anforderungsanalyse durchgeführt, in der festgelegt wird, welche Daten die Datenbank enthalten soll und welche Operationen auf den Daten durchführbar sein sollen. Die zu verarbeitenden Daten stammen in der Regel aus der realen Welt, wie zum Beispiel Personendaten oder von der realen Welt bereits diskretisierte Daten wie Messergebnisse oder etikettierte Sprachdaten. Dieser Ausschnitt der realen Welt wird anschließend konzeptuell modelliert. Eine vollständige Modellierung des Ausschnitts der realen Welt gilt als Schema, hierbei ist die Wahl des Datenbanksystems noch nicht relevant. Aufgrund des konzeptuellen Schemas wird ein logisches Schema erstellt, das den Anforderungen des gewählten Datenbanksystems angepasst ist. Schlussendlich wird dieses Schema physisch umgesetzt, so dass Dateien und Indexstrukturen auf dem Rechner festgelegt werden. Der Nutzer ist in der Regel nur mit der konzeptuellen Modellierung konfrontiert. Datenbankadministratoren und Programmierer sind für die Umsetzung auf den anderen beiden Ebenen zuständig. Die unterschiedlichen Betrachtungsweisen (umgesetzt als sog. Sichten) und Anforderungen sind auch während der Entwurfsphase einer Datenbank vorhanden. Die Speicherstrukturen werden erst festgelegt, nachdem ein Konzept der Datenbankinhalte festgelegt ist. Somit sind unterschiedliche Datenmodelle für die Beschreibung der Informationen notwendig.

Jedem Datenbanksystem liegt ein logisches Datenmodell, ein Implementierungsdatenmodell[1], zu Grunde. Dieses Modell und dessen Implementierung sind maßgeblich für die Effizienz des gesamten Systems, d. h. die Geschwindigkeit des Zugriffs auf die Daten. Für die konzeptuelle Strukturierung einer Datenbank kann vom logischen Datenmodell abgewichen und ein konzeptuelles Modell[2] benutzt werden. Die Literatur bietet eine Vielzahl von Modellen zur Datenmodellierung an: Das Netzwerk-Datenmodell (CODASYL, 1971), das Hierarchische Modell (McGee,

[1] In der Literatur finden sich sowohl die Begriffe Implementierungsmodell wie in Saake, Sattler und Heuer (2010) als auch Implementierungsdatenmodell wie in Elmasri und Navathe (2009). Da es sich im Kontext von Datenbanken immer um Datenmodelle handelt, lässt sich dieser Unterschied durch die Redundanz der Wortkonstituente <daten> leicht erklären. Natürliche Sprache hat eine kaum kontextfreie Semantik.

[2] Auch hier wird in der Literatur auch innerhalb eines Buches von Konzeptdatenmodellen oder von konzeptuellen Modellen als auch konzeptuellen Datenmodellen gesprochen, was sich wiederum durch die Redundanz der Konstituente <daten> in diesem Kontext erklären lässt.

1977), das Entity-Relationship Modell (Chen, 1976), das relationale Modell (Codd, 1970) und das Objektbasierte Modell (Cattell und Atwood, 1993). Besonders in der Konzeptualisierung einer Datenbank kann das verwendete Modell von standardisierten Vorlagen sehr stark abweichen und zugunsten programmeigener Modelle unberücksichtigt bleiben (Elmasri und Navathe, 2009).

In Kapitel 3 wird das Sprachdatenbanksystem EMU vorgestellt, d. h. es wird gezeigt, wie Datenbanken und Datenbankmanagementsystem in diesem System umgesetzt werden. Außerdem wird in Kapitel 3.1 ein programmeigenes konzeptuelles Modell vorgestellt und in Kapitel 8 wie Teile dieses Modells in ein relationales Modell übersetzt werden könnten.

Im Folgenden werden die einzelnen Komponenten eines Datenbanksystems und Datenbankmodelle näher betrachtet. Um Datenbankgrundbegriffe zu erklären, die später verwendet werden, soll im Folgenden beispielhaft ein kleiner Einblick in das heute oft verwendete und gelehrte Entity-Relationship Modell (ER-Modell) und das relationale Modell gegeben und auch die beiden grundlegenden Bestandteile eines Datenbanksystem näher betrachtet werden. Es sei noch einmal darauf hingewiesen, dass dieses Kapitel keineswegs alle Datenbankkonzepte abdeckt, sondern nur Begriffe klärt, die für das Verständnis des datenbanksystemähnlichen Ansatzes des EMU-Systems, wie es in Kapitel 3 beschrieben wird, notwendig sind.

2.1.1. Datenbank

Wie die folgenden beiden Zitate zeigen, ist der Begriff Datenbank nicht eindeutig zu beschreiben. Aus diesem Grund werden im Folgenden mehrere Quellen miteinander verglichen.

> „Eine Datenbank ist zunächst einmal nichts anderes als eine Sammlung von irgendwelchen Objekten.“ (Simpson, 1998, S. 15)

> „Eine Datenbank ist eine Sammlung von Daten, die einen Ausschnitt der realen Welt enthalten.“ (Elmasri und Navathe, 2009, S. 18)

Die beiden Zitate weisen eine sehr allgemeine Definition von Datenbank auf. Beide verwenden den Begriff Sammlung, wobei die Sammlung in der ersten Definition aus irgendwelchen Objekten besteht, während in der zweiten Definition diese auf Daten eingeschränkt wird. Zieht man folgende Definition von Mertens et al. (2004) zu Daten heran, so ist zu erkennen, dass der Begriff Daten in der Definition von

Elmasri und Navathe den gleichen außersprachlichen Referenten hat wie der Begriff Objekte in der Definition von Simpson (1998).

> „Daten werden [...] als eine Folge maschinell verarbeitbarer Zeichen [...] verstanden, die Objekte und Objektbeziehungen der Realwelt durch ihre Merkmale beschreiben und damit repräsentieren." (Mertens et al. 2004, S. 55)

Unter diesen Definitionen kann jegliche auch zufällige Sammlung an Objekten aus der realen Welt als Datenbank bezeichnet werden, so auch die unformatierte Datensammlung, wie sie das WWW darstellt. Andere Beispiele sind Textdokumente, in denen maschinell verarbeitbare Zeichen aneinandergereiht sind und in einer Datei gespeichert werden. Die einzelnen Zeichen, bzw. die Wörter und Sätze, bilden unter den gegebenen Definitionen eine Datenbank. Mehrere solcher Dateien bilden wiederum eine Datenbank, wobei die Dateien selbst logisch zusammengehörig sein müssen.

Auch Elmasri und Navathe (2009, S. 18) schränken Datenbanken in einer eingeschränkten Sicht auf logisch zusammenhängende Daten mit „inhärenter Bedeutung" ein und schließen damit in dieser Sicht „zufällige Datensammlungen" unter dem Begriff Datenbank aus.

Nach Kemper und Eickler (2006) ist eine Datenbank jedoch nicht nur die Sammlung logisch zusammenhängender Daten, vielmehr setzt sich eine Datenbank aus der Datenbasis (Datenbankausprägung) und der strukturellen Beschreibung der Datenbasis (Datenbankschema) zusammen. Die Datenbasis bezeichnet die in geeigneter Form gespeicherte Sammlung logisch zusammenhängender Daten. Eine derartige Zusammensetzung einer Datenbank wird von Elmasri und Navathe (2009) nicht beschrieben. Sie sehen die Definition der Datenbankstruktur als Teil des Datenbanksystems, der in einem Datenbankkatalog gespeichert ist.

In der Beschreibung eines Datenbankstandards SQL2 (Date und Darwen, 1998) unterscheiden Elmasri und Navathe (2009, S. 183) Schemata in- und außerhalb des Datenbankkatalogs. So ist ein im Datenbankkatalog enthaltenes Schema „ein spezielles Schema namens 'INFORMATION SCHEMA', das [...] Informationen zu allen Element-Deskriptoren aller Schemas im Katalog bereitstellt." Ohne den Begriff ‚Element-Deskriptoren' näher zu erläutern, erschließt sich, dass ein ‚Informations-Schema' wiederum eine strukturelle Beschreibung der einzelnen Datenbankschemata ist. Die strukturelle Beschreibung der Datenbasis, die zusammen mit der Datenbasis

selbst nach Kemper & Eickler die Datenbank bilden, führen Elmasri & Navathe in der Beschreibung von SQL2 unter dem Begriff ‚SQL-Schema' an, obgleich auch an dieser Stelle nicht explizit hervorgeht, dass es ein Teil der Datenbank ist.

Das Konzept eines Datenbankkatalogs wird in Kemper & Eickler nicht aufgegriffen. Obwohl sich die Quellen an diesem Punkt unterscheiden, betonen beide die Notwendigkeit der Trennung von Datenbankschema und Datenbankausprägung (Kemper & Eickler) bzw. Datenbankinstanz (Elmasri & Navathe). Im Folgenden wird diese Trennung vom Begriff der Datenbasis ausgehend verdeutlicht.

Die Datenbasis besteht aus Informationen, die in Textform oder numerisch gespeichert werden. Sie bildet einen kleinen Teil der realen Welt ab. Hierfür wird die reale Welt soweit abstrahiert, bis sie für den jeweiligen Zweck handhabbar ist. In Bezug auf Datenbanken bedeutet das, Informationen so aufzubereiten, dass sie rechnergestützt abrufbar sind. Weiterhin muss die Datenbasis strukturiert sein. Das Abbilden der realen Welt und die Strukturierung der Daten erfordert eine Modellierung dieser.

Auch in der Phonetik wird die reale Welt abstrahiert, um sie rechnergestützt verarbeiten zu können. Luftdruckschwankungen über der Zeit, die durch eine Äußerung eines Sprechers über eine Membran in einem Mikrofon übertragen werden, müssen digitalisiert werden. Hierfür wird das analoge Signal der Membranschwankungen abgetastet, um einzelne Zeitpunkte des Signals zu erfassen und die Amplitude der Membranbewegung, die durch die unterschiedlichen Luftdrucke entsteht, wird diesem Zeitpunkt in Stufen quantisiert zugeordnet. Das Signal besteht in digitalisiertem Zustand aus einer endlichen Menge an Zeitpunkten in einem festen zeitlichen Intervall und den dazugehörigen Amplitudenwerten, die ebenfalls aus einer endlichen Menge an Werten stammen. Eine Digitalisierung ist somit eine Modellierung der unterschiedlichen Molekülдichten in der Luft in Form von Amplituden über einen Zeitraum. Die modellierten Amplituden und auch die Zeiten, die zu diskreten Werten abstrahiert sind, nehmen in der realen Welt innerhalb eines bestimmten Bereiches unendlich viele Werte an.

Die Datenbasis in Datenbanksystemen wird in einer anderen Form modelliert. Verschiedene Datenmodelle werden in Kapitel 2.1.2 kurz vorgestellt. Eine vollständige Modellierung der Daten resultiert in einem Datenbankschema, welches die Struktur der Datenbasis beschreibt, somit eine Datenbankbeschreibung darstellt und als Metainformation einen Teil der Datenbank ausmachen kann. Vom Schema ausgehend betrachtet, müssen die Daten in der Datenbank diesem Schema „gehorchen" (Kemper und Eickler 2006, S. 22). Dafür ist es notwendig, im Datenbankaufbau

zunächst ein Schema zu definieren, ohne die Datenbank mit Daten zu füllen. In diesem Zustand ist die Datenbank leer. Die Informationen selbst können dann gemäß des Schemas strukturiert der Datenbank hinzugefügt werden. Jeder so entstehende Datensatz ist eine Instanz der Datenbank bzw. eine Datenbankausprägung. Die Datenbankausprägung stellt somit „den momentan gültigen Zustand der Datenbasis“ (Kemper und Eickler 2006, S. 22) dar.

2.1.2. Datenmodelle

Einem Datenbanksystem liegt ein Datenmodell zugrunde, das „die Infrastruktur für die Modellierung der realen Welt zur Verfügung stellt.“ (Kemper und Eickler 2006, S. 21). Die meisten Datenmodelle modellieren Gegenstände und Konzepte als Objekte, wobei Objekte mit Attributen versehen sind, die die Eigenschaften der Objekte beschreiben. Es wird ganz klar zwischen Typebene und Ausprägungsebene getrennt. Das heißt, ein Objekt der realen Welt ist eine Ausprägung eines modellierten Typs, der wiederum für alle Ausprägungen identisch ist. Die Unterschiede zwischen verschiedenen Datenmodellen ergeben sich durch die unterschiedliche Strukturierung der Objekte. Unterschiedliche Strukturierungen eröffnen unterschiedliche Operationsmöglichkeiten auf den Daten.

Unter Berücksichtigung von Datenmodellen werden Datenbankschemata entworfen, die die strukturelle Beschreibung der Datenbank darstellen. Hierbei werden u. a. konzeptuelle, physische Datenmodelle und Implementierungsdatenmodelle unterschieden. Ersteres stellt Konzepte zur Verfügung, die der Benutzer in seiner realen Welt wahrnehmen kann. Implementierungs(daten)modelle stellen anders als die konzeptuellen (Daten)Modelle eine Instanz zwischen Konzept und physischer Speicherung dar. Konzeptuelle Modelle und Implementierungsmodelle verbergen die Details der Datenspeicherung. Derartige Details werden mithilfe eines physischen Datenmodells umgesetzt. In diesem Modell werden Datentypen und Funktionen definiert, die direkt in die Implementierung einfließen. Auf eine nähere Beschreibung wird hier verzichtet, da sie aus Benutzersicht nicht mehr relevant sind.

2.1.2.1. Entity-Relationship Modell

Das Entity-Relationship (ER) Modell ist ein konzeptuelles Datenmodell, das Objekte aus der realen Welt als Entitäten mit Attributen darstellt und durch Beziehungen (Relationships) Zusammenhänge zwischen diesen Entitäten herstellt. Entitäten, At-

tribute und Beziehungen können spezielle Eigenschaften haben, die vom Modell vorgegeben sind. Eine Entität könnte zum Beispiel eine Person X sein, die als Attribut eine Postanschrift besitzt und ein gewisse Körpergröße aufweist. Diese Person hat eine Beziehung zu einer anderen Entität, denn die Person kauft in einem Kaufhaus A ein. Andere Personen Y und Z gehen auch in Kaufhäusern A oder B einkaufen. Y und Z haben als Personen gemeinsame Eigenschaften (Attribute) und A und B haben als Kaufhäuser gemeinsame Eigenschaften. Person und Kaufhaus sind also Oberbegriffe für X, Y und Z bzw. A und B, sog. Entitätstypen[3]. Somit ist festzustellen: es gibt in der realen Welt einen Beziehungstyp, der die Entitätstypen Person und Kaufhaus in Verbindung bringt und zwar durch die Beziehung, dass sie dort einkaufen. Als Entitätstypen können nicht nur physisch vorhandene Dinge modelliert werden, sondern auch nicht gegenständliche Konzepte wie die Eigenschaft Kunde einer Bank zu sein.

Zum Modell gibt es festgelegte Konventionen zur Notation. Die Modellierung des Datenbankschemas kann in grafischer Form unter der Verwendung von Schemadiagrammen erfolgen. Entitätstypen werden im ER-Modell beispielsweise als beschriftete Rechtecke dargestellt, während Attribute als Ovale über eine Linie mit dem Rechteck verbunden sind. Eine Beziehung wird durch eine Linie zwischen zwei Entitätstypen dargestellt und die Art der Beziehung mithilfe einer beschrifteten Raute in der Mitte der Verbindungslinie spezifiziert. Die Beschriftung ist dabei jedoch optional und nur zur besseren Veranschaulichung ggf. notwendig.

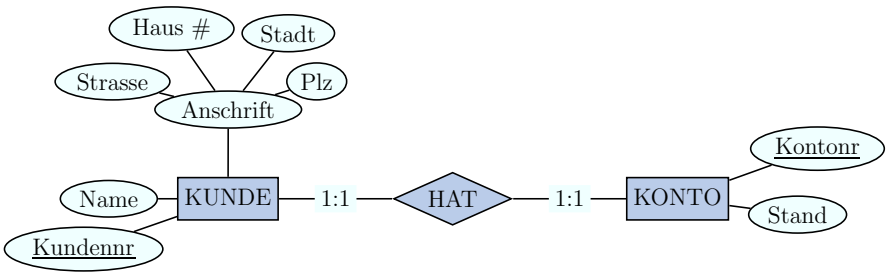


Abbildung 2.1.: Ein Beispiel für ein Schemadiagramm unter der Verwendung des Entity-Relationship Modells mit den zwei Enititäẗstypen **KUNDE** und **KONTO** mit jeweils verschiedenen Attributen und dem Beziehungstyp **HAT**.

Abbildung 2.1 modelliert im Entity-Relationship Modell einen Beziehungstyp (Relationship-Typ), der einen Ausschnitt der realen Welt abbildet, nämlich die Tatsache,

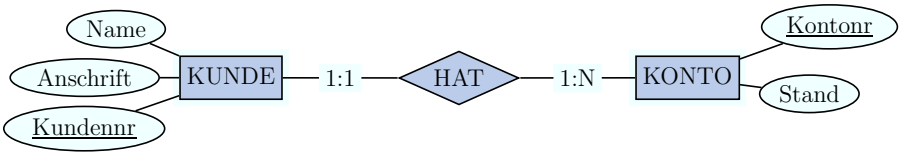
[3]Formale Definition von Entitätstypen: Sammlung von Entitäten, die die gleichen Attribute besitzen.

dass Kunden einer Bank bei einem Institut Konten haben. Dabei haben Kunden die als Attribute modellierten Eigenschaften, Name sowie Anschrift und werden innerhalb des Instituts über ihre Kundennummer eindeutig identifiziert. Die Anschrift zeigt sich in Abbildung 2.1 als ein strukturiertes Attribut, dass aus weiteren Attributen zusammengesetzt ist. Konten hingegen werden über die Kontonummer eindeutig identifizierbar und besitzen die Eigenschaft, einen Kontostand zu haben. Attribute, die zu einer eindeutigen Identifizierung beitragen, werden im ER-Modell unterstrichen und werden Schlüsselattribute genannt.

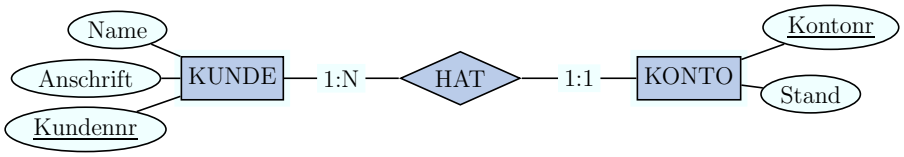
Die Beziehungen zwischen Entitätstypen und damit einhergehend der Entitäten können eingeschränkt werden, indem wie in Abbildung 2.2 auf den Linien eine *min:max* Notation eingefügt ist. In dieser Arbeit werden Einschränkungen auf der Seite des betroffenen Entitätstypen eingefügt, da es für diese Arbeit intuitiver ist. In der Literatur werden unterschiedliche Notationen der Beschränkungen verwendet. Es werden die Bedingungen auch „auf den entgegengesetzten Seiten“ dargestellt (Elmasri und Navathe, 2009, S. 81)[4].

Die *min:max* Notation gibt an, wie oft eine Entität an der gleichen Beziehung beteiligt sein darf. So kann in Abbildung 2.1 nur jeweils ein Kunde und ein Konto zusammen in einer Beziehung stehen. Dies gilt trotz der Einschränkung für jedes Konto und jeden Kunden. Die Bank kann also dennoch mehrere Kunden und Konten verwalten. Das gilt auch für jedes dargestellte Schema in der Abbildung 2.2. Die Einschränkung in (a) sieht für die Beziehung Kunde und Konto die Möglichkeit vor, das genau ein Kunde in mindesten einer aber darüber hinaus in beliebig vielen Beziehungen (N) mit verschiedenen Konten auftreten darf. Für die Konten gilt dennoch umgekehrt, dass sie nur einen Kunden als Kontoinhaber haben können. Genau andersherum und vergleichbar mit der Kontoführung für Ehepaare ist die Beschränkung in (b). Hier können ein oder mehrere Kunden ein und dasselbe Konto haben. In (c) wird ein Sonderfall für Banken dargestellt, indem ein Kunde auch gar keine oder genau eine Beziehung zu einem Konto haben kann. Für Banken ist das der Fall, wenn der Kunde das Konto gerade anlegt oder aufgelöst hat. Die Person mit ihren Attributen ist dann in der Datenbank für mögliche Werbezwecke u. ä. vorhanden, das Konto jedoch noch nicht bzw. nicht mehr. Es können weitere Beschränkungen formuliert werden. Nur die hier aufgeführten werden jedoch in dieser Arbeit Verwendung finden, sodass für weitere Informationen auf die Literatur verwiesen wird.

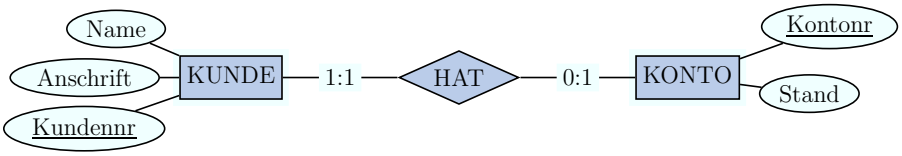
[4] Auf die verschiedenen Möglichkeiten weisen Elmasri und Navathe (2009) hin und verwenden eine andere Ausrichtung als in dieser Arbeit als auch z. B. Biethahn et al. (2007, S. 71).



(a) Ein Kunde hat mindesten ein oder mehrere Konten.



(b) Ein oder mehrere Kunden haben genau ein gemeinsames Konto.



(c) Ein Kunde hat ein oder kein Konto.

Abbildung 2.2.: Beispiel für Schemadiagramme mit unterschiedlichen Beschränkungen in min:max Notation und ihrer Bedeutung in (a–c) unter der Verwendung des Entity-Relationship Modells mit den zwei Enititätstypen **KUNDE** und **KONTO** mit jeweils verschiedenen Attributen und dem Beziehungstyp **HAT**.

Ein weiterer in Abbildung 2.3 dargestellter Aspekt sind die so genannten Rollen. Hiermit werden die Rollen, die die Entitäten in einer Beziehung spielen können, markiert. In den Beispielen vorher sind Entitäten des Entitätstyp **KUNDE** im Beziehungstyp zu **KONTO** Kunden oder auch Sparer. Diese Rolle, die der Entität zugesprochen wird, ist in den abgebildeten Schemata nicht eingezeichnet, da es aus der Entitätstypbezeichnung hervorgeht. Ist ein solcher Zusammenhang aus dem Kontext nicht ersichtlich, werden die Linien zusätzlich mit der Rollenbezeichnung beschriftet. Abbildung 2.3 zeigt eine wahrscheinlich nicht verwendete aber für die Erklärung der Verwendung von Rollen geeignete Modellierung der Situation auf, dass Geld sparende Kunden in der Bank ein Konto führen dürfen, Schuldner jedoch nicht.

Zur nicht grafischen Notation im laufenden Text sei angemerkt: es werden in der Literatur unterschiedliche Formatierungen genutzt. In dieser Arbeit werden Beziehungstypen in Großbuchstaben und Attribute in genormter Orthographie oder gänzlich in Kleinbuchstaben notiert. Abweichende Notationen werden verwendet, wenn der Kontext dies für eine bessere Verständlichkeit erfordert. Dann werden Entitätstypen

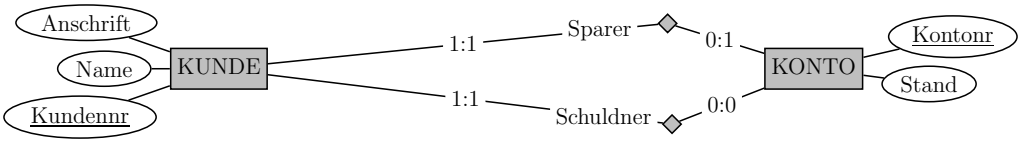


Abbildung 2.3.: Ein konstruiertes Beispiel für ein Schemadiagramm mit Rollen (Sparer, Schuldner) unter der Verwendung des Entity-Relationship Modells mit den zwei Enititätstypen **KUNDE** und **KONTO** mit jeweils verschiedenen Attributen und einem unbeschrifteten Beziehungstyp.

unterstrichen. Zum Hervorheben der Modellkonstrukte werden diese gegenüber dem Text in Fettdruck dargestellt.

2.1.2.2. Relationales Datenmodell

Das relationale Datenmodell ist ein Implementierungsdatenmodell (vgl. Elmasri und Navathe (2009, S. 40)). Die einzigen Objekte, die dieses Modell zur Verfügung stellt, sind Relationen. Somit enthält eine Datenbank ausschließlich Relationen, die Ausprägungen verschiedener Relationstypen sind.

Für jeden Entitätstypen eines konzeptuellen ER-Modells gibt es im Datenbankschema einen Relationstypen. Ein Relationstyp kann als leere Tabelle aufgefasst werden, die als Spaltenbezeichnung die Attributnamen trägt. Die Beziehungen aus dem ER-Modell können z. B. mithilfe gleicher Attributnamen in verschiedenen Relationstypen umgesetzt werden, wie in Schema 1 beispielhaft aufgrund der ER-Modellierung in Abbildung 2.1 (S. 35) umgesetzt wurde. Es zeigt zwei Relationstypen mit den Attributen der Entitätstypen aus Abbildung 2.1. Die Beziehung zwischen Kunde und Konto wird über das hinzugefügte Attribut Kundennummer im Relationstyp **KONTO** beibehalten. Ein solches Attribut wird Fremdschlüsselattribut genannt. Auf diese Weise sind zwei Tabellen durch die gleichen Attribute verbunden und bilden einen neuen Relationstyp, wenn die Attributwerte identisch sind. Alternativ können alle Informationen in einem Relationstyp zusammengefasst werden, wie in Schema 2 dargestellt.

Die Relationen sind erst in einem Datenbankzustand enthalten, wenn die Tabelle mit Werten gefüllt ist, d. h. wenn ein Datensatz (eine Zeile in der Tabelle mit einem Wert pro Spalte) in die Datenbank eingefügt wurde. Eine Relation bestimmt sich durch die Werte für die Attribute im Relationstyp. Diese Wertemenge wird Tupel genannt. In Ausprägung 1 sind Instanzen (Relationen) den Relationstypen hinzugefügt. Es ist

somit ein Datenbankzustand dargestellt. Er enthält Relationen mit unterschiedlichen Tupeln (Menge der Werte in einer Tabellenzeile: (Kundennr: 0001, Anschrift: Kiel, Name: Hanson) oder kurz (0001, Kiel, Hanson)).

Schema 1.

KUNDE

Kundennr	Anschrift	Name

KONTO

Kontonr	Stand	Kundennr

Schema 2.

KUNDENKONTEN

Kundennr	Anschrift	Name	Kontonr	Stand

Operationen, wie Einfügen, Löschen und Aktualisieren der Datenbasis sind ebenfalls im Modell enthalten, sowie Operationen für das Formulieren von Anfragen, also zur Suche nach Relationen. Für Letzteres steht eine Relationenalgebra zur Verfügung. Die JOIN Operation zum Verbinden von Tupeln verschiedener Relationen oder die SELECT Operation zum Auswählen einzelner Attribute aus einer Relation sind u. a. in der relationalen Algebra definiert.

Ausprägung 1.

KUNDE

Kundennr	Anschrift	Name
0001	Kiel	Hanson
0002	Hamburg	Meyer
0003	München	Tegel
0004	Hamburg	Schmidt

KONTO

Kontonr	Stand	Kundennr
23417	1000	0001
56767	1300	0002
97823	800	0003
68945	700	0004

2.1.3. Datenbankmanagementsystem

> "Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation)." (Borden, Harris und Raphael, 2003, S. 377)

> "One of the most important features of the DBMS is it's ability to shield the people and programs using the data from the details of its physical storage." (Haigh, 2006, S. 1)

Auf Grundlage der beiden vorangegangenen Zitate stellt das Datenbankmanagementsystem (DBMS) eine absolut notwendige Barriere zwischen Nutzer und physischem Speicher dar. In der Tat ist diese Barriere vorhanden, aber je nach Nutzer und entsprechenden Zugriffsrechten mehr oder weniger durchlässig. Über das Datenbankmanagementsystem kann ein Benutzer anhand der Datenbankbeschreibung (Datenbankschema, vgl. S. 32ff.) auf die eigentlichen Informationen in der Datenbank zugreifen. Das DBMS stellt dafür mit unterschiedlichen formalen Sprachen Benutzerschnittstellen zur Verfügung.

Zu diesen Sprachen gehören die DDL (*Data Definition Language*) zur Beschreibung der Datenobjekte und die DML (*Data Manipulation Language*), die in der Literatur oft synonym mit *Query Language* (QL) gebraucht wird. Die DML dient der Abfrage, Anfrage sowie der Manipulation der Daten und Datenobjekte. Die QL im engeren Sinne ist eine Abfragesprache, durch die der Benutzer ausschließlich Informationen aus der Datenbank abrufen kann. Sie ist die Sprache zwischen Datenbank und Benutzer. Die Sprache zwischen Anwendung, Administrator und Datenbank für Manipulationen wie Einfügen und Löschen von Daten liefert ein weiterer Teil der DML. Basierend auf dieser Grundlage werden im Folgenden »Anfragen« und »Abfragen« u. U. anders als in der Literatur als unterschiedliche Termini verwendet werden. »Abfrage« bezeichnet die gezielte benutzerdefinierte Extraktion von Daten durch die Verwendung einer QL aus der Datenbasis, während »Anfrage« den direkten Zugriff des DBMS auf die Datenbasis bezeichnet, die mithilfe eines weiteren Teils der DML formuliert wird.

Aufgabe des DBMS ist es außerdem, die jeweilige Sicht auf die Daten entsprechend der Benutzergruppe zu gewährleisten. Mit unterschiedlichen Gruppen an Nutzern kommen auch unterschiedliche Zugriffsrechte zum Tragen, die mithilfe der unterschiedlichen Sichten ebenfalls abgedeckt und vom DBMS gesteuert werden. Dem DBMS unterliegt weiterhin die Verwaltung der Datenbankschemata und der Daten selbst sowie die Datenintegrität. Somit müssen die Daten den Integritätsbedingungen der Datenbank entsprechen. Integritätsbedingungen können sein, dass eine Instanz nur Werte in einem bestimmten Wertebereich annehmen kann oder modellnäher, dass jede Relation ein Schlüsselattribut hat oder dass alle in Beziehung stehenden Tupel vorhanden sein müssen.

Das DBMS stellt Operationen zur Verfügung, um vom externen zum und vom konzeptuellen auf das interne Schema zu transformieren, um die gespeicherten Daten zu verarbeiten. Umformatierungen der Daten in die externen Sichten ist eine Ab-

bildung (*Mapping*) zwischen den unterschiedlichen Sichten. Das DBMS muss Datenunabhängigkeit gewährleisten, so dass Änderungen auf einer Ebene vorgenommen werden können, ohne dass es Änderungen in einer anderen Ebene bedarf. Nur das *Mapping* zwischen den Ebenen sollte sich durch das DBMS automatisch ändern.

2.2. Die Standardsprache relationaler Datenbanken - SQL

In Kapitel 8 werden Etikettierungen aus einer EMU-Datenbank in eine relationale Datenbank überführt und abgefragt. Hierfür wird die Sprache MySQL (Reese, Yarger und King, 2002) verwendet, welcher SQL-der Standard (Date und Darwen, 1998) zugrunde liegt. Daher soll dieses Kapitel die für das Verständnis benötigten Formalismen der Sprache darstellen. Die Formalismen sind festgeschrieben und in Einführungsbüchern oder reinen SQL-Beschreibungen wie in Date und Darwen (1998) oder Elmasri und Navathe (2009) aufbereitet. Diese Quellen wurden auch für die Darstellung hier verwendet. Die Umsetzungen in der exakten Zeichensetzung sind vom verwendeten Datenbanksystem abhängig. Im Folgenden werden Umsetzungen ggf. nur für MySQL dargestellt.

Die 'Structured Query Language' (SQL) hat sich zum relationalen Datenbankstandard entwickelt, d. h. sie wird als Standard in kommerziellen relationalen Datenbanken als Datenbanksprache in Datenbankmanagementsystemen verwendet. Nach Elmasri und Navathe (2009) soll SQL maßgeblich für die erfolgreiche Etablierung relationaler Datenbanken verantwortlich sein, da diese Sprache gegenüber der Relationenalgebra, die für Operationen in relationalen Datenbanken verwendet wird, eine benutzerfreundlichere Syntax aufweist. Der Datenbankstandard ist eine deklarative Sprache, die vom Benutzer das Spezifizieren des gewünschten Resultats fordert. Die benötigten Operationen werden dann aufgrund dieser Spezifizierung entsprechend vom System selbst erstellt.

In SQL werden Relationstypen, die in Schemadiagrammen im relationalen Modell als Tabellen dargestellt werden, auch als solche bezeichnet, da sie nicht streng genommen als Relationen behandelt werden. Die Tabelle selbst stellt also den Relationstypen dar, die Spalten Attribute und eine Zeile ein relationales Tupel. Da die Begriffe Tabelle, Spalte und Zeile wirklich ersetzend in SQL verwendet werden, wird, wie auch in der Literatur üblich, im Folgenden relationale Terminologie nicht von SQL-Terminologie streng unterschieden.

SQL kann sowohl als DDL als auch als DML (vgl. S. 40ff.) eingesetzt werden. Das heißt, in kommerziellen relationalen Datenbanksystemen werden beide Datenbanksprachen über SQL realisiert. Als umfassende Sprache kann sie für Datendefinitionen, Anfragen, Abfragen als auch das Definieren von Sichten (vgl. S. 29), von Inte-

gritätsbestimmungen und Datensicherheitsaspekten verwendet werden. Im Standard sind bereits Schnittstellen zu Programmiersprachen beschrieben und mit Regeln verknüpft.

Ein Beispiel für eine Anweisung, die mit einer DDL ausgeführt würde, ist der SQL-Befehl zum Erstellen eines Datenbankschemas[5] in SQL-Anweisung 1. Das explizite Erzeugen eines Datenbankschemas wird notwendig, sobald mehrere Datenbankschemata zum Beispiel im Datenbankkatalog verwaltet werden müssen. In diesem Fall muss eine **use** Anweisung erfolgen, über die das Datenbankmanagementsystem die zu bearbeitende Datenbank auswählt.

SQL-Anweisung 1. CREATE SCHEMA `Geldinstitut`

SQL-Anweisung 2. CREATE DATABASE `Geldinstitut`

In MySQL wird die Anweisung leicht abgeändert über die SQL-Anweisung 2 formuliert. Eine **DROP SCHEMA** bzw. **DROP DATABASE** Anweisung löscht Schemata oder Datenbanken. Datenbankzustände können in die Datenbank über verschiedene Operationen, die jedoch datenbanksystemabhängig sind, vollständig aus z. B. Dateien eingelesen werden. In Kapitel 8 wird ein Beispiel für das Laden des Datenbestandes für den Datenbankzustand gezeigt werden.

Eine weitere typische DDL Anweisung ist je nach Modell das Erstellen von Relationstypen. **CREATE TABLE** SQL-Anweisungen, die den Tabellennamen als Parameter mit sich führen, erstellen die entsprechenden SQL-Tabellen. SQL-Anweisung 3 erstellt den Relationstyp **KUNDE** in Schema 1 aus Kapitel 2.1.2 (S. 39). Mithilfe dieser Anweisung wird die Tabelle erzeugt, die Attribute gesetzt und auch ein Datentyp der Attributwerte angegeben. An dieser Stelle ist die Eigenschaft als logisches Schema zu sehen. Weiterhin werden die Schlüsselattribute als solche deklariert (vgl. `PRIMARY KEY` in SQL-Anweisung 3).

Der Datenbankzustand ist nach dem Erzeugen der Tabellen leer. Um ihn zu füllen, werden Informationen über **INSERT** Anweisungen eingepflegt. So wird mit SQL-Anweisung 4 der erste Kunde aus Schema 1 auf S. 39 in die Datenbasis eingefügt. **INSERT INTO**[6] wählt hierbei den Relationstypen also die Tabelle aus und **VALUE** beschreibt das einzupflegende Tupel.

[5] vgl. S. 32ff.

[6] Die Angabe der Attributnamen ist nur obligatorisch, wenn einem Attribut kein Wert aus dem Tupel zugewiesen werden soll oder kann.

SQL-Anweisung 3.

```
CREATE TABLE KUNDE (
        Kundennr    INT
        Anschrift   VARCHAR(30)
        Name        VARCHAR(30)
)
PRIMARY KEY (Kundennr)
```

SQL-Anweisung 4.

```
INSERT INTO KUNDE (Kundennr, Anschrift, Name)
VALUES ('0001','Kiel','Hanson')
```

Abfragen sind über einen so genannten *select-from-where* Block formuliert, in dem das erwartete Ergebnis spezifiziert ist. Attribute, deren Werte in Erfahrung gebracht werden wollen, sind in einer **SELECT**[7] Anweisung wie in SQ 1 aufgelistet.

SQ 1.

```
SELECT Anschrift
```

SQ 2.

```
FROM KUNDE
WHERE Name='Hanson'
```

In **FROM** in SQ 2 ist die Tabelle angegeben, die das gesuchte Attribut enthält. Bedingungen in Form von booleschen (bedingten) oder regulären Ausdrücken schränken die Werte in der Attributsspalte auf gesuchte Tupel ein. Tupel, die der Abfrage entsprechen, müssen diese Bedingungen in den Werten erfüllen. Derartige Einschränkungen stehen in einer **WHERE** Anweisung (vgl. SQ 2). Die benötigte Tabelle in der **FROM** Anweisung ist unweigerlich von den Anweisungen in **WHERE** und **SELECT** abhängig, da nur Attribute aus dieser Tabelle eingeschränkt und ausgewählt werden können.

In SQ 2 werden die Tupel gesucht, die als Attributwert in der Name-Spalte der SQL-Tabelle KUNDE das Wort <Hanson> haben. Die Spalten sind explizit mit Namen angegeben, da die Werte innerhalb der Tupel eigentlich keine feste Ordnung haben.

Als Vergleichsoperatoren zwischen Spaltenname und gewünschtem Attributwert stehen der Gleichheits-, Ungleichheitoperator und alle gemeinhin bekannten Vergleichsoperatoren ($=, <, >, \neq$). Der Operator richtet sich nicht nur nach der gewünschten Bedeutung des Ausdrucks sondern auch nach dem Datentyp des Attributs, der im Datenbankschema deklariert ist. So kann eine Zeichenkette mit dem Gleichheits-

[7] In der Relationenalgebra wird das entgültige Ergebnis der Abfrage, das wieder eine Relation darstellt, aufgrund dieser Anweisung auf die angegebenen Attribute, von denen die Information gewünscht ist, projiziert: $\pi_{Attribut}(R)$ mit R als Ergebnisrelation.

und Ungleichheitsoperator verbunden werden, jedoch nicht mit Kleiner- oder Größeroperatoren. Logischen Operatoren (z. B. `and` für UND Verknüpfungen) können Bedingungen verschiedener Attribute verbinden, sodass mehrere Attribute eingeschränkt sind.

Reguläre Ausdrücke (The Open Group, 2003) als Alternative zum booleschen Ausdruck in **WHERE** Anweisungen, stellen Schablonen dar, in die verschiedene Attributwerte fallen. Oder aus einer anderen Sichtweise, können reguläre Ausdrücke als Attributwertfilter angesehen werden. Die **WHERE** Anweisung in SQ 2 würde mit einem regulären Ausdruck wie in SQ 3 den Namen <Hanson> sowie <Hannson> suchen. Ein Ausdruck kann somit auf mehr als einen expliziten Wert passen.

SQ 3. **WHERE** `Name REGEXP 'Ha(n){1,2}son'`

Gehören die auszuwählenden Attribute nicht zu einem Relationstypen, d. h. sie stehen nicht gemeinsam in einer Tabelle, enthält die **FROM** Anweisung mehrere Tabellen[8]. Gibt es in den ausgewählten Tabellen identische Attribute und somit modellbedingt identische Attributwerte, so werden die Tabellen anhand dieses Attributs bzw. der Attributwerte miteinander verbunden (Natural Join). Das Schema und die Ausprägung, die durch einen Natural Join der beiden Tabellen in Ausprägung 1 (vgl. S. 39) entstehen, sind in folgender SQL-Tabelle 1 abgebildet. Die Tabelle setzt somit das beispielhafte **KUNDENKONTO** in Schema 2 um. In den anderen Fällen gestaltet sich der Join komplexer, indem für alle Tupel der einen Tabelle alle Tupel der anderen Tabelle kombiniert werden. Auf die beschriebene Weise werden Attributwerte aus verschiedenen Tabellen in neue Relationen gebracht. Es entstehen neue Datenbanktupel.

SQL-Tabelle 1.

R

Kundennr	Anschrift	Name	Kontonr	Stand
0001	Kiel	Hanson	23417	1000
0002	Hamburg	Meyer	56767	1300
0003	München	Tegel	97823	800
0004	Hamburg	Schmidt	68945	700

SQ 4 zeigt ein komplexeres Beispiel mit zwei Tabellen, die für die Abfrage benötigt werden. Abgefragt werden *Kontonummer und Name von Kunden aus Hamburg, die weniger als 1000 Euro auf ihrem Konto haben.* Ein konstruierter Kontext einer

[8] Für diese Tabellen erfolgt in relationaler Sicht eine Join Operation ($R' = R_1 \bowtie R_2$), die die Tabellen zu einer Tabelle zusammenfasst.

solchen Abfrage wäre, dass den betreffenden Kundenkonten mehr Kontoführungsgebühren abgezogen werden sollen, um den Kunden den Umzug in ein anderes Institut schmackhaft zu machen, da das Institut im Bereich Hamburg zu wenig Kapital zum Wirtschaften erhält. Durch die **FROM** Anweisung werden die Tabellen aus Ausprägung 1 (S. 39) über einen Natural Join verbunden und die SQL-Tabelle 1 wird für die weitere Suche verwendet. Die Bedingungen der Attribute aus der **WHERE** Anweisung werden in der Tabelle überprüft. Alle Tupel, die alle Bedingungen erfüllen, werden in einer weiteren SQL-Tabelle verarbeitet. Die der Abfrage entsprechenden Tupel sind in SQL-Tabelle 2 dargestellt. Zuletzt stellt die SQL-Tabelle 3 die Rückgabewerte der Abfrage nach der Projektion auf die in der **SELECT** Anweisung deklarierten Attribute dar.

SQ 4.

```
SELECT Kundennr, Name
FROM KUNDE, KONTO
WHERE Anschrift='Hamburg' and Stand < 1000
```

SQL-Tabelle 2.

FUND

Kundennr	Anschrift	Name	Kontonr	Stand
0004	Hamburg	Schmidt	68945	700

SQL-Tabelle 3.

RÜCKGABE

Kundennr	Name
0004	Schmidt

Dieser kleine Einblick in die 'Structured Query Language', die als Datenbankstandard relationaler Datenbanken gilt, zeigt eine einfache Grundstruktur der Sprache – der Grund für den Erfolg in der Datenbanktechnologie. Aufgrund der relationalen Tabellen sind die Einschränkungen der zu suchenden Tupel einfach über die Festlegung ihrer Attributwerte gegeben. Bei dieser Einfachheit müssen dennoch die in der Suche zu verwendenden SQL-Tabellen gewählt werden. Abfragen können mit den hier vorgestellten Formalismen einen hohen Grad an Komplexität erreichen, wenn Attribute aus sehr vielen verschiedenen Tabellen in Beziehung gesetzt werden müssen, um abfrageäquivalente Tupel finden zu können. Tabellen im Datenbankschema sollten entsprechend benötigter Abfragen angelegt sein, um unnötig komplizierte Ausdrücke zu vermeiden, die diese Tabellen miteinander in Verbindung bringen. Die hier gegebenen Beispiele haben kein hohes Maß an Komplexität. Eine komplexere Abfrage ist als Beispiel im Anhang C.5[9] (S. 271) dargestellt.

[9] In Kapitel 8 wird auf die Abfrage eingegangen.

2.3. Erweiterte Backus-Naur Form – eine formale Sprache

In natürlichen Sprachen werden mithilfe des Alphabets Wörter gebildet, die unter bestimmten grammatischen Bedingungen (Syntax) miteinander kombiniert werden können, um eine Bedeutung (Semantik) auszudrücken. Die Bedeutung selbst ist unter Umständen durch das Vorwissen des Gesprächspartners, den Kontext und andere Einflüsse vielfältig interpretierbar. Hinzu kommt, dass auch syntaktisch inkorrekte Ausdrücke den gewünschten Inhalt vermitteln können.

Genauso wie natürliche Sprachen (Deutsch, Englisch, ...) definieren sich formale Sprachen über ihre möglichen korrekten Ausdrücke. In formalen Sprachen werden über dem Alphabet Zeichenketten erzeugt, die wiederum zu längeren Zeichenketten miteinander kombiniert sein können. Die Korrektheit des Ausdruckes ist über die Syntax der Sprache definierbar. Syntaktisch richtige Ausdrücke, die nur aus Zeichen bestehen, die das Alphabet der Sprache anbietet, haben eine nicht kontextabhängige exakte Semantik. Diese Einschränkung gegenüber natürlichen Sprachen bietet in der EDV den Vorteil der leichteren Verarbeitung (Hedtstück, 2007). Formale Sprachen sind u. a. Datendefinitionssprachen, wie die DDL im EMU-System, die in Kapitel 3.2.3 formal beschrieben wird und An- bzw. Abfragesprachen, wie die EMU Query Language, deren formale Beschreibung in Kapitel 3.3 erfolgt.

Für die Beschreibung formaler Sprachen haben sich unterschiedliche Notationsmöglichkeiten entwickelt. Zu nennen sind hier u. a. die erweiterte Backus-Naur Form (EBNF) (ISO/IEC 14977, 1996) sowie die Verwendung von Syntaxdiagrammen. Da in Kapitel 3.3 für die Beschreibung der EMU Query Language nur EBNF verwendet wird, wird diese im Folgenden kurz vorgestellt.[10] Der EBNF Standard (ISO/IEC 14977, 1996) als Erweiterung der Backus-Naur Form ist als vorgeschlagene Notation zu sehen, sodass abweichende Verwendungen möglich sind. Informationen über Syntaxdiagramme können aus Hedtstück (2007, S. 45-48)[11] und anderen Einführungsbüchern entnommen werden.

Nach ISO/IEC 14977 enthält eine formale Syntaxdefinition eine Aufzählung der verschiedenen syntaktischen Teile, zeigt die Symbolsequenzen auf, die einen wohl-

[10] Direkt an dieser Stelle sei angemerkt, es ist auffällig, dass viele Autoren für die Beschreibung einer formalen Sprache die Backus-Naur Form verwenden, wobei sie angeben, die EBNF zu verwenden.

[11] verwendet jedoch nicht die EBNF nach ISO/IEC 14977 (1996)

geformten Ausdruck bilden und bildet die syntaktische Struktur jedes möglichen Ausdrucks in der Sprache ab. Hierfür werden terminale und nicht terminale Symbole unterschieden. Ein terminales Symbol ist eine Sequenz bestehend aus einem oder mehreren Zeichen ("character" ISO/IEC 14977 (1996, S. 1)), die nicht weiter zerlegbar ist. Terminale Symbole sind mit Morphen in der natürlichen Sprache vergleichbar. Nicht terminale Symbole hingegen sind definierte Teile der Sprache, die durch syntaktische Verbindungen entstehen. Sie werden über einen *Meta-Identifier* benannt.

Die Tabellen 2.1 und 2.2 listen die terminalen Symbole[12] der EBNF auf, die eine andere Semantik haben, als die Zeichen an sich. Hierbei werden Operatoren und Klammern gemäß ISO/IEC 14977 (1996) unterschieden. Nach ISO/IEC 14977 (1996, S. 1ff.) repräsentiert das Zeichen an sich den Operator der EBNF. Derartige Terminale binden unterschiedlich stark ihren Kontext, vergleichbar mit der ‚Punkt vor Strich'-Regel in der Mathematik, wobei Klammerpaare eine höhere Bindung haben als Operatoren. Andere terminale Symbole wie Buchstaben repräsentieren sich selbst (vgl. ISO/IEC 14977 (1996, S. 6)).

Die Syntax der zu beschreibenden formalen Sprache wird in EBNF durch Produktionsregeln formuliert. Derartige Produktionsregeln sind dem Linguisten aus der generativen Grammatik bekannt, denn sie sind mit den Produktionsregeln in Chomsky (1957) vergleichbar, wie in der Phrasenstrukturbeschreibung in Abbildung 2.4 dargestellt ist.

(13) (i) $Sentence \rightarrow NP + VP$
(ii) $NP \rightarrow T + N$
(iii) $VP \rightarrow Verb + NP$
(iv) $T \rightarrow the$
(v) $N \rightarrow man, ball,$ etc.
(vi) $Verb \rightarrow hit, took,$ etc.

Abbildung 2.4.: Phrasenstrukturbeschreibung aus Chomsky(1957, S. 26).

In EBNF wird durch Produktionsregeln einem *Meta-Identifier* eine Definitionsliste zugewiesen (=). Definitionslisten selbst bestehen aus geordneten Einzeldefinitionen (getrennt durch |), die wiederum eine Liste syntaktischer Terme enthalten (getrennt

[12] In der formalen Beschreibung sind diese Symbole keine Einzelzeichen, sondern Wörter, die auf -symbol enden, sodass ein Symbol formal genau betrachtet nur das eigentliche Terminal repräsentiert. Auf diese formale Unterscheidung soll hier jedoch verzichtet werden.

Tabelle 2.1.: Terminale Symbole (Klammern) der EBNF und ihre Bedeutung nach Prioritätsstufen geordnet (abnehmend).

Symbol	Verwendung
`'`	beginnt und beendet Terminal außer `"`
`"`	beginnt und beendet Terminal außer `'`
`(*`	beginnt Kommentar
`*)`	beendet Kommentar
`(`	beginnt eine gruppierte Definitionsliste
`)`	beendet eine gruppierte Definitionsliste
`[`	beginnt eine optionale Definitionsliste
`]`	beendet eine optionale Definitionsliste
`{`	beginnt eine wiederholbare Definitionsliste
`}`	beendet eine wiederholbare Definitionsliste

Tabelle 2.2.: Terminale Symbole (Operatoren) der EBNF und ihre Bedeutung nach Prioritätsstufen geordnet (abnehmend).

Symbol	Verwendung	Bedeutung als Operator
`*`	natürliche Zahl `*` Grundelement	wiederholt das folgende syntaktische Grundelement so oft, wie zuvor als ganze Zahl angegeben
`-`	Faktor `-` Faktor (Ausnahme)	schließt den folgenden syntaktischen Faktor aus
`,`	Term `,` Term	verbindet syntaktische Terme
`|`	Einzeldefinition `|` Einzeldefinition	zeigt Alternativen
`=`	*Meta-Identifier* `=` Definitionsliste	definiert
`;`	beendet eine syntaktische Regel	terminiert

Tabelle 2.3.: Syntaktische Grundelemente der EBNF mit Angabe eines Beispiels.

Grundelement	Beispiel
optionale Definitionslisten	`['A']`
wiederholbare Definitionslisten	`{'A'}`
gruppierte Definitionslisten	`('A'|'B') ('A','B')`
Meta-Identifier	`AZaz09`
Terminal-Zeichenkette	`'A' "'" '"'`
spezielle Sequenzen	`?`arbiträre Zeichen`?`
leere Sequenzen	

durch ,). Syntaktische Terme sind syntaktische Faktoren mit oder ohne syntaktischen Ausnahmen, wobei die syntaktischen Faktoren syntaktische Grundelemente der Sprache entweder mit oder ohne Wiederholung sind. Die syntaktischen Grundelemente der EBNF sind in Tabelle 2.3 aufgelistet.

Das Grundelement Terminal-Zeichenkette wird in der Beschreibung formaler Sprachen verwendet, um die Terminale der zu beschreibenden Sprache von den Terminalen der Beschreibungssprache (hier EBNF) unterscheiden zu können. FB 1 zeigt ein konstruiertes Beispiel aus der deutschen Orthographie. Ein möglicher Ausdruck ist in FB-B 1 angegeben.

FB 1 (Komma).

```
(* HAUPTSATZ - intuitiv *)
(* RELATIVSATZ - intuitiv *)
SATZ = HAUPTSATZ,",","," ",RELATIVSATZ,".";
```

FB-B 1 (Kommabsp.). *Ich warte auf den Bus, der zum Bahnhof fährt.*

Abbildung 2.5 zeigt EBNF-Produktionsregeln für wohlgeformte Ausdrücke. Im ersten Beispiel ist die Terminal-Zeichenkette `"A"` das syntaktische Grundelement, das gleichzeitig syntaktischer Faktor, Term, Einzeldefinition und Definitionsliste ist, welches über das Gleichheitszeichen dem *Meta-Identifier* `aa` zugeordnet ist. Der *Meta-Identifier* enthält daher nur den Buchstaben[13] `A`.

```
aa = "A";                          aa:  A
bb = 3 * aa, "B";                  bb:  AAAB
cc = 3 * [aa], "C";                cc:  C   AC   AAC   AAAC
dd = {aa}, "D";                    dd:  D   AD   AAD   AAAD   AAAAD   etc.
ee = aa, {aa}, "E";                ee:  AE   AAE   AAAE   AAAAE   AAAAAE   etc.
ff = 3 * aa, 3 * [aa], "F";        ff:  AAAF   AAAAF   AAAAAF   AAAAAAF
```

Abbildung 2.5.: Beispiele von EBNF-Produktionsregeln (links) und die dadurch erzeugbaren Ausdrücke (rechts) übernommen aus ISO/IEC 14977 (1996, S. 4).

Im zweiten Beispiel besteht eine Einzeldefinition aus der Verbindung (Konkatenation) zweier syntaktischer Terme (durch Komma getrennt), wobei für den zweiten Term wiederum gilt: er ist gleichzeitig syntaktisches Grundelement und syntaktischer Faktor. Im ersten Term ist das Grundelement (`aa`-*Meta-Identifier*) durch die Angabe der dreifachen Wiederholung (`3 *` Grundelement) verschieden vom syntaktischen Faktor. Da der *Meta-Identifier* `aa` nur symbolhaft für ein `A` stehen kann, kann der *Meta-Identifier* `bb` nur für die Zeichenkette `AAAB` stehen. Das ergibt sich

[13] vgl. oben: Buchstaben repräsentieren sich selbst

aus den beiden Termen: der dreifachen Wiederholung des `A` im *Meta-Identifier* `aa` wird der Term, der nur ein `B` produzieren kann, angehangen.

Im dritten Beispiel bildet der *Meta-Identifier* `aa` die Definitionsliste innerhalb des syntaktischen Grundelementes (die optionale Definitionsliste). Zusammen mit der Wiederholungsangabe bildet sich der syntaktische Faktor, der gleichzeitig syntaktischer Term ist. Dieser und der zweite Term zusammen bilden eine Einzeldefinition und gleichzeitig Definitionsliste. Der erste Term drückt aus, dass das `A` dreimal hintereinander entweder vorhanden oder nicht vorhanden ist und von einem `C` gefolgt wird.

Für alle anderen Beispiele in Abbildung 2.5 gilt entsprechendes bezüglich der syntaktischen Elemente. In Bezug auf die möglichen Ausdrücke wird in Beispiel 4 im ersten Term angegeben, dass `A` beliebig oft vorkommen kann und dann gefolgt wird von einem `D`. Die letzten beiden Beispiele lassen sich entsprechend der erläuterten Beispiele erklären.

```
letter = "A" | "B" | "C" | "D" | "E" | "F"
  | "G" | "H" | "I" | "J" | "K" | "L" | "M"
  | "N" | "O" | "P" | "Q" | "R" | "S" | "T"
  | "U" | "V" | "W" | "X" | "Y" | "Z";
vowel = "A" | "E" | "I" | "O" | "U";
consonant = letter - vowel;
ee = {"A"}-, "E";
```

```
letter:     A  B  C  D  E  F  G  H  I  J  etc.

vowel:      A  E  I  O  U
consonant:   B  C  D  F  G  H  J  K  L  M  etc.
ee:         AE  AAE  AAAE  AAAAE  AAAAAE  etc.
```

Abbildung 2.6.: Weitere Beispiele von EBNF-Produktionsregeln (links) und die dadurch erzeugbaren Ausdrücke (rechts) übernommen aus ISO/IEC 14977 (1996, S. 5).

Abbildung 2.6 zeigt noch ein weiteres Beispiel, in dem die Einzeldefinitionen nicht gleichzeitig Definitionslisten und syntaktische Faktoren nicht unverändert auch Terme sind. So zeigt sich für den *Meta-Identifier* `letter` und auch `vowel` die Definition einer Definitionsliste, die ausdrückt, dass nur eines der angegebenen Grundelemente, die gleichzeitig Faktor, Term und Einzeldefinition sind, im produzierten Ausdruck auftauchen darf. In `consonant` wird ein einziger Term angegeben (gleichzeitig Einzeldefinition und Definitionsliste), der aus einen Faktor, dem terminalen Symbol (`-`) und der syntaktischen Ausnahme besteht. Dieser Term drückt aus, dass alle Zeichen (hier Buchstaben), die der *Meta-Identifier* `letter` produzieren kann, als Ausdruck zulässig sind, solange sie nicht auch ein mögliches Zeichen (Vokalbuchstaben) enthalten, die der *Meta-Identifier* `vowel` kodiert.

Der erste Term für die Produktionsregel des *Meta-Identifiers* `ee` drückt die Möglichkeit aus, `A` beliebig oft im Ausdruck zu haben, wobei durch die syntaktische Einschränkung (`-`) das nicht Vorkommen dieses Symbols ausgeschlossen wird. In der Verbindung mit dem zweiten Term, können also nur Ausdrücke mit einem oder mehreren `A`s gefolgt von einem `E` produziert werden.

An dieser Stelle soll der Einblick in formale Sprachen nicht weiter vertieft werden, da in der gegebenen Übersicht der erweiterten Backus-Naur Form alle in den Kapitel 3.2.3 und 3.3 verwendeten Konventionen der EBNF für das Verständnis der Kapitel dargestellt sind. Es sei jedoch an dieser Stelle angemerkt, dass die formale Beschreibung einer Sprache im Grad der Formalisierung an den Nutzen der Beschreibung angepasst werden kann. So werden in ISO/IEC 14977 (1996, S. 8ff.) 3 verschiedene formale Beschreibungen der EBNF formuliert, die auf die formale Art alle verwendeten *Meta-Identifier* formal beschreibt, während die informellere Beschreibung auf die formale Darstellung von Buchstaben und Zahlen verzichtet und die Bedeutung der *Meta-Identifier* in Kommentaren erläutert. Diese informellere Art der formalen Beschreibung wird in dieser Arbeit für den Zweck als angemessen erachtet und verwendet.

3. EMU Speech Database System

Für die Weiterentwicklung der Systems war es notwendig, die Etikettierungsstrukturen, die das System anbietet, genau zu kennen und für die Konzeptionierung der Umsetzung der Interoperabilität eine Notationsmöglichkeit der Etikettierungsstrukturen als Werkzeug zur Hand zu haben. Die Kenntnisse über die möglichen Strukturen und Notationen sind auch für den Einsatz in der Lehre sowie in der Forschung für den Benutzer wichtig, um eigene Sprachdatenbanken zu konzipieren und aufzubauen.

Für die Weiterentwicklungen im Bereich der Nutzbarkeit des datenbankbasierten Ansatzes im Sinne der benutzerfreundlichen Anwendung sowie um alle gegebenen Möglichkeiten für weitere Funktionalität ausnutzen zu können, war ein gutes Verständnis des Ansatzes selbst sowie der Umsetzung notwendig. Auch hierfür bot sich eine grafisch modellhafte Aufbereitung an.

Formale Beschreibungen und Dokumentationen, die detailreich genug sind, standen vor dieser Dissertation nicht zur Verfügung. Diese Arbeitsgrundlage wurde mithilfe der Literatur und dem EMU-Quellcode eruiert und wird im Folgenden vorgestellt. Hiermit soll gleichzeitig das Konzept des Systems und dessen Funktionsweise aufgezeigt werden.

3.1. Etikettierungsstrukturen

Die Etikettierungsstrukturen im EMU-System werden durch das EMU-Datenmodell vorgegeben. Das Modell erhebt den Anspruch, die Anforderungen der Nutzer, in erster Linie die Ansprüche der Phonetiker, zu erfüllen. Dazu gehören das Etikettieren auf verschiedenen Etikettierungsebenen, die Verbindung zwischen Etikett und Zeitsignal als auch die Assoziation[1] zwischen verschiedenen Etiketten. Im Folgenden wird das EMU-Etikettierungsmodell, wie es in Cassidy und Harrington (2001) aus einem anderen Blickwinkel und in Harrington (2010a) beschrieben wird, vorgestellt. Gefolgt wird dies von Notationsvorschlägen für Schemadiagramme, die zwar bereits in Harrington (2010a) das erste Mal Verwendung in der Literatur fanden, vorher auch in der Lehre erfolgreich eingesetzt worden sind, aber im Rahmen der Dissertation gemeinsam diskutiert worden waren. Sowohl Modellbeschreibungen als auch die Notationsvorschläge werden anders als in der Literatur als Ganzes und im Detail dargestellt. Dies weniger, um das System durch das mögliche Verständnis benutzerfreundlicher zu machen, sondern um auch Programmierern die zukünftige Weiterentwicklung zu erleichtern.

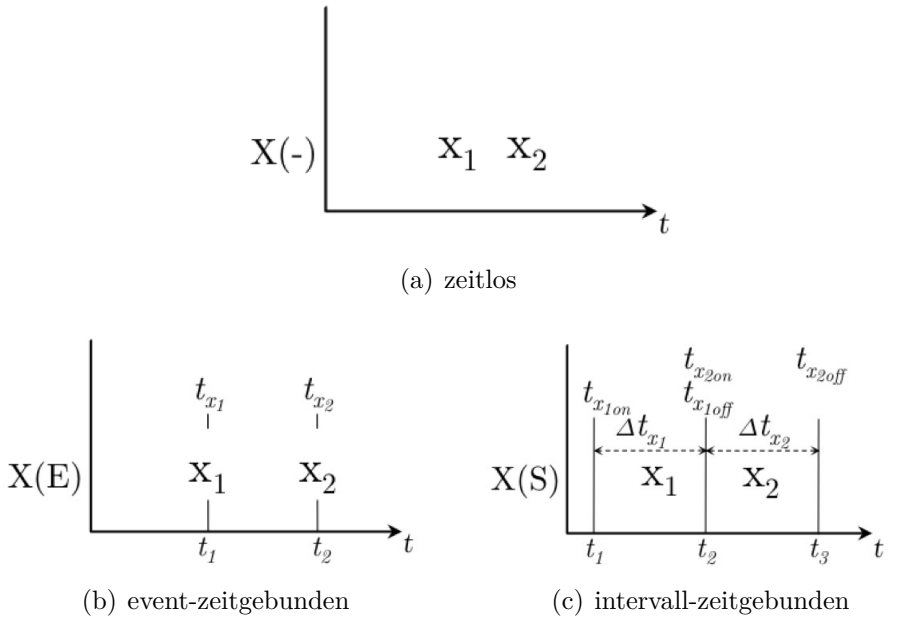
Zum Abschluss wird in Kapitel 3.1.3 die Relevanz der Strukturen anhand von Beispielen aus der eigenen phonetischen Forschung erläutert. Vorweggenommen sei an dieser Stelle, dass die Anforderungen an die Datenbankabfrage und die Eigenschaften der Abfragesprache auch in den Etikettierungsstrukturen berücksichtigt werden. Außerdem müssen sie bei der Verwendung der Etikettierungsmöglichkeiten während des Entwurfs des Etikettierungsschemas für die Datenbank immer berücksichtigt werden. Insofern ergibt sich die Relevanz verschiedener Strukturen u. U. erst in der Datenbankabfrage durch die EMU Query Language, die in Kapitel 3.3 vorgestellt wird.

3.1.1. Modell

Wie in Kapitel 2.1.2 beschrieben wurde, stellt ein Modell Objekte, deren Attribute sowie Operationen auf Objekten bereit. Als Objekte bietet das EMU-Modell Eti-

[1] In EMU-Terminologie sind Ebenen auf unterschiedliche Art miteinander assoziiert. Sie stehen in einer bestimmten Beziehung zueinander. Gleiches gilt terminologisch für die entsprechenden Etiketten (auch kontextabhängig: Segmente, Tokens genannt). Ebenen und Tokens auf Ebenen stehen in einer Relation zu einander. <Relation> ist in der Datenbanktechnologie ein Terminus Technicus, somit wird diese Bezeichnung weitgehend vermieden.

kettierungsebenen an. Wie in Cassidy und Harrington (2001) beschrieben, müssen Ebenen eines der Attribute zeitlos, intervall-zeitgebunden oder event-zeitgebunden haben. Etikettierungsebenen enthalten später die Etiketten, die als Instanz der Ebene angesehen werden können[2].



(a) zeitlos

(b) event-zeitgebunden

(c) intervall-zeitgebunden

Abbildung 3.1.: Darstellung der drei verschiedenen Arten der Zeitgebundenheit in (a–c) der Ebenen im EMU-Etikettierungsmodell und der damit einhergehenden Eigenschaften der Tokens auf den Ebenen. Hierzu gehören die Zeitmarken (t_{xn}), die Anfangszeitmarken (t_{xnon}) und Endzeitmarken (t_{xnoff}) als auch die Dauern (Δt_{xn}), die sich aus den Zeitmarken für die Tokens ergeben.

Die Beziehungen der Instanzen zueinander auf den verschiedenen Ebenen sind somit die gleichen wie die der Ebenen selbst. Instanzen einer Ebene, die im Folgenden »Tokens« genannt werden, stehen in einer sequentiellen Anordnung (sequentiellen Beziehung), so dass jede Instanz keinen oder einen Vorgänger und Nachfolger hat. In Abbildung 3.1 werden Tokens auf einer zeitlosen Ebene, event-zeitgebundenen und intervall-zeitgebundenen Ebene dargestellt. In allen drei Fällen ist die sequentielle Anordnung der Tokens zu erkennen. Während für die zeitlosen Tokens keinerlei Bezug zur Zeitlinie gegeben ist, haben die event-zeitgebundenen Tokens diesen Bezug und sie haben eine Zeitmarke (t_{xn}). Zwei Zeitmarken (t_{xnon} und t_{xnoff}) erhält ein Token auf intervall-zeitgebundenen Ebenen und hat somit im Gegensatz zu Tokens

[2]In Kapitel 3.2 wird gezeigt werden, dass Ebenen in der Modellierung nur Container für die Etiketten darstellen. Die eigentlichen Objekte sind die Etiketten, die im logischen Modell modelliert sind und als Segmente oder Tokens bezeichnet werden.

in den anderen beiden Fällen eine Dauer (Δt_{x_n}). Etiketten der Tokens sind in der derzeitigen Implementierung hinsichtlich der Länge auf 70 Bit beschränkt[3]. Somit kann ein gesamter größerer Text kein alleiniges Etikett bilden, längere Textabschnitte müssen somit in kleinere Zeichenketten zerlegt werden.

Neben dem Vorhandensein von Objekten, wie Ebenen und Tokens auf Ebenen können zwischen Objekten unterschiedliche Verbindungen bestehen. Mehrere Ebenen können zu »Ketten« zusammengefügt werden wie aus Harrington (2010a) hervorgeht. In der Implementierung des Systems (EMU Developers, 2011a) in den Versionen < EMU 2.3 waren bis zu zwölf Ebenen für die Verkettung erlaubt. Aufgrund neuer Modellierungen an Etikettierungsschemata wurde dieser Wert auf zwanzig[4] angehoben und orientiert sich an der längsten bekannten aber rein fiktiven alle sinnvollen linguistischen und phonetischen Ebenen abdeckenden Kette. Die längste bekannte Kette ist in John und Bombien (iE) modelliert und im Anhang A.1 dargestellt. Das EMU-System unterstützt mehrere solcher Ketten gleichzeitig innerhalb eines Etikettierungsschemas. Abbildung 3.2 zeigt die möglichen Assoziationen zwischen Ebenen auf und deren Auswirkung auf Instanzen der Ebene, die miteinander assoziiert sind, in Abhängigkeit von der Zeitgebundenheit der Ebenen.

In der one-to-one Spalte ist eine lineare Beziehung zwischen den Tokens der Ebenen zu erkennen, d. h. jedes Token der Ebene X ist mit einem y-Token assoziiert. X- und Y-Ebene sind in diesem Fall nicht unabhängig voneinander. Abgesehen vom Etikett haben Tokens auf der X-Ebene die selben Eigenschaften wie Tokens auf der Y-Ebene. Somit teilen sie sich auch das Zeitintervall, wie es in Abbildung 3.2 für die lineare Beziehung gezeigt wird. Eine unterschiedliche Art in der Zeitgebundenheit (event oder segment) der beiden Ebenen ist durch das erwähnte Teilen der Attribute ausgeschlossen. Derartige Ebenen werden in der Literatur »Label-Links«, »Labeltypen« oder im umgangssprachlichen Gebrauch als »Parallelebenen« bezeichnet.

In den Spalten zwei und drei der Abbildung 3.2 sind die möglichen Dominanzbeziehungen dargestellt. Ebene X dominiert die Y-Ebene, somit können x-Tokens sog. »Eltern« von y-Tokens sein, die dadurch sog. »Kinder« der x-Tokens sind. Durch diese Beziehung können Tokens auf der zeitlosen X-Ebene die Zeitmarken der assoziierten Tokens der dominierten Ebene erben, sofern diese zeitgebunden ist. Das

[3] Diese Beschränkung wurde bis heute von keinem Benutzer als Einschränkung gemeldet, insofern bestand im Rahmen der Weiterentwicklung keine Notwendigkeit diese Beschränkung aufzuheben und soll hier nur als Information für eventuelle Weiterentwicklungen gegeben sein.

[4] Die Einschränkung auf die Kettenlänge wurde durch den Benutzer noch nicht als Einschränkung gemeldet, sodass das Wegfallen der Einschränkung in der Weiterentwicklung nie Diskussionsgrundlage fand.

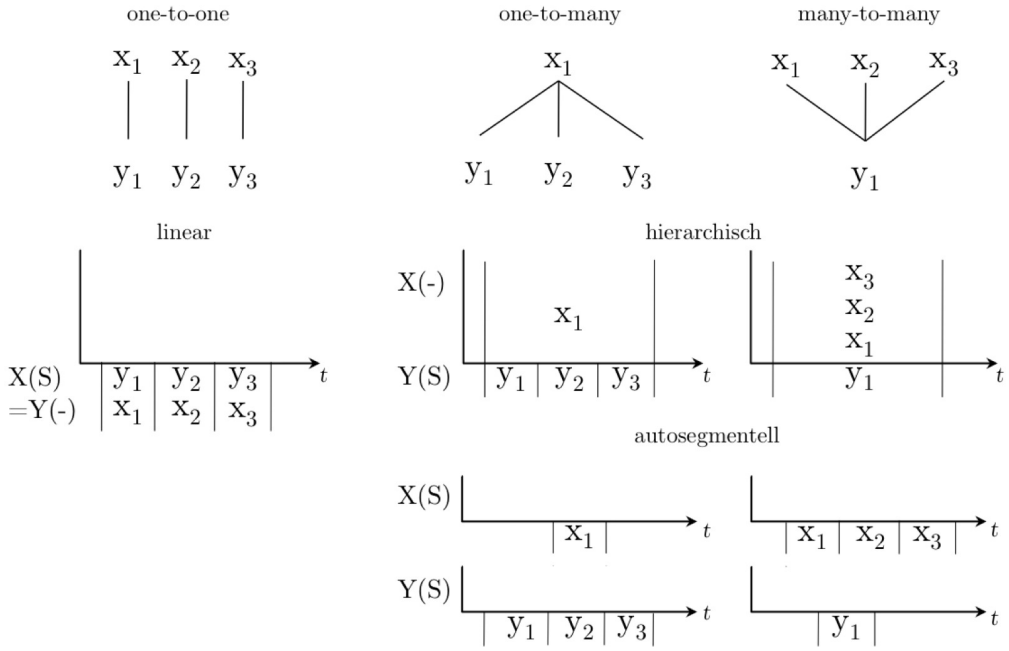


Abbildung 3.2.: Mögliche Assoziationen zwischen Tokens auf Ebenen im EMU-Etikettierungsmodell. Die Spalten unterscheiden die Arten der der Eltern-Kind Beziehungen zwischen den Ebenen und Tokens. In der ersten Reihe ist dabei jeweils eine schematische Darstellung der Beziehungen ohne Berücksichtigung der Zeitgebundenheit der Ebenen dargestellt. Die zweite Reihe zeigt jeweils die Vererbung der Zeitmarken der y-Tokens auf die x-Tokens im Fall einer Zeitgebundenheit der Y-Ebene und Zeitlosigkeit der X-Ebene. Reihe drei zeigt die zeitliche Unabhängigkeit der Tokens, wenn beide Ebenen zeitgebunden sind. Tokens mit abgeleiteten Zeiten sind dabei oberhalb der x-Achse und die Tokens, die die Zeit zur Verfügung stellen, direkt unterhalb der x-Achse dargestellt.

x-Token erhält als Zeitmarken die Startzeit des ersten Tokens, mit dem es assoziiert ist, und die Endzeit des letzten Tokens. Die Dauer des x-Tokens ergibt sich somit aus der Zeitspanne zwischen erstem und letztem assoziierten Token auf der Y-Ebene. In der nicht in Abbildung 3.2 dargestellten hierarchischen Beziehung mit einer eventzeitgebundenen Ebene würde das x-Token nur die Startzeit des ersten Tokens als eigene Startzeit vererbt bekommen. Es hätte somit keine Dauer. Nach Cassidy und Harrington (2001) kann es keine Verbindung zweier zeitgebundener Ebenen mit einer zeitlosen Elternebene geben, da es zur Ambiguität der Zeitenvererbung käme. Das ist nicht der Fall, es kann bzw. es muss immer eine Hauptkette modelliert sein, aus der die Zeiten abgeleitet werden können. Nebenläufige Ketten werden damit in der Vererbung nicht berücksichtigt.

Durch die hierarchische Verknüpfung sind x-Tokens von den in der Hierarchie untergeordneten Tokens auf anderen Ebenen abhängig. Diese Abhängigkeit geht verloren und die Beziehung wird als autosegmentell bezeichnet, d. h. Tokens auf den Ebenen sind zeitlich voneinander unabhängig, wenn beide Ebenen zeitgebunden sind. In diesem Fall dominiert die X-Ebene weiterhin die Y-Ebene, aber es findet keine Vererbung der Zeitattribute statt. Die Relevanz dieser Beziehung wird in Kapitel 3.1.3 demonstriert.

Unabhängig von hierarchischer und autosegmenteller Beziehung zwischen Ebenen sind die one-to-many und many-to-many Beziehungen, die in Abbildung 3.2 die Spalten zwei und drei unterscheiden. Im ersten Fall kann ein Token der dominierenden Ebene mit mehreren Tokens der dominierten Ebene assoziiert werden. Diese Assoziation gleicht einer Baumstruktur. Durch die vom Modell erlaubte many-to-many Beziehung kann jedoch ein Token der dominierten Ebene (Y-Ebene) auch mit mehreren Tokens auf der dominierenden Ebene (X-Ebene) assoziiert werden. Die Baumstruktur gilt im Modell als Standard, welche durch die many-to-many Beziehung erweitert werden kann. Abbildung 3.3 zeigt eine solche Erweiterung und die Konsequenz auf die Vererbung der Zeitinformation in einer hierarchischen Beziehung der beteiligten Ebenen. Für Token x_3 gilt die hierarchische Vererbung der Zeit aus den Tokens y_1 und y_2, während für x_1 und x_2 die zeitliche Information ausschließlich von y_1 geerbt wird.

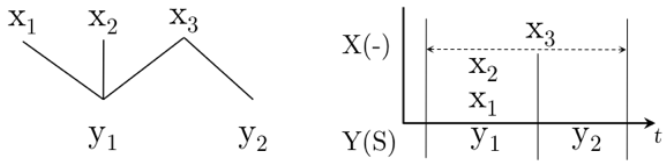


Abbildung 3.3.: Schematische Darstellung einer Baumstruktur mit der Erweiterung um many-to-many Beziehungen (links) und die Konsequenz auf die hierarchische Vererbung der Zeitinformation (rechts).

Selbstverständlich bietet das System auch die Möglichkeit, keine dieser Beziehungen zwischen Ebenen anzuwenden. Tokens auf unterschiedlichen Ebenen können somit auch gänzlich unabhängig sein. Auch eine einzige Ebene ist im EMU-Etikettierungsmodell erlaubt.

In den Beispielen werden keinerlei event-zeitgebundene Ebenen verwendet, da bisher die Relevanz derartiger Ebenen in der phonetischen Forschung nicht für die hierarchische Dominanzbeziehung belegbar ist, also nur für die autosegmentelle Do-

minanzbeziehungen und in linearen Beziehungen eingesetzt wurde. Das Modell ist aber nicht auf diese beiden Assoziationstypen für intervall-zeitgebundene Ebenen beschränkt.

3.1.2. Notation

In Harrington (2010a) wird für die schematische Darstellung von Etikettierungsstrukturen eine übersichtliche Notation ohne Redundanzen verwendet aber nicht formal beschrieben. Für das Konzipieren und Implementieren von Applikationen (z. B. dem Graphischen Template Editor und labConvert) war eine vollständige Notation, d.h. mit Redundanzen, notwendig. Im Folgenden wird die in dieser Dissertation verwendete Notation formal beschrieben. Eine derartige Beschreibung ist in der Literatur bisher nicht zu finden.

Etikettierungsebenen werden allein durch ihren Namen dargestellt, gefolgt von runden Klammern, die den Status der Zeitgebundenheit der Ebene enthalten (vgl. Schema 4). Während `(-)` zeitlose Ebenen markiert, steht `(S)` für intervall-zeitgebundene und `(E)` für event-zeitgebundene Ebenen. Die Schemata in Schema 4(a–c) zeigen Beispiele mit eingesetzten Namen für Etikettierungsebenen. Eine explizite Markierung der Zeitlosigkeit wurde in Harrington (2010a) nicht verwendet, was formal gesehen jedoch zu unterschiedlicher Syntax für die verschiedenen Ebenen führen würde. Derartige Unterschiede führten in der Implementierung dazu, Ebenen je nach Typ unterschiedlich behandeln zu müssen, was nicht schwierig aber ineffizient ist.

Die Beziehung zwischen den Ebenen (vgl. Schema 5 und 6) wird durch gerichtete Linien zwischen untereinander angeordneten Ebenen ausgedrückt. Ein Pfeil am Ende der Linie markiert hierbei eine one-to-many Beziehung, während Pfeile an beiden Seiten der Linie eine many-to-many Beziehung modellieren.

Schema 3 (Ebene).

	$NAME(T)$		
`mit` T :	$-(zeitlos)$	$S(segment)$	$E(event)$

Schema 4 (Ebenenbsp.).

$Sprecher(-)$	$Phon(S)$	$Ton(E)$
(a)	(b)	(c)

In der EMU-Notation werden Ebenen in linearer Beziehung ohne Linienverbindung direkt untereinander gesetzt, wobei alle der ersten Ebene folgenden Ebenennamen

Schema 5 (Assoziation). $\underset{\text{(a)}}{\begin{matrix} EBENE \\ \downarrow \\ EBENE \end{matrix}}$ $\underset{\text{(b)}}{\begin{matrix} EBENE \\ \updownarrow \\ EBENE \end{matrix}}$ $\underset{\text{(c)}}{\begin{matrix} EBENE \\ =EBENE \end{matrix}}$

Schema 6 (Assoziation Bsp.). $\underset{\text{(a)}}{\begin{matrix} Phrase(S) \\ \downarrow \\ Ton(E) \end{matrix}}$ $\underset{\text{(b)}}{\begin{matrix} Silbe(-) \\ \updownarrow \\ Phon(S) \end{matrix}}$ $\underset{\text{(c)}}{\begin{matrix} Phon(S) \\ =Phonem(-) \end{matrix}}$

mit = vor dem Namen versehen sind. Im Gegensatz dazu werden in Harrington (2010a) lineare Beziehungen nicht explizit markiert. Hier werden die in linearer Beziehung stehenden Ebenen ausschließlich direkt untereinander geschrieben. Eine fehlende explizite Markierung der linearen Beziehung stellt wieder eine Ausnahme in der Modellierung dar, denn Assoziationen sind sonst markiert. Weiterhin kommt es zur Ambiguität zwischen untereinander aufgelisteten Ebenen ohne Assoziation und mit linearer Beziehung. Aus diesem Grund wird auf die explizite Markierung in der hier vorgeschlagenen Notation nicht verzichtet.

In Harrington (2010a) werden diese Ebenen auch nicht mit einer Zeitgebundenheitsmarkierung versehen. Somit sind sie explizit als zeitlos modelliert. Jedoch könnte alternativ die Zeitgebundenheitsmarkierung, derjenigen der übergeordneten Ebene entsprechen. Denn wie in Kapitel 3.1 beschrieben wurde: teilen alle Tokens auf beiden Ebenen alle Eigenschaften außer dem Etikett, so können derartige Tokens als mit unterschiedlichen Etiketttypen (Labeltypen) ausgestattet angesehen werden. Im EMU-Datenbankmanagementsystem werden derartige Etiketten auch als Labeltypen angefragt.

Unter Berücksichtigung phonetischer und besonders phonologischer Theorien, zeigt sich die Notwendigkeit, die Ebene als zeitlos markieren zu können, wie in Schema 6(c) dargestellt ist. Derartige lineare Beziehungen können dafür genutzt werden, um Tokens weitere Informationen hinzuzufügen, so dass auch zeitunabhängige Metadaten zum eigentlichen Token theoriegerecht angegeben werden können. Auch in Hinsicht auf die Vererbung der Zeiten aus der assoziierten zeitgebundenen Ebenen ist die Modellierung der Labeltypen als zeitlose Ebene sinnvoll.

Ein Aspekt für die Verwendung einer Modellierung mit einer expliziten Zeitgebundenheitsmarkierung, die der assoziierten Ebene entspricht, ist in einer Anwendung und einer angebotenen Sicht auf die Daten zu finden. Das EMU-System bietet eine Sicht und eine Applikation an, die Ebenen aus einer Etikettierungskette extrahie-

ren kann, wobei die Ebene X und Ebene Y in Abbildung 3.2 als separate Ebenen behandelt und somit individuell mit allen Zeitmarken extrahiert werden können.

Beide Modellierungen (als zeitlose Ebene, als identisch zeitgebunden mit der assoziierten Ebene) sind gleichermaßen vertretbar. Da in der hier vorgeschlagenen Notation jede Ebene durch Ebenenname und Zeitlosigkeitsmarkierung modelliert ist (vgl. Schema 4), wird diese auch bei Label-Links notiert. So kann bei der Konzeptionierung des Etikettierunsschemas durch den Forscher eine geeignete Wahl getroffen werden.

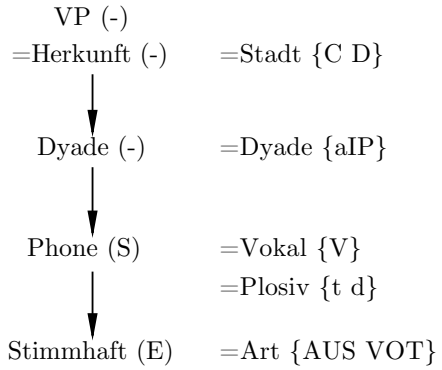


Abbildung 3.4.: Konzeptuelle Darstellung der Etikettierungsebenen und Beziehungen in John(2004).

Enthält ein Schema mehrere Ketten, so werden diese nebeneinander dargestellt. Sind Ketten miteinander verbunden, dann ist diese Beziehung durch einen gerichteten Pfeil zwischen den Ebenen, die diese Verbindung der Ketten auslösen, markiert. Wie aus Kapitel 3.1 hervorgeht, muss eine Hauptkette definiert sein, die festlegt, aus welcher assoziierten zeitgebundenen Ebene assoziierte zeitlose Ebenen Zeitinformationen ableiten. Die Ebenen der Hauptkette werden sowohl in Harrington (2010a) als auch in dieser Arbeit direkt vertikal untereinander dargestellt. Nebenketten verzweigen sich nach rechts oder links. Die Kette im Anhang A.1 zeigt eine solche mehrfache Kette.

Bisher gibt es keine Notationsvorschläge in der Literatur, die den Wertebereich für die Instanzen im konzeptuellen Modell abdeckt. Im EMU-System kann kein Wertebereich im Sinne von Datentypen angegeben werden, dieser ist nur eine definierte Zeichenmenge (*legal labels*). In dieser Arbeit wird die folgende Modellierung vorgeschlagen: Neben den Ebenen wird die Zeichenmenge als Menge in den aus der Mengenlehre bekannten geschweiften Klammern angegeben, geführt von einem

Gleichheitszeichen und dem Klassennamen. Abbildung 3.4 zeigt ein Beispiel von assoziierten Etikettierungsebenen und deren Zeichenmenge.

Die hier vorgestellte Notation in vertikaler Ausrichtung ist ein Vorschlag für die Modellierung. Alle Elemente können auch in horizontaler Ausrichtung verwendet werden, so dass die Kette in Abbildung 3.4 als

VP(-) = Herkunft(-) → Dyade(-) → Phone(S) → Stimmhaft(E)

bzw. mit möglichen Zeichenmengen als:

VP(-) = Herkunft(-) = Stadt{C D} → Dyade(-) = Dyade {aIP} → Phone(S) = {Vokal {V}, Plosiv {t d}} → Stimmhaft(E) = Art {AUS VOT}

dargestellt werden kann.

3.1.3. Relevanzbeispiele

In der Literatur (John und Bombien, iE; Harrington, 2010a; Cassidy und Harrington, 2001) werden Etikettierungsstrukturen vorgestellt, die oft in größeren Korpora verwendet werden. Im folgenden soll am Beispiel der Abbildung 3.4 aus einer kleinen Datenbank von John (2004), die für eine spezielle phonetische Forschungsfrage – einer ‚Akustischen Analyse der Lenis/Fortis Opposition in Varietäten des Sächsischen' – entworfen wurde, der Nutzen und die Umsetzung verschiedener Konzepte des EMU-Etikettierungsmodells gezeigt werden. Die Datenbank enthält ausschließlich jeweils drei Wiederholungen von 19 Sprechern der folgenden Äußerungen.

- Dieses Geschäft kann er allein nicht leiten.
- Peters Freundin kann sie nicht leiden.

Tokens auf intervall-zeitgebundenen Ebenen sind mit dem Zeitsignal assoziiert und etikettieren verschiedene Bereiche des Zeitsignals. Zeitbereiche im Sprachsignal, die aufgrund ihrer akustischen Eigenschaften verschiedenen Phonen zuzuordnen sind, können mit Tokens auf einer zeitgebundenen Ebene markiert werden. In John (2004) wurden der Diphthong [aɪ] und die Plosive [t] bzw. [d] in den Wörtern <leiden> und <leiten> im Zeitsignal detektiert, On- und Offset der Segmente markiert und mit den Etiketten *V* (Vokal) oder *t* bzw. *d* markiert. Die hierfür verwendete Ebene

wurde in John (2004) entgegen der spontanen Intuition PHONEME[5] benannt. Im Beispiel (vgl. Schema 7) soll die Ebene jedoch Phone heißen, da eine Diskussion der phonologischen Theorien an dieser Stelle ausbleiben soll.

Schema 7. *Phone(S)*

In der phonetischen Forschung sind auch Zeitbereiche interessant, die nur als ein Ereignis im Signal zu finden sind. Das Einsetzen der zweiten Resonanzfrequenz (F2) im Spektrum nach der Verschlusslösung eines Plosivs sowie die Verschlusslösung selbst sind Ereignisse, die sich nicht über ein Zeitintervall erstrecken. Den Bereich zwischen Verschlusslösung und Einsetzen von F2 wird in der Phonetik als Aspirationsphase zumeist mit dem Symbol `h` oder `H` etikettiert. In John (2004) wurde zur Bestimmung des Anteils der Stimmhaftigkeit im postvokalischen Plosivverschluss der Zeitpunkt im Plosivverschluss markiert, der als Ende der letzten periodischen Schwingung nach dem Vokal detektiert wurde und außerdem der Zeitpunkt im Verschluss, falls die periodische Schwingung vor dem Verschluss wieder einsetzt. Für Ersteres wurde das Etikett *AUS* und für letzteres *VOT* verwendet. In solchen Fällen sind die event-zeitgebundenen Ebenen interessant, wie in Schema 8 modelliert, so dass die Tokens auf diesen Ebenen nur mit einer Zeitmarke bzw. einer identischen Start- und Endzeit versehen sind.

Schema 8. *Stimmhaft(E)*

Der Zeitpunkt des Aussetzens und des Einsetzens von Stimmhaftigkeit im Verschluss sollten im Zeitintervall des etikettierten Plosivs liegen, so dass eine Zugehörigkeit der Ereignisse durch die Zeiten impliziert ist. Bei einer Analyse müsste man somit in dem Fall, dass mehrere Plosive in einem Zeitsignal vorkommen, die Zeiten der Etiketten abfragen und die Zuordnung der beiden Marken *AUS* und *VOT* zum richtigen Plosiv anhand der Zeitmarken detektieren. Dieser Umstand wird umgangen, wenn die Zusammengehörigkeit der einzelnen Etiketten explizit gemacht wird. Hierfür bietet das EMU-Etikettierungsmodell das Konzept der Assoziationen an. Die Tokens auf der Phone-Ebene können mit denen auf der Stimmhaft-Ebene assoziiert werden. Da Tokens auf beiden Ebenen mit unterschiedlichen Zeitmarken assoziiert sind, ist die Beziehung als autosegmentell einzustufen. Die Modellierung zeigt Schema 9.

Schema 9.
$$\begin{array}{c} Phone(S) \\ \downarrow \\ Stimmhaft(E) \end{array}$$

[5]Die Begründung in John(2004) liegt in der Aufbereitung der Daten für eine spezielle Forschungsfrage: wie werden verschiedene Phoneme realisiert. Die Realisierung wurde segmentiert und das angenommene zugrundeliegende Phonem wurde als Etikett verwendet.

Zeitlose Ebenen, d. h. Ebenen, auf denen Tokens keine explizite Verbindung zum Zeitsignal haben, können in ganz verschiedenen Situationen zum Einsatz kommen. Informationen zum Sprachsignal, die jedoch nicht einem Ereignis oder einem Zeitintervall im Zeitsignal entsprechen, können auf diese Weise in die Datenbasis aufgenommen werden. In der Beispieldatenbank stammen die 19 Sprecher aus unterschiedlichen Dialektregionen. Für Sprecher aus Chemnitz wurde auf einer zeitlosen Ebene mit dem Namen Herkunft ein *C* gesetzt, während für Sprecher aus Dresden ein *D* gesetzt wurde. Derartige Informationen sind Metadaten, die auf diese Weise in die Datenbank eingepflegt werden können. Es gibt u. U. mehrere Informationen, die als Metainformation eingepflegt werden sollen, wie in der Beispieldatenbank etwa das Versuchspersonenkürzel des Sprechers. Herkunft und Name des Sprechers sind abhängig voneinander, in der Art und Weise, dass jeder Sprecher genau eine Herkunft hat. Diese Art von Zusammengehörigkeit kann mithilfe von unterschiedlichen Etiketten für das gleiche Token realisiert werden. Im EMU-Etikettierungsmodell wird diese Art der Beziehung wie in Schema 10 über Ebenen in linearer Beziehung repräsentiert. Obwohl die Herkunft für die eigentliche Untersuchung entscheidender ist als das Versuchspersonenkürzel des Sprechers, aber nicht jede Herkunft notwendiger Weise nur von einem Sprecher erfüllt wird, sollte das Modell die Versuchsperson als Haupteigenschaft definieren mit der zusätzlichen Information der Herkunft auf einem so genannten Label-Link zu einer Versuchspersonen-Ebene.

Schema 10.
$$\begin{array}{c} VP(-) \\ =Herkunft(-) \end{array}$$

Wenn alle Phone im Zeitsignal markiert und etikettiert sind, könnten Wortgrenzen zusätzlich auf einer weiteren zeitgebundenen Ebene hinzugefügt werden[6]. Sind jedoch alle Phone markiert, ergeben sich die Wortgrenzen aus den Segmentgrenzen der Phone. Einem Wortetikett müssten somit nur die Phone assoziiert werden. Hierfür bietet sich die Verwendung einer Dominanzbeziehung (vgl. Schema 11) aus dem EMU-Etikettierungsmodell an. Ein Wort dominiert im Modell seine Phone oder die Phone bilden das Wort. Auf diese Weise erbt das Token der Wortebene die Zeitmarken der Phone. Anfangs- und Endzeiten des Tokens werden durch diese hierarchische Beziehung impliziert. In John (2004) wurde die Vokal-Plosivdyade auf diese Weise etikettiert.

Schema 11.
$$\begin{array}{c} Dyade(-) \\ \downarrow \\ Phone(S) \end{array}$$

[6] In anderen Programmen ist das die einzige Variante, Wortgrenzen hinzuzufügen.

Ein weiteres Beispiel für die Nutzung zeitloser Ebenen ergibt sich für das Einfügen von phonologischen Informationen in die Datenbank. Phoneme können per Definition nicht an die Zeit gebunden sein. Nur eine Realisierung eines Allophons des Phonems kann mit dem Zeitsignal assoziiert sein. In John und Bombien (iE) wird eine Modellierung vorgestellt, in der die zugrundeliegenden Formen auf einer zeitlosen Ebene markiert sind. Diese Ebene steht mit einer zeitgebundenen Ebene in hierarchischer Beziehung, die die realisierten Phone markiert und etikettiert. Auf diese Weise können auch Tilgungen markiert werden. Tilgung ist einer der möglichen phonologischen Prozesse, die mit hoher Frequenz in der Spontansprache auftreten. Im Deutschen ist der Zentralvokal Schwa besonders in [ən] Morphen der Morpheme {PLURAL}, wie in <Frauen> [fʁaʊ+ən] aber auch {INFINITIV}, wie in <leiden> [laɪd+ən] von Tilgung betroffen. So wird in beiden Fällen das in der Zitierform durchaus vorhandene Morph [ən] in der Spontansprache zu einem silbischen Nasal [n̩] reduziert. Da es ein häufig auftretender Prozess an diesen Morphen ist, kann er in der Etikettierung berücksichtigt und über hierarchische many-to-many Beziehungen modelliert werden. Ein solches Beispiel wird in Kapitel 7 und Abbildung 3.5 vorgestellt.

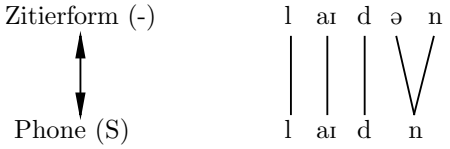


Abbildung 3.5.: Konzeptuelle Darstellung der Etikettierungsebenen und Assoziationen eines Beispiels für hierarchische many-to-many Beziehungen (links) und einer Etikettierung (rechts).

Im Etikettierungsbeispiel in Kapitel 7 werden auch mehrere Ketten von assoziierten Ebenen verwendet, um die etikettierten prosodischen Strukturen von den segmentellen unabhängig betrachten zu können. Für mehrfache Ketten findet sich in der Literatur häufig das Beispiel der Morph- und Silbenzerlegung von Wörtern anhand von Phonen oder Phonemen. Da Silben- und Morphgrenzen nicht übereinstimmen müssen, die Einheiten Silbe und Morph in keiner Beziehung zueinanderstehen, aber dennoch die Gemeinsamkeit teilen, eine Einheit zwischen Phon und Wort darzustellen, sollten beide auf ähnliche Weise in einer Etikettierung behandelt werden können. In den Hierarchien Wort → Silben → Phone und Wort → Morphe → Phone sind Wort und Phone identisch.

Derartige Zusammenhänge können im EMU-Etikettierungsmodell mithilfe mehrfacher Ketten modelliert werden, wie in Abbildung 3.6(a). In dieser Abbildung bilden die Morpheme, von denen die Morphe Allomorphe sind, das Etikett des Morphtokens. Somit erbt das Morphem die Zeiten aus den Phonen. Da Morpheme genauso wie Phoneme per se zeitlos sind, ist diese Etikettierung nicht mit phonologischen Theorien konform. In Abbildung 3.6(b) wird das Morphem als Eigenschaft des Morphs modelliert, um diese Problematik zu umgehen.

An dieser Stelle sei noch mal betont, dass eine Etikettierung eine Diskretisierung durch Modellbildung ist. Dabei stehen auch die Forschungsfrage und die Methoden zur Analyse im Fokus. Ein Datenbankschema muss nicht gänzlich forschungstheoretisch belegbar sein. Primär müssen die Informationen, die zur Analyse benötigt werden, einfach zugänglich sein. Die Schemata in Abbildung 3.6(a) und (b) sind bezüglich des Informationsgehaltes nicht verschieden, (b) enthält jedoch zugunsten der Theoriekonformität redundante Information.

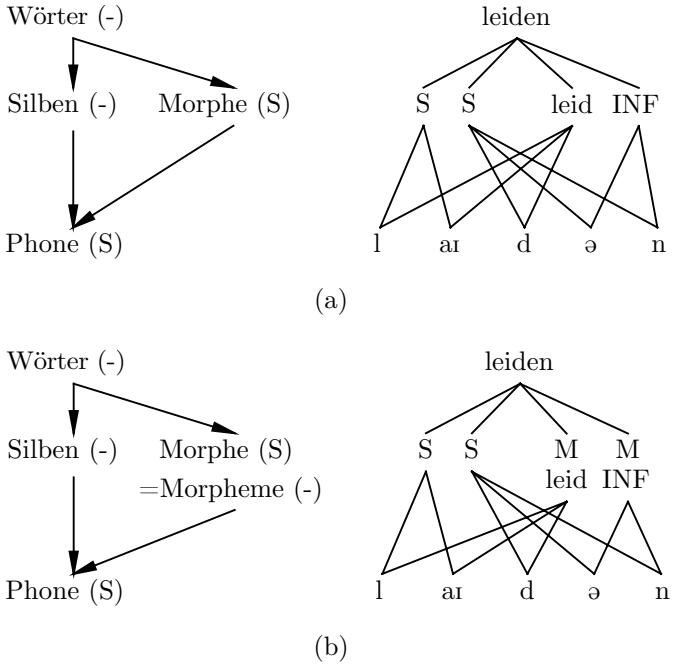


Abbildung 3.6.: Konzeptuelle Darstellung der Etikettierungsebenen und Assoziationen eines Beispiels für mehrfache Ketten (links) und einer Etikettierung in mehrfachen Ketten (rechts). In (a) sind die Morphe über ihre Morpheme etikettiert. (b) zeigt eine theoriekonformere Etikettierung mit dem Morphem als Eigenschaft des Morphs.

Zusammenfassend zeigen die gegebenen Relevanzbeispiele, wie mit dem EMU-Etikettierungsmodell auf einfache Weise theoriekonforme, aber auch einfach nur praktische Etikettierungsschemata entworfen werden können. Hierfür bietet das Modell einen kleinen Satz an »Bausteinen«, deren Bedeutung leicht nachvollziehbar ist. Den größten Vorteil, den das Etikettierungsmodell im EMU-System gegenüber Etikettierungsmöglichkeiten anderer Software hat, ist das Bereitstellen zeitloser Ebenen, die nicht nur für Metadaten, sondern auch für Signaletikettierungen genutzt werden können. Dieser Vorteil ergibt sich durch die weitere wichtige Eigenschaft des Modells, explizite Assoziationen zwischen den Ebenen zu erlauben und auch zu interpretieren (Ableiten der Zeiten). Obwohl das Modell aus derzeitiger Sicht alle Möglichkeiten der Modellierung zulässt, können die Entwickler des EMU-Systems nicht alle Anforderungen der Nutzer kennen. Das Modell als auch die Implementierung ist erweiterbar, sofern Bedarf besteht, den die Nutzer mitteilen können und sollten[7].

[7] www.emu.sf.net

3.2. Datenbankenkonzept im EMU-System

Für die Weiterentwicklung des Systems in jeglicher Hinsicht aber besonders für die Erweiterungen in der Interoperabilität, wie sie in Kapitel 6 vorgestellt werden und in der Benutzerfreundlichkeit (vgl. Kapitel 4) war es notwendig, das EMU-Datenbankenkonzept zu kennen sowie zu wissen, inwiefern der Benutzer in die Datenbankstrukturen eingreifen kann. Hierfür musste mithilfe der Beschreibungen in der Literatur (Cassidy und Harrington, 1996; Cassidy und Harrington, 2001), der eigenen Erfahrung und dem EMU-Quellcode (EMU Developers, 2011a) eruiert werden, welche Teile zu einer Datenbank gehören, um sie zum Beispiel in andere Systeme exportieren zu können. Weiterhin wurde eruiert, wie die Daten in der Datenbank angefragt und manipuliert werden können, um sie zum Beispiel an andere Programme weiterzuleiten sowie Manipulationsmöglichkeiten in geeigneter Form auch dem Benutzer zugänglich zu machen als auch externe Veränderungen an den Daten in die Datenbank einpflegen zu können. Die Art der Datendefinition und auch welche Definitionen durch den Benutzer und in welcher Form erfolgen, musste bekannt sein, um zum Beispiel Daten aus anderen Programmen importieren zu können oder dem Benutzer geeignete Werkzeuge zur Verfügung zu stellen, die die Arbeit der Datendefinition erleichtern.

Das EMU-Datenbanksystem unterscheidet sich von Datenbanksystemen, die in Firmen zur Erfassung von Kunden-, Artikeldaten oder anderen Einheiten verwendet werden. Eine EMU-Datenbank soll von jedem Datenbankanwender erzeugt werden können, denn jeder Wissenschaftler, der mit Sprachdaten umzugehen hat, wird während seiner Forschungsaufgaben eine Vielzahl an Datenbanken erstellen. Hierzu sollte kein externer Datenbankadministrator oder Informatiker notwendig sein.

In Kapitel 2.1 wurden die vier Schritte Anforderungsanalyse, konzeptioneller, logischer und physischer Entwurf vorgestellt, die während des Datenbankentwurfs durchlaufen werden. Die Anforderungsanalyse entfällt für die EMU-Anwender in ihrer Rolle als Datenbankdesigner, da mit der Verwendung des EMU-Systems die Anforderungen festgeschrieben sind. Die Datenbank soll Etikettierungen enthalten, die auf Ebenen organisiert sind und Signale enthalten, die unterschiedlichen Formats sein können. Das sind die grundlegenden Anforderungen des Benutzers an eine Etikettierungssoftware in der linguistischen und phonetischen Forschung. Das Ergebnis der Anforderungsanalyse ist für unterschiedliche Programme für die Sprachverarbeitung nicht identisch. Somit können für das EMU-System weitere Anforde-

rungen postuliert werden. Die Etikettierungen auf unterschiedlichen Ebenen sollen miteinander auf unterschiedliche Weise assoziiert werden können. Weiterhin darf es Etikettierungen geben, die entweder mit einer Zeit, einem Zeitintervall oder keiner Zeitinformation assoziiert sind.

Da die einzelnen Schritte im Datenbankentwurf nicht klar zu trennen sind, könnte man auf der einen Seite die Entscheidung, welche Ebenen und Beziehungen zwischen den Ebenen für die aktuelle Benutzung der zu entwerfenden Datenbank benötigt werden, zur Anforderungsanalyse zählen. Auf der anderen Seite könnte das Definieren der Ebenen und Beziehungen Teil des konzeptuellen Entwurfs sein. Nicht jeder EMU-Anwender kann sich dabei des ER-Modells (vgl. Kapitel 2.1.2) bedienen. In Kapitel 3.1 wurde jedoch ein konzeptuelles Datenmodell verwendet, welches bisher keinen expliziten Namen trägt, aber als ein solches für das EMU-Etikettierungsmodell verstanden werden kann. Das konzeptuelle EMU-Etikettierungsschema, das vom Benutzer erstellt wird, kann jedoch nur als eine Sicht betrachtet werden. Der Nutzer kann und muss nicht einschätzen, welche Objekte in der Gesamtsicht noch benötigt werden, um das Datenbankschema umsetzen zu können. Jedoch ist es Aufgabe des Programmierers, der sich mit der Weiterentwicklung des Systems auseinandersetzt.

Der Nutzer definiert also eine Sicht des konzeptionellen EMU-Etikettierungsschemas unter Berücksichtigung des EMU-Etikettierungsmodells, das verschiedene Objekte und Attribute für die Modellierung bereitstellt (vgl. Kapitel 3.1). Der Nutzer definiert hierbei nur die Ebenen und deren Beziehungen, die als Container für die Etikettierungen benötigt werden.

Neben den Etikettierungen, für die ein Modell für die Konzeptualisierung der Datenbank bereitsteht, enthalten die EMU-Datenbanken auch Zeitsignale. Diese Art von Objekten wurde jedoch in der Literatur bisher nicht modellhaft dargestellt.

> "[. . .] setting up a new template file [Datenbankschema] is sometimes not completely straightforward, and this represents a major part of the learning curve needed to master EMU." Williams (2008, S. 168)

Der im Zitat beschriebene Umstand wurde in dieser Dissertation als Nachteil angesehen, der durch eine benutzerfreundlichere Lösung behoben werden sollte. Datenmodelle, die Speicherung der Daten, die Erzeugung von Datenbankschemata und die Möglichkeiten zur Modifikation von Daten werden im Folgenden als notwendiges grundlegendes Wissen für die Vorstellung der Weiterentwicklungen in den Kapiteln 4.3, 5.1, 5.3 und 6 näher erläutert.

3.2.1. Datenbank

Während nach Elmasri und Navathe (2009) Datenbanken separat von dem Datenbankschema im Datenbankkatalog des Datenbanksystems enthalten sind, bilden nach Kemper und Eickler (2006) Datenbankschema und Datenbasis zusammen die eigentliche Datenbank. Im EMU-System können Datenbanken sowohl vergleichbar zu Kemper & Eickler als auch zu Elmasri & Navathe angesehen werden. Bereits an dieser Stelle ist der EMU-Anwender für das Design verantwortlich.

Ob mit oder ohne Datenbankschema enthält jede EMU-Sprachdatenbank eine Sammlung von Sprachdaten. Dazu gehören Sprachsignale, vom Sprachsignal abgeleitete Signale wie Grundfrequenzdaten, Formantdaten etc. und Etikettierungen der vorhandenen Sprachsignale. Zur Datenbank gehörende Signale und Etikettierungen liegen in Form von unterschiedlichen Dateien vor (Cassidy und Harrington, 1996). Jedes einzelne Sprachsignal ist mit einem in der Datenbank einzigartigen Namen versehen. Der Name des Sprachsignals gilt als Name der Äußerung. Alle Signal- und Etikettierungsdateien einer Äußerung haben den gleichen Basisnamen, so dass jede Datei eindeutig einer Äußerung zugeordnet werden kann. Zu einer Äußerung abc001 könnten die Dateien für das Sprachsignal abc001.wav, die Grundfrequenzdaten abc001.sf0 und die Etikettierungen abc001.hlb und abc001.ph gehören.

3.2.2. Datenmodell mit ER-Modellierungen

Das EMU-Datenmodell setzt sich aus verschiedenen Komponenten zusammen. Hierzu gehören das Datenmodell auf interner Ebene, dass dem Anwender nicht bekannt sein muss, jedoch aber dem Programmierer. Auf interner Ebene gibt es weiterhin den Unterschied zwischen persistenten und transienten Daten. Persistente Daten sind hierbei physisch vorhandene Dateien, während transiente Daten nur während der Laufzeit des Systems im Speicher vorhanden sind. Diese Unterscheidung muss im EMU-System gegeben sein, da EMU eine Software für PCs ist und nicht wie andere Datenbanksysteme auf Servern eine permanente Laufzeit hat. Da der Anwender auf Grundlage des gegebenen Datenmodells die Struktur der selbst entworfenen Datenbank definieren muss, besteht die Notwendigkeit, ein vereinfachtes Datenmodell auf externer Ebene anzubieten.

Sowohl internes als auch externes Datenmodell werden konzeptuell im Folgenden vorgestellt. Auch die Modellierung persistenter Daten wird gezeigt, während für

die physische Modellierung transienter Daten auf den EMU-Quellcode verwiesen sein soll (EMU Developers, 2011a). Die Umsetzung des Datenmodells auf dieser Ebene wurde im Rahmen der Arbeit eruiert, jedoch nur um das konzeptuelle EMU-Datenmodell als Orientierung abbilden zu können.

3.2.2.1. Konzeptuelle Modelle auf verschiedenen Ebenen

Wie bereits für die Etikettierungsstrukturen des EMU-Systems im vorangegangenen Kapitel gezeigt wurde, sind konzeptionelle Modellierungen von Strukturen ein sehr gutes Werkzeug in der Arbeit mit Software aber auch in der Softwareentwicklung. In der Literatur (Cassidy und Harrington, 2001; Harrington, 2010a) finden sich Beschreibungen des externen Datenmodells und auch in Ansätzen wie in Cassidy und Harrington (2001) des internen. Diese sollen jedoch dazu dienen, die Funktionsweise des Systems beispielhaft zu erläutern. Somit eignen sie sich nicht als Arbeitsgrundlagen für die Entwicklung des Systems. Eine konzeptuelle Modellierung in Bezug auf das Datenmodell des EMU-Systems ist weder für die externe noch interne Ebene aus der Literatur bekannt und ist daher im Rahmen der Dissertation als Vorarbeit entstanden. Besonders für die Entwicklung einer grafischen Oberfläche, die den Benutzer in der Definition des externen Datenbankschemas unterstützen soll, war der Überblick über die Zusammenhänge verschiedener Objekte im Modell sehr wichtig, um eine geeignete Darstellung aber auch effiziente Verarbeitung gewährleisten zu können. Als Modellierungssprache wurde das ER-Modell gewählt und für das EMU-Modell auf interner und externer Ebene ER-Schemadiagramme entworfen.

Ein EMU-Datenbankschema ist auf externer Ebene in einem so genannten EMU-Template festgelegt. Ein EMU-Template ist eine textbasierte und benutzereditierte Datei. Es enthält Beschreibungen zum Etikettierungsschema, den Signaldaten und den Zugriffspfaden auf die Dateien, in denen die Daten gespeichert sind. Weiterhin enthält es Konfigurationen für Applikationen im EMU-System, wie die gewünschte Verwendung von Plug-Ins oder Skripten und Darstellungsoptionen (Cassidy und Harrington, 2001; Harrington, 2010a). Das Datenbanktemplate enthält nur die Benutzersicht des Datenbankschemas. Die Gesamtsicht wird durch das DBMS automatisch erzeugt. So werden die in der Benutzersicht vorhandenen Objekte automatisch mit ggf. weiteren oder anderen Attributen ausgestattet, die das interne Schema datenmodellpassend benötigt.

Abbildung 3.7 (S. 73) zeigt das in dieser Dissertation erstellte und als Werkzeug verwendete Templateschemadiagramm in ER-Notation[8] mit allen Elementen, die im externen benutzerdefinierten EMU-Datenbankschema bezüglich Etikettierungen und Signalen definiert werden. Es stellt somit das konzeptuelle EMU-Datenmodell auf externer Ebene dar. Für jeden Entitätstypen können bei der Erstellung eines Datenbankschemas die Attribute mit Werten belegt werden. Abbildung 3.8 zeigt ein ER-Schemadiagramm des EMU-Datenmodells ohne Implementierungdetails wie Datentypen und -strukturen. Es stellt somit wiederum ein konzeptuelles EMU-Datenmodell dar jedoch auf interner Ebene. Da beide Modelle als ER-Schemadiagramme dargestellt sind, werden sie im Folgenden als internes und externes Schema bezeichnet, wobei mit Schema ER-Schemata gemeint sind, soweit sie nicht explizit als EMU-Schemata bezeichnet werden[9].

Im Gegensatz zu allen anderen Entitätstypen in Abbildung 3.7 bildet **variable** keinen Beziehungstyp aus. Der Grund dafür liegt in der Funktion. In den Ausprägungen dieses Typs werden Variablen gesetzt, mithilfe derer das EMU-System, im besonderen das Etikettierungstool, konfiguriert wird. Der Entitätstyp **TEMPLATE** im internen Schema in Abbildung 3.8 stellt die Verbindung der Variablen zur Datenbank her, indem sie im strukturierten Attribut **variables** des Entitätstypen vorhanden sind. Strukturiert ist das Attribut, da es in weitere Attribute aufgespalten ist. Die Ausprägung einer **TEMPLATE variable** bildet die entsprechende Variable mit ihren Wert. Mögliche Variablen in EMU-Datenbankschemata (Templates) werden in Kapitel 3.2.3 aufgelistet.

Der **track**-Entitätstyp in Abbildung 3.7 repräsentiert die Signale, da sie im EMU-System als sog. *Tracks* verwendet werden. Die Zeitsignale liegen als Dateien vor, welche die Existenz des Attributs **extension** für die Dateiextension nahelegt. Der Name im **Name**-Attribut ist hierbei je nach Dateityp arbiträr oder für SSFF-Dateien im Dateiheader vordefiniert. Im SSF-Format können unterschiedliche Daten in verschiedenen Spalten gesichert werden, wie Formantwerte und Bandbreiten zusammen in einer Datei. Der *Track*-Name entspricht der Spalte in der SSFF-Datei.

[8] Die Entitätstypen werden in Analogie zur *Data Definition Language* (vgl. Kapitel 3.2.3) in Kleinbuchstaben angegeben und entsprechend alternativ formatiert. Attribute sind, soweit kein analoger DDL Ausdruck vorhanden ist, orthographisch dargestellt. Auf die Spezifikation struktureller Einschränkungen für Beziehungen wurde bewusst verzichtet, da diese aus der Beschreibung und der Semantik hervorgehen.

[9] Schema ist hier wie im allgemeinen Sprachgebrauch als Strukturbeschreibung zu verstehen und in diesem Kontext mit dem Informationsschema (vgl. S. 32) vergleichbar, das jedoch nicht im Datenbanksystem enthalten ist.

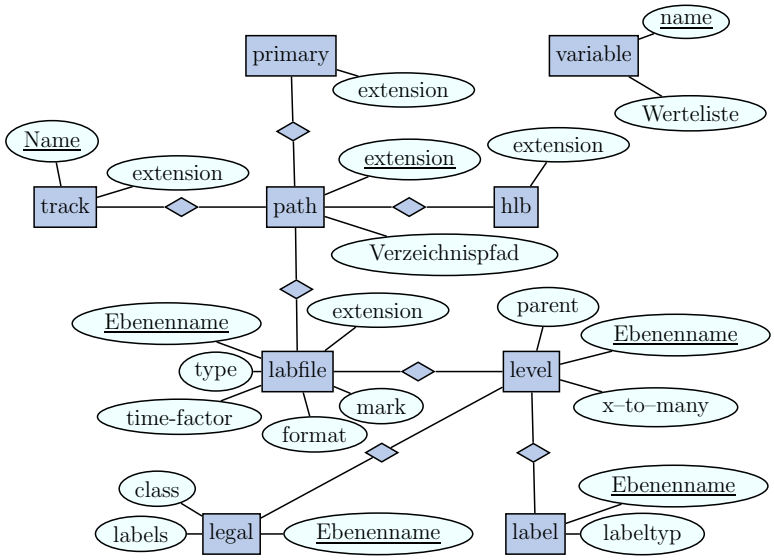


Abbildung 3.7.: Vereinfachtes Templateschemadiagramm in ER-Darstellung.

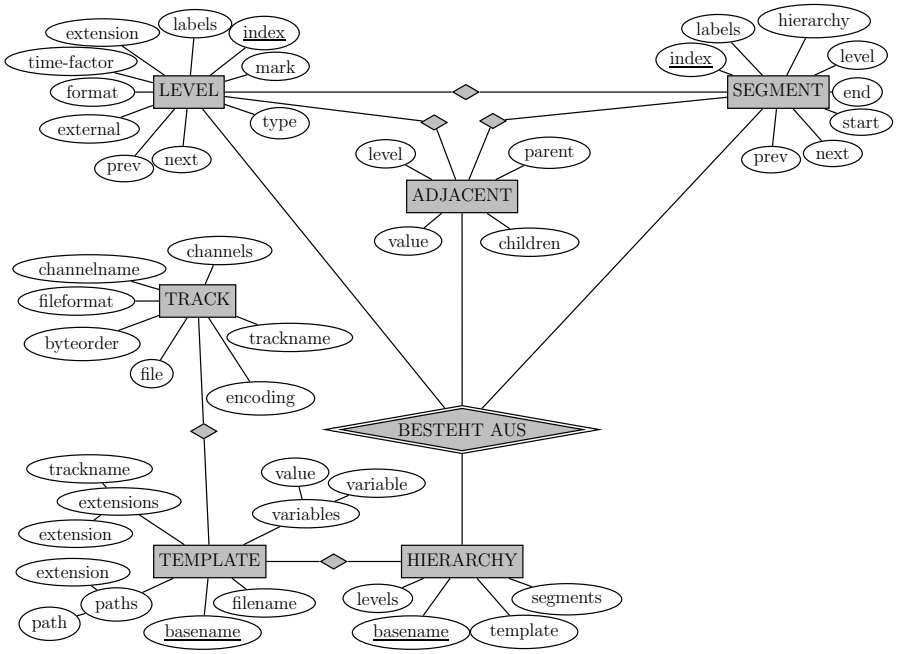


Abbildung 3.8.: Vereinfachter Ausschnitt des aus dem Quellcode abgeleiteten internen EMU-Datenmodells aufbereitet als ER-Schemadiagramm.

Die Entitätstypen **path** und **track** bilden einen Beziehungstyp. Die explizite Verbindung der beiden ist in der Verwendung gleicher Attributnamen umgesetzt, nämlich **extension**. Somit müssen auch Verzeichnispfade für die Dateiextensionen der definierten *Tracks* angegeben sein. Im EMU-Datenbankschema sind die Signale in der Datenbasis damit ausreichend beschrieben. Im internen Schema sind Signale als `snack`-Sound-Objekte realisiert, die wie der Entitätstyp **TRACK** zeigt, verschiedene Attribute aufweisen, die nicht im externen Schema enthalten sind wie das **file**-Attribut, während das Attribut **extension** nicht enthalten ist. Über das **trackname**-Attribut sind die Sound-Objekte jedoch in der Datenbank durch den mit **TEMPLATE** gebildeten Beziehungstyp, eingebunden. Der Entitätstyp **TEMPLATE** enthält das für **TRACK** notwendige Attribut **extension** mit **trackname** zusammen als Struktur des **extensions**-Attributs. In der Ausprägung sind hier die *Track*-Namen und die Dateiextensionen der Signaldateien zu sehen, deren Pfad wiederum als Ausprägung des ebenfalls strukturierten Attributs **paths** zu finden ist, dass wiederum in Kombination im **file**-Attribut des **TRACK**s enthalten ist. Alle weiteren Attribute von **TRACK** sind signalspezifisch.

Für die Etikettierung werden analog zum Etikettierungsmodell, wie es in Kapitel 3.1 vorgestellt wurde, Ebenen über ihre Benennung definiert. Den einzelnen Ebenendefinitionen werden die Attribute, die Art der Zeitgebundenheit, die in der Ebenenhierarchie direkt dominierende Ebene und deren mögliche x-to-many Assoziation hinzugefügt. Das externe Schema in Abbildung 3.7 sieht dafür verschiedene Entitätstypen vor: für die Ebenendefinition **level** mit der möglichen Verbindung zu **labfile** für die zeitgebundenen Ebenen, und **label** für Label-Links bzw. Labeltypen.

Gegenüber dem konzeptuellen Etikettierungsschema benötigen **labfile** Entitäten, die zeitgebundene Ebenen repräsentieren, einen Wert für das Attribut **time-factor**[10] und genauso wie die **track** Entitäten einen Wert für das Attribut **extension**, damit einhergehend weiterhin einen Wert für den Verzeichnispfad in **path**. Außerdem gibt es die Attribute **mark** und **format**. In der Ausprägung kann **mark** die Werte START und END enthalten und beeinflusst damit die Speicherstrukturen (vgl. Kapitel 3.2.2.2). Gleiches gilt für **format** mit den Ausprägungen: ESPS, SPEECHSTN, ACCOR, TIMIT, KIEL.

Im internen Schema (vgl. Abbildung 3.8, S. 73) sind die Ebenen, ob zeitlos oder nicht, über einen Entitätstyp **LEVEL** ausgedrückt, der alle Ebenenattribute ent-

[10] Das Attribut time-factor wird in Kapitel 3.2.3 erwähnt; es stellt die Länge einer Zeiteinheit dar. Damit wird die Genauigkeit der Zeitangaben bei der Speicherung definiert.

hält. In der Ausprägung werden die Ebenentypen über das **type**-Attribut bestimmt, das EVENT, SEGMENT oder leer sein kann. Auch eine separate Modellierung der Label-Links, wie sie im externen Schema vorhanden ist, wird im internen Schema im Entitätstyp **LEVEL** als Attribut untergebracht. Label-Links werden hier direkt als Liste im **labels**-Attribut zusammengefasst. An dieser Stelle sei darauf hingewiesen, dass das genau mit der Label-Link Notation für das Etikettierungsschema in Harrington (2010a) übereinstimmt. Weiterhin sei auch darauf hingewiesen, dass das interne Schema implementierungsnah ist. In der Implementierung ist die Effizienz und nicht die Theoriekonformität vorrangig.

Für Effizienz sorgt auch das Attribut **index** des **LEVEL**-Entitätstypen. Denn obwohl Ebenen eindeutig durch ihren Namen identifiziert sind, enthält dieses Attribut eine Ebenen-Identifikationsnummer, da Zahlen als Indizes in der Indizierung schneller verarbeitbar sind. Durch die implementationsnahe Modellierung der Label-Links in einem Ebenennamenvektor haben diese keine eigene Identifikationsnummer.

Genauso, wie für die Signale sind Verzeichnispfad und Dateiextension für **LEVEL** nicht als Attribut am entsprechenden Entitätstyp enthalten, sondern als Ausprägungen der **paths**-Attribute im **TEMPLATE**-Entitätstypen.

Ganz anders verhält es sich mit den im Etikettierungsschema vorhandenen Assoziationen zwischen den Ebenen. Sie sind im externen Schema über ein einziges Attribut **parent** zu **LEVEL** modelliert, das die jeweilige Elternebene in der Ausprägung enthält und im internen Schema über die Attribute **prev** und **next**, die die jeweilige Identifikationsnummer (ID) der jeweiligen Eltern- bzw. Kindebene in der Ausprägung enthalten. Die Beziehungen sind weiterhin im Entitätstyp **ADJACENT** des internen Schemas enthalten. In der Ausprägung gibt es pro Ebene eine Liste der assoziierten Ebenen.

Im EMU-Template ist es möglich, für jede Ebene unterschiedliche Wertebereiche bzw. Zeichenmengen in unterschiedlichen Klassen zu definieren. Diese Zeichenmengen stehen im externen Schema nach Abbildung 3.7 (S. 73) ebenfalls in Verbindung mit der Ebene, und erst durch das interne Schema auch zu den Segmenten, denn Segmente können derartige Werte annehmen. Man kann diese Wertebereiche auch als Integritätsbedingungen ansehen, wobei das System trotzdem Werte außerhalb des Wertebereichs zulässt aber in verschiedenen Sichten[11] nur diesen Wertebereich berücksichtigt[12].

[11] vgl. S. 29

[12] z. B. kann eine Klasse das Etikett in einer Abfrage ersetzen (vgl. Kapitel 3.3)

Im internen Schema in Abbildung 3.8 wird das Etikettierungsmodell um das Objekt der realen Welt Token auf einer Ebene mithilfe des Entitätstyps **SEGMENT** erweitert, wobei das Etikett im **labels** und die Zugehörigkeit zur Ebene im **level**-Attribut ausgedrückt sind. Letzteres ist in der Ausprägung die Indexnummer der entsprechenden Ebene. Das **labels**-Attribut in **SEGMENT** zeigt eine enge Verbindung zum **labels**-Attribut im Entitätstyp **LEVEL** und enthält in einem Vektor alle Etiketten pro Label-Link.

In der Datenbankausprägung ist es möglich, Tokens mit gleichen Etiketten zu erhalten. Somit muss ausgeschlossen sein, das Etikettattribut als Schlüsselattribut zu verwenden. Das **index**-Attribut, welches jedem Segment eine eindeutige ID zuordnet, schafft hier Abhilfe und kann als Segmentnummer angesehen werden. Eine Segmentnummer ist eine ganze Zahl, die sich für jedes neue Segment angefangen von 0 um eins erhöht. Die Segmente sind durchgezählt in der Reihenfolge ihres Einfügens.

Im EMU-Etikettierungsmodell sind Segmente auf Ebenen sequenziell angeordnet, dadurch haben sie eine feste Ordnung. Während diese Ordnung bei zeitgebundenen Ebenen durch die **start**- und **end**-Zeitattribute in **SEGMENT** geregelt werden könnte, sind die Attribute für Vor- und Nachfolgetoken (**prev** und **next**) für zeitlose Ebenen unumgänglich. In der Ausprägung enthalten **prev** und **next** die Segmentnummer des jeweiligen Tokens.

Wiederum vergleichbar mit **LEVELS** sind die Assoziationen zwischen den einzelnen Segmenten im **ADJACENT**-Entitätstyp modelliert. Hierbei werden in der Ausprägung jedoch alle möglichen Eltern- und Kindsegmente in der Ebenenliste (siehe oben) paarweise gegenüber gestellt und mit dem Wert 1 im **value**-Attribut als miteinander assoziiert markiert und mit dem Wert 0 als nicht assoziiert[13]. Diese Modellierung ist eine effiziente Variante, um Änderungen in den Assoziationen der Segmente abzubilden. Die Segmente sind indizierbar, die möglichen Assoziationen sind bereits definiert und ihnen wird nur ein Wahrheitswert (boolescher Wert) zugewiesen.

Für die einzelnen Sprechakte aus der realen Welt, für die Äußerungen, welche in unterschiedlichen Dateien vorliegen können, ist kein explizites Äußerungs-Objekt in der externen Modellierung vorgesehen. Sie werden im externen Schema indirekt nur über den Entitätstypen **primary** definiert. Im internen Schema kann auf ein explizites Objekt, das Äußerungen abbildet, nicht verzichtet werden. In Abbildung 3.8 (S. 73)

[13] Zum besseren Verständnis: Eine Assoziation zwischen Tokens ist möglich, sobald sie auf assoziierten Ebenen vorhanden sind. Dennoch müssen die Tokens nicht miteinander assoziiert sein.

werden Äußerungen über den Entitätstypen **HIERARCHY** ausgedrückt. Dieser Typ besitzt die Attribute **basename**, **template**, **levels** und **segments**. Die letzten beiden Attribute sind im internen Schema definiert und in der Ausprägung jeweils nur Referenzen auf die gesamten Entitäten in **LEVEL**, **SEGMENT** bzw. in **ADJACENT**. Diese Verknüpfung ist in der ER-Darstellung mithilfe des **BESTEHT AUS** Beziehungstyps ausgedrückt. An dieser Stelle ergibt sich in der Implementierung eine ineffiziente Redundanz, die jedoch mit dem Funktionieren der EMU-DML (vgl. S. 88) zusammenhängt.

In der Datenbankausprägung gelten die Basisnamen der im Dateisystem im Verzeichnispfad vorliegenden EMU-Etikettierungsdateien oder Signaldateien als Instanzen vom Objekt, das die Äußerungen abbildet. Welche Art von Dateien hierbei in Betracht gezogen werden (Werte in **extension**-Attributen), kann in dem EMU-Template über das **extension**-Attribut des Entitätstyps **primary** definiert werden. An dieser Stelle sei angemerkt, dass das DBMS auf diese Weise umgegangen werden kann und muss, um eine Datenbank um Äußerungen zu erweitern.

Bisher nicht betrachtet sind die modellgerechten Einschränkungen im Datenbankschema. In Abbildung 3.9 ist hierfür das interne konzeptuelle Datenmodell noch einmal mit allen Datenbankkomponenten und notwendigen Einschränkungen für einzelne Beziehungen detaillierter dargestellt. Für die Etikettierung ist im detaillierten Schema festgelegt, dass jede Ebene **LEVEL** nur maximal einmal als Elternebene (0:1 parent zu **ADJACENT**) Verwendung finden kann, dafür aber beliebig oft als Kindebene (0:N child zu **ADJACENT**). Gleiches gilt übertragen auf die Tokens in **SEGMENT** zu **ADJACENT**. Für Assoziationen in **ADJACENT** gilt wiederum, dass es von ihnen beliebig viele zwischen Ebenen aber auch Segmenten geben kann. Über diese Beschränkungen werden alle vorgestellten möglichen Etikettierungsstrukturen in Kapitel 3.1.1 modelliert aber auch auf diese eingeschränkt. Auch ausgedrückt ist die beliebige Anzahl an Segmenten auf einer Ebene, wobei jedes Segment nur zu einer Ebene gehören kann.

Weiterhin gibt es klare Modellierungen und damit auch Beschränkungen in der Hinsicht, dass eine Äußerung beliebig viele Ebenen, Segmente und Assoziationen enthalten kann aber diese jeweils nur zu einer Äußerung gehören dürfen. Mehrere Äußerungen sind mit einer Datenbank assoziiert, wobei die Äußerungen umgekehrt nur zu einer Datenbank gehören. Ähnliches gilt für Signale. Jeder Signaltyp kann nur einen *Track* in der Datenbank darstellen, wobei die Datenbank mehrere *Tracks* enthalten kann aber nicht muss. An dieser Stelle kann es in der Verwendung zu

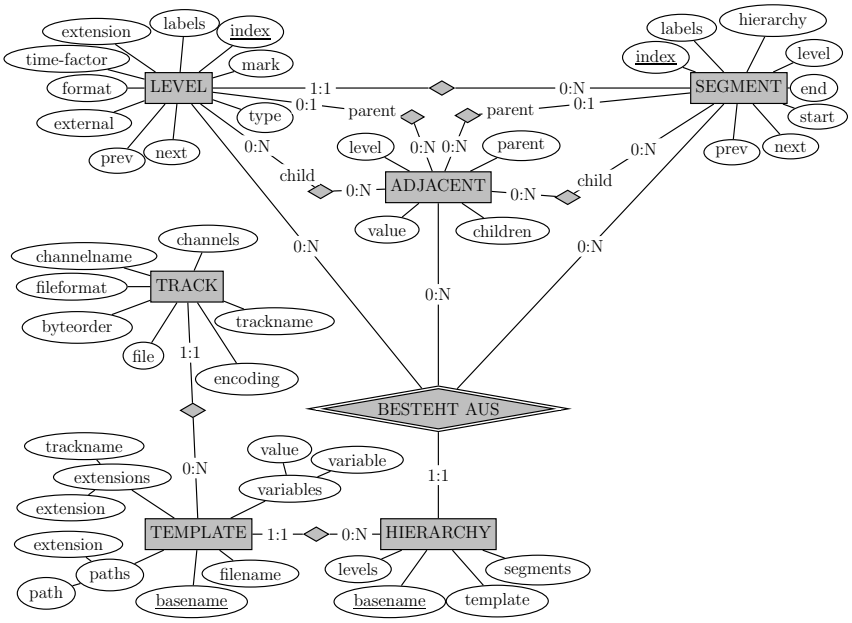


Abbildung 3.9.: Detaillierter konzeptueller Ausschnitt des internen EMU-Datenmodells als ER-Schemadiagramm mit Beziehungseinschränkungen min:max auf der Seite des betreffenden Entitätstypen und verschiedener Rollen von **LEVEL** und **SEGMENT** zu **ADJACENT**.

Problemen kommen, wenn man den gleichen Signaltyp jedoch mit unterschiedlichen Parametern vergleichend darstellen möchte. Eine Lösung hierfür ist die Deklaration der beiden gleichen *Tracks* als vermeintlich unterschiedliche, indem andere *Track*-Namen und Extensionen zugewiesen werden. Die Dateiextension der Dateien selbst muss dafür auch geändert werden.

3.2.2.2. Physisches Modell auf Dateiebene

Wie aus Cassidy und Harrington (2001) hervorgeht, sind Datenbestände im EMU-System nicht integriert, sondern als im Dateisystem sichtbare Dateien gespeichert. Dieser Umstand führt dazu, dass auf den Datenbeständen auch dateibasiert gearbeitet werden kann, was Nachteile aber auch Vorteile bietet.

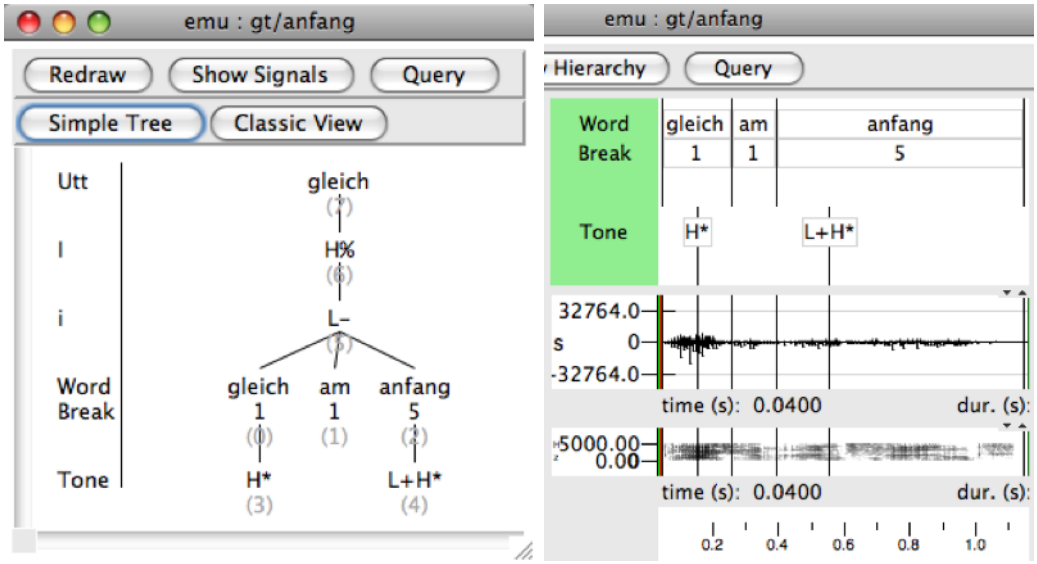
Dateien Die Zeitsignale liegen jeweils als Datei im signaltypgeeigneten Format vor. Für Sprachsignale werden überwiegend Dateien im WAV-Format (Microsoft,

2001) verwendet[14]. Wie bereits erwähnt wurde, werden parametrische Daten im EMU-System hauptsächlich im Simple Signal File Format (SSFF) verarbeitet, das in Cassidy (2011a) beschrieben wird. In diesem Format ist es möglich, den Dateiheader um beliebige Variablen zu erweitern, so dass sehr viel Information über die gespeicherten Daten eingefügt werden kann. Im SSFF kann daher jegliche Art von Daten gespeichert werden. Der Datentyp schränkt nicht auf numerische oder alphabetische Zeichen ein und bietet die Möglichkeit, Informationen in einer beliebigen Anzahl von Feldern zu speichern, die wiederum eine beliebige Größe haben können. Mit diesem Datenformat ist es also möglich, Formantdaten für die ersten vier Formanten in einem Feld unterzubringen und in einem weiterem Feld die Bandbreiten der Formanten. Wie die Daten auf die Felder verteilt sind, ist im SSFF Header angegeben. Liegen Sprachsignale im SSF-Format vor, so kann im Header zum Beispiel eine Startzeit verschieden von 0 definiert werden, wie es in den Sprachsignalen der EMU-Beispieldatenbank `demo` der Fall ist, die als Beispiel in Kapitel 8 verwendet wird. Diese variable Startzeit ist nützlich im Rahmen von Signalextraktionen aus längeren Signaldateien. Auf diese Weise können Signale wieder in der richtigen Reihenfolge zusammengesetzt werden, wenn es notwendig ist und der ursprüngliche Zeitbereich ist zu jeder Zeit bekannt. Das EMU-System ist in der Lage, die Startzeiteninformation zu berücksichtigen.

Auch die Etikettierungen liegen in einem typgeeigneten Format vor (Cassidy und Harrington, 2001). Abbildung 3.10 (S. 80) zeigt ein komplettes Beispiel einer Äußerung mit Etikettierungsdarstellungen und den Dateien. Für jede zeitgebundene Ebene gibt es jeweils eine Datei (sog. externe Etikettierungsdateien *labfiles*), die Informationen zu Tokens der Ebene durch die Attributwerte, die Etikett und Endzeit modellieren, tabellarisch im ASCII Format ablegt (vgl. Abbildung 3.10 rechts für die `Word`- und `Tone`-Ebene). Die verschiedenen Ausprägungen der Attribute **mark** und **format** des Entitätstypen **labfile** in Abbildung 3.7 geben vor, ob Start- End- oder beide Zeiten gespeichert sind.

Weiterhin gibt es eine Datei (*hlb*-Datei, vgl. Abbildung 3.10 links), die einerseits alle Tokens aller Ebenen enthält, jedoch nicht mit den Ausprägungen der Zeitattribute, sondern ausschließlich mit der Token-ID aus den **index**-Ausprägungen (Ausprägung des Schlüsselattributes) und den Etiketten (**labels**) und andererseits die Verknüpfungen zwischen den Tokens in einer Tabellenform, in der jede Reihe eine Token-ID und die IDs der Tokens, die als Kinder assoziiert sind, in Spalten angibt.

[14] Unter Berücksichtigung aller Datenbankapplikationen bietet das EMU-System auch die Unterstützung für AU, SND, AIFF und CSL.




```
**EMU hierarchical labels**
7
Utt Utt
7 gleich

I I
6 H%

i i
5 L–

Word Word Break
0 gleich 1
1 am 1
2 anfang 5

Tone Tone
3 H*
4 L+H*


0 3
1
2 4
3
4
5 0 1 2 3 4
6 0 1 2 3 4 5
7 0 1 2 3 4 5 6

0
```

```
signal anfang
nfields 1
#
        0.045942        125     H#
        0.255965        125     gleich
        0.393793        125     am
        1.150750        125     anfang
```

```
signal anfang
nfields 1
#
        0.153411        125     H*
        0.553167        125     L+H*
```

Abbildung 3.10.: Etikettierung im EMU Labeller der Äußerung `anfang` der in EMU frei verfügbaren `gt`-Datenbank mit der Ansicht der hierarchischen Etikettierungsstruktur (links) und der zeitgebundenen Etikettierung (rechts) und EMU-Etikettierungsdateien der Äußerung mit der *hlb*-Datei (links) und den externen Etikettierungsdateien (rechts) für die `Word`-Ebene (oben) und die `Tone`-Ebene darunter.

Das physische Schema (die Dateien und deren Struktur) wird zur Laufzeit des Datenbanksystems um Speicherstrukturen im transienten Speicher erweitert, wie bereits durch den Ausschnitt des internen Schemas in Abbildung 3.8 deutlich wurde. Das heißt, die Dateien werden eingelesen und die Informationen in programminternen, nur zur Laufzeit des Systems vorhandenen Objekten gespeichert. Hierfür sei auf den Quellcode (EMU Developers, 2011a), der frei zugänglichen Implementierung des EMU-System verwiesen, aus dem sich diese Strukturen lesen lassen.

Vor der Weiterentwicklung des EMU-Systems im Rahmen dieser Dissertation wurden die Datenbankschemata verschiedener Datenbanken zentral innerhalb des Speicherbereichs des Systems selbst gemeinsam und somit von der Datenbasis getrennt gespeichert. Ein Datenbankspeicherort war und ist im EMU-core (vgl. Kapitel 1.3) für jede Plattform individuell fest als EMU-Templatepfad implementiert.

Diskussion des Ansatzes Datenbankschemata getrennt von der Datenbasis zu speichern, hatte den Vorteil, dass sämtliche Datenbankschemata dem System automatisch bekannt waren, sobald sie in Form einer Datei am entsprechenden Speicherort abgelegt wurden. Die Nachteile dieses Vorgehens ergaben sich unter Betrachtung von Mehrbenutzersystemen. Auf diese Weise musste die EMU-Software auch auf Mehrbenutzersystemen für jeden Benutzer separat installiert werden. Zudem musste der Nutzer Zugriffsrechte auf den Speicherbereich des Systems haben, um Zugriff auf Datenbankschemata zu erhalten, denn sie werden vom Benutzer selbst festgelegt. Dieser Nachteil wurde im Laufe der Weiterentwicklung behoben, indem automatisch jedes Benutzerverzeichnis für die Speicherung der Datenbankschemata vorgesehen wurde. Durch diese Erweiterung waren die Grundsteine für den heutigen Stand gelegt, der benutzerdefinierte Speicherorte ermöglicht. Das Datenbankschema sowie die Datenbank können durch die Weiterentwicklung in dieser Dissertation (vgl. Kapitel 5.1) an jedem beliebigen Ort abgelegt werden, welcher vom System über native Dateipfade erreichbar ist. Das EMU-System ist derzeit nicht fähig, UNC-Pfade (Doragh, 1994) wie //server/home zu verarbeiten. Die Datenbankausprägung wird vom DBMS über die Pfadangaben im Datenbankschema ermittelt. Orte für Datenbankkataloge sind nun im EMU-System konfigurierbar.

Der physische Speicherort der Dateien wird vom Benutzer selbst festgelegt, wie aus Kapitel 3.2.2 hervorgeht. Somit unterliegt die Zugriffsverwaltung dem Betriebssystem bzw. dem Benutzer, der für seine Datenbank administrative Aufgaben übernimmt.

In der heutigen Zeit werden Zugriffsbeschränkungen zunehmend wichtiger. Die zu bearbeitenden Datenmengen werden umfangreicher, so dass wissenschaftliche Unternehmungen in der Linguistik und Phonetik zunehmend in größeren Forschergruppen unternommen werden müssen. Die Aufbereitung der Rohdaten liegt nicht mehr in der Hand des Einzelnen, sondern wird von mehreren Forschern bewältigt. Mit der Integration der Datenbestände in das System könnte die Datenzugriffsverwaltung dem Datenbankmanagement als Aufgabe übertragen werden, wie es für Datenbanksysteme vorgesehen ist. Das EMU-System für die Datenintegration zu implementieren ist eine lösbare Aufgabe und könnte in Angriff genommen werden. Jedoch müsste für eine mögliche Interoperabilität mit anderen rein dateibasierten Programmen bei einer vollständigen Datenintegration eine zusätzliche Exportfunktion zur Verfügung gestellt werden. Im Rahmen der Austauschbarkeit von Datensätzen zwischen verschiedenen Programmen könnte diese Funktion für unterschiedliche Formate zur Verfügung gestellt werden.

Aus einem anderem Blickwinkel betrachtet, zeigen sich Vorteile, wenn der Benutzer den physischen Speicherort der Datenbasis selbst definiert und dadurch kennt. So können Datenbestände ohne Umweg eines Exports direkt ausgetauscht werden. Zudem sind die Daten aus Benutzersicht transparent. Sie sind als Dateien sichtbar, was mit einer psychologisch positiven Wirkung, einer Art Sicherheit, einhergeht. Eine solche psychologische Wirkung ist zur Zeit noch nicht von der Hand zu weisen, denn der Benutzer ist es noch gewöhnt, mit Dateien zu arbeiten.

Ein weiterer Vorteil bzw. eine Notwendigkeit der transparenten Dateispeicherung ist auf technischer Seite zu finden: Das EMU-System bildet eine Station in den Arbeitsabläufen der Benutzer. Während der Forschungsarbeit verwenden diese verschiedene Programme, um die Daten aufzubereiten und zu verarbeiten. Somit müssen auch Programme auf die Datenbasis zugreifen können, für die das DBMS entweder noch keine Schnittstelle zur Verfügung stellt oder aus technischen Gründen keine Schnittstelle zur Verfügung stellen kann. Letzteres ist der Fall, wenn die jeweilige Software keine Kommunikationsschnittstelle zu anderen Programmen zulässt. In beiden Fällen ist der Benutzer gezwungen, auf Dateien zuzugreifen, um diese in anderen Applikationen manuell öffnen oder laden zu können. Diese Möglichkeit würde durch die Integration der Daten in das System jedoch nicht unterbunden werden. Eine gleichzeitig zur Verfügung gestellte Exportfunktion würde auch hierfür die Daten als Dateien sichtbar machen. An dieser Stelle müsste eine Aufwand-, Nutzen-, Kosten-, Benutzerfreundlichkeitsanalyse entscheiden, ob die Integration der Datenbasis in das System sinnvoll ist. Da sich neuere Datenbanksysteme mit dieser Funktion durchge-

setzt haben und die Benutzer dieser Art von Intransparenz immer häufiger im Leben ausgesetzt sind, weswegen sie sich daran gewöhnen, sollte auch für das EMU-System dieses Ziel angestrebt werden. Neuere Technologien der Datenspeicherung könnten auf diese Weise sofort ältere Methoden ersetzen, ohne dass sich Änderungen für den Benutzer in der Softwareanwendung ergeben, da über das DBMS eine Datenunabhängigkeit (vgl. Kapitel 2.1.3) zwischen externer und interner Ebene gewährleistet ist. So könnten die Etikettierungen in Tabellen für ein relationales Datenbankmodell (vgl. Kapitel 2.1.2) umgewandelt werden, wie in Cassidy (1999a) vorgeschlagen wurde und in Kapitel 8 erläutert wird. Ein anderes Vorgehen wäre für Zeitsignale zu entwickeln. Zum einen können Signale nicht als Datenbanktupel dargestellt werden und zum anderen ist der Nutzen nicht offensichtlich. Eine grundlegende Funktion eines Datenbanksystems ist die Datenbankabfrage zum schnellen Zugriff auf Informationen aus der Datenbasis. Eine Abfrage auf Signale kann nur darin bestehen, Abtastwerte abzufragen.

3.2.3. Datenbankmanagementsystem

Das Datenbankmanagementsystem in EMU stellt eine DDL (Data Definition Language) zur Verfügung, womit das Datenbankschema im Template beschrieben wird. Die Erstellung und Veränderungen des Datenbankschemas werden nicht sukzessive über die DDL vorgenommen. Stattdessen wird das Schema als Ganzes bearbeitet, indem ein Dateiinhalt verändert und dieser in die Template-Datei geschrieben wird. Die Datei wird als Datenbankschema in das DBMS geladen, sobald eine Datenbank geladen werden soll. Diese Funktionsweise macht es unmöglich, Datenbankschemata über das DBMS zu erzeugen und während der Laufzeit zu verändern. Es kann nur dateibasierte Definitionen beim Laden der Datenbank auf Zulässigkeit prüfen und ggf. ablehnen.

3.2.3.1. DDL in EBNF

Direktive Anweisungen wie in der Relationenalgebra sind in EMUs DDL nicht zu erwarten. Die DDL Anweisungen haben ähnlich der SQL deklarativen Charakter. Cassidy und Harrington (2001) vergleichen die DDL mit einer DTD (*document type declaration*) in SGML (Goldfarb und Rubinsky, 1990, zitiert in Cassidy). Folgender kommentierter Ausschnitt des EMU-Datenbanktemplates aus John (2004),

das bereits in Kapitel 3.1 im Relevanzbeispiel verwendet wurde, zeigt ein Beispiel (Kommentar = '!'):

```
! Deklaration Ebenen durch ihren Namen
level VP

! weitere Ebenen mit zusätzlichen Attributen, die dominierende Ebene
! und die fakultative Art der Assoziation
level Zitierform VP many-to-many
level Phone Zitierform
level Stimmhaft Phone

! Deklaration eines zweiten Etiketttyps für die Ebene Wort
label VP Herkunft

! Zeichenmenge der Ebene Phonetic für die Klasse vowel
legal Phone Vokal V
legal Phone Plosiv t d

! Attribut der Art der Zeitgebundenheit für die Ebene Phonetic
! Dateiextension lab
! Angabe der Genauigkeit für die Speicherung der Zeitangaben
labfile Phone:type SEGMENT:extension phone:time-factor 1000
labfile Stimmhaft:type EVENT:extension sthaft:time-factor 1000

! Deklaration der Speicherorte nach Dateiextensionen getrennt
path hlb John2004
path phone John2004
path sthaft John2004
path wav John2004

! Deklaration der Sprachsignaldateien
track samples wav

! Konfiguration für die Äußerung-Objekte in der Datenbankausprägung
set PrimaryExtension hlb
```

Deklarationen[15] werden für jeden Entitätstypen im ER-Templateschema in Abbildung 3.7 vorgenommen: *level* für die Deklaration der Ebenen, *label* für die Label-Links, *labfile* für die zeitgebundenen Ebenen, *path* für die Deklaration der Speicherorte, *track* für die Signale und *set* für die Deklaration des **primary**-Entitätstypen (vgl. S. 77) sowie für Variablen.

Für jede Entität sind die Attribute des Typs nur über die Attributwerte in einer festen Reihenfolge angegeben. Für **labfile**-Entitäten jedoch können die Attributwerte nicht ohne explizite Angabe des Attributs stehen, da einige von ihnen fakultativ

[15] Während in diesem Kapitel explizit aufgrund des Kontextes <Deklaration> verwendet wird, werden in anderen Kontexten <Deklaration> und <Definition> synonym verwendet.

sind[16]. Für **level**-Entitäten werden die Attributwerte nur angegeben, wenn sie vom Standardwert abweichen, das gilt für Eltern-Kind Beziehungen zwischen den Ebenen, wenn diese gegenüber des Standardwertes[17], der one-to-many ist, als many-to-many definiert werden.

Für die verschiedenen Attributausprägungen stellt das DBMS spezielle Verarbeitungsmethoden bereit. Zu den vordefinierten *Track*-Namen gehören dft, css, lps, cep für spektrale Daten, samples für Sprachsignale, fm für Formanten und F0 für die Grundfrequenz. So wird z. B. für samples-*Tracks* eine hohe Abtastrate angenommen und die Daten entsprechend anders verarbeitet als F0-Signale, die als Signale mit geringerer Abtastrate angenommen werden. Derartig vordefinierte Ausprägungen mit gesonderter Bedeutung sind auch für die Variablen und Formate vorzufinden.

Aufgrund des gegebenen Beispiels und der Tatsache, dass diese Template-Datei als externes Datenbankschema vom Benutzer selbst definiert wird, ist es vorstellbar, dass potentielle Nutzer der Software, wie Phonetiker und Linguisten, die mit derartigen formalen Beschreibungen gewöhnlich nicht tagtäglich umgehen, bereits beim Erstellen des Schemas unter der Verwendung der DDL Probleme haben könnten. Aus diesem Grund wurde im Rahmen der Dissertation eine grafische Lösung für die Deklaration entwickelt. Diese Applikation (vgl. Kapitel 4.3) stellt grafische Eingabefelder gemäß der Entitäten im Templateschemadiagramm auf S. 73 zur Verfügung und erzeugt aus den Angaben des Benutzers DDL Ausdrücke.

Neben dem Templateschemadiagramm, welches einen Überblick über vorhandene Beziehungstypen im EMU–Datenmodell gibt, wurde in der Vorbereitung die DDL formal beschrieben. Anhand dieser Beschreibung wird somit zusätzlich dokumentiert, wie Ausprägungen des Schemas syntaktisch wohlgeformt durch den Benutzer aber auch semi-automatisch (vgl. Kapitel 4.3.4) definiert werden müssen. Als Beschreibungssprache wurde die erweiterte Backus-Naur Form (vgl. Kapitel 2.3) mit einem informelleren Grad der Beschreibung gewählt, in der nicht alle *Meta-Identifier* vollständig formal dargestellt sind, sondern deren Bedeutung über Kommentare ausgedrückt ist[18]. In der Beschreibung wurden alle bekannten vordefinierten DDL-Ausdrücke berücksichtigt. Für die Semantik einzelner Ausdrücke, die in dieser Arbeit nicht berücksichtigt wurden, sei auf die veraltete aber in Hinblick auf die DDL fast aktuelle Dokumentation von Cassidy (2004) verwiesen. Die formale Beschreibung

[16] Fakultative Attribute sind mark und format, die für den Import systemfremder Etikettierungsdateien vorgesehen sind.
[17] Der Wert, der für die Beziehung vom DBMS erwartet wird.
[18] Für die Möglichkeit informellerer Beschreibungen vgl. Kapitel 2.3

der DDL ist auf der nächsten Seite abgebildet. Hierbei sind die Produktionsregeln in einer *bottom-up* Reihenfolge zu lesen.

```
(* EBENE, ETIKETTIERUNG, CLASS - freiwählbare Zeichenkette A-Z0-9 *)
(* NAME - freiwählbare Zeichenkette, bei Dateien im SSFF jedoch vorgegeben *)
(* EXTENSION - freiwählbare Zeichenkette, sollte den Dateiextensionen der
Signal- und Etikettierungsdateien entsprechen *)
(* PFAD - freiwählbare Zeichenkette, sollte dem Dateipfad entsprechen, wo die
Dateien gespeichert sind bzw. gespeichert werden sollen *)
(* LF ist ein Zeilenwechsel *)
LEVEL = 'level',' ',EBENE,[(' ',EBENE,[(' ','X2MANY')])],LF;
X2MANY = 'many-to-many' | 'one-to-many' | '';
LABEL = 'label',' ',EBENE,' ',EBENE,LF;
LEGAL = 'legal',' ',EBENE,' ',CLASS,' ',ETIKETTIERUNG,LF;
LABFILE = 'labfile',' ',EBENE,TYPE,EXT,TIMEF,MARK,FORMAT,LF;
TYPE = ':type',' ',(SEGMENT | EVENT);
EXT = ':extension',' ', EXTENSION;
TIMEF = ':time-factor',' ',REALP;
REALP = INTPN,['.', {DIGIT}];
INTPN = '0' | INTP;
INTP = DIGIT-'0',{DIGIT};
DIGIT = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';
MARK =':mark',' ',('START' | 'END');
FORMAT = ':format',' ',('ESPS' | 'SPEECHSTN' | 'ACCOR' | 'TIMIT' | 'KIEL');
HLBPATH = 'path',' ','hlb',' ',PFAD,LF;
PATH = 'path',' ',EXTENSION,' ',PFAD,LF;
KNOWNTRACK = 'samples' | 'F0' | 'fm' | 'dft' | 'css' | 'lps' | 'cep';
TRACK = 'track',' ',( KNOWNTRACK | NAME ),' ',EXTENSION,LF;
FILE = PFAD,NAME,'.',EXTENSION;
VIEW = 'HierarchyViewLevels' | 'LabelTracks',' ',EBENE;
SPEC = ('SpectrogramScale'|'SpectrogramWhiteLevel'|'MaskThreshold'),' ',REALP;
SCRIPT = ('AutoBuild',' ',FILE) | ('EmulabelModules',' ',FILE,' ',NAME);
VAR = 'set',' ', VIEW | SPEC | SCRIPT | ('FormantTrack' | NAME,' ',{NAME}),LF;
PRIM = 'set',' ','PrimaryExtension',' ',(EXTENSION | 'hlb'),LF;
TPLTRACK = TRACK,PATH,[PRIM],{VAR};
TPLLAB = LABFILE,PATH;
TPLLEV = {LEVEL}- ,HLBPATH,{LABEL},{LEGAL},{TPLLAB},{TPLTRACK},[PRIM],{VAR};
TPL = TPLLEV | TPLTRACK;
```

3.2.3.2. DML

Im Gegensatz zur DDL wird die DML im EMU-System durch Tcl-Kommandos repräsentiert. Für die Anfrage an die Datenbank stehen unterschiedliche Kommandos für das Datenbankschema, die Etikettierungs-Datenbasis und die Signal-Datenbasis zur Verfügung. Das Datenbankschema kann mithilfe so genannter `emutemplate` Befehle eingesehen werden, für die Etikettierungskomponenten stehen `hierarchy` Befehle zur Verfügung und für die Signale, Befehle aus der `snack` Bibliothek. Während letztere unabhängig sind, sind `hierarchy` Befehle Unterkommandos von `emutemplate` Befehlen.

Im Folgenden werden in einem kommentierten Tcl-Skript (*Befehle*, Parameter, **Rückgaben**, #=Kommentar) einzelne DML Befehle beispielhaft dargestellt. Neben den im Beispiel angegebenen Operationen können alle Informationen, die in der Datenbankbeschreibung enthalten sind, angefragt werden. Das gilt beispielsweise auch für sämtliche Attribute der einzelnen Ebenen, Art der Signale, Speicherorte usw. Für eine Gesamtbeschreibung der DML Befehle sei auf die EMU-Hilfe (Cassidy, 2011b) verwiesen. Auf eine formale Beschreibung der DML konnte für die Weiterentwicklung verzichtet werden. Sie wurde zwangsläufig als Kommunikationsmittel zwischen Datenbank-Applikation und DBMS in der Implementierung verwendet und musste daher erlernt werden. Jedoch wurden keinerlei grafische Oberflächen oder Funktionen entwickelt, die selbst DML Ausdrücke erzeugen müssen.

Die DML kann vom Benutzer in Skripten und EMU Labeller Modulen verwendet werden. Dem besseren Zugang des Benutzers zu diesen Möglichkeiten wurde sich im Rahmen dieser Dissertation ebenfalls gewidmet (vgl. Kapitel 5.3).

```
# Zugriff auf den Datenbankkatalog
# Namen aller EMU-Templates zurück
emutemplate
demo xassp2emu manuals KCspont KCspontKIM

# Verzeichnisse, in denen EMU-Templates gespeichert sind
emutemplate info
/Library/Emu /Users/xxx/Library/Emu /Users/xxx/dbs/KCspontKIM

#Laden einer Datenbank (hier demo) in das laufende System
emutemplate db demo

#Datenbankausprägungen für Äußerung
db utterances
msdjc001 msdjc002 msdjc003 msdjc004 msdjc005 msdjc006

# Sicht auf das Datenbankschema - Anfrage der definierten Ebenen
db getlevels
```

```
Utterance Word Syllable Phoneme Phonetic Foot Nuclear

#Laden der gesamten Datenbankausprägung, die mit der Äußerung in Beziehung steht
db hierarchy utt msdjc002

#Segment-IDs der Segmente auf der Ebene Phonetic
utt segments Phonetic
0 6 8 10

#Etikett des Segments
utt seginfo 8 label Phonetic
d

#Start- und Endzeit des Segments
utt seginfo 8 times
6244.550000 6308.550000
```

Neben DDL und DML ist das Datenbanksystem auch mit einer Abfragesprache ausgestattet. Aufgrund der besonderen möglichen Etikettierungsstrukturen, findet bisher keine standardisierte Abfragesprache Anwendung im EMU-System. Die speziell auf das Etikettierungsmodell (vgl. Kapitel 3.1) zugeschnittene Abfragesprache ist die EMU-Query Language, die gesondert in Kapitel 3.3 betrachtet wird.

Für die Suche selbst werden für jede einzelne Äußerung die Etikettierungsdateien nacheinander eingelesen. Anschließend werden die Tokens auf allen Ebenen nach und nach auf die in der Abfrage spezifizierten Eigenschaften geprüft. Bei diesem Vorgehen erfolgt für jede Äußerung mindestens der Zugriff auf die *hlb*-Datei (vgl. Kapitel 3.2.2.2) bis alle Tokens gefunden wurden. Bei zeitgebundenen Ebenen werden weitere Dateizugriffe notwendig, nämlich Zugriffe auf die externen Etikettierungsdateien (*labfile*s), um die Zeitmarken zu extrahieren. Jeder Dateizugriff ist mit rechenaufwändigen Dateiöffnungs- und Dateileseoperationen verbunden. Diese Art der Suche ist durch das Datenmodell bedingt. Eine effizientere Lösung stellt Cassidy (1999a) vor. Dieser Ansatz wird in Kapitel 8 erläutert.

Das Ergebnis einer Suche wird durch eine Sicht (vgl. Kapitel 2.1, S. 29) auf die Daten, durch eine Segmentliste (Harrington, Cassidy, John und Scheffers, 2003; Harrington, 2010a) bzw. “a list of the tokens” Cassidy und Harrington (2001, S. 63) präsentiert. Die Segmentliste ist eine vierspaltige Tabelle aus Etikett, Startzeit, Endzeit und Äußerung für jedes gefundene Segment bzw. Token (pro Reihe ein Segment). Bei sequentiellen Abfragen besteht das Etikett aus einer Verknüpfung durch -> der Etiketten der Segmente, die in der Sequenz vorkommen. Start- und Endzeiten werden durch das DBMS abgleitet und ergeben sich jeweils aus dem Startzeitpunkt des ersten Segments und Endzeitpunkt des letzten Segments. Für Segmente auf zeitlosen

Ebenen werden Start- und Endzeiten soweit möglich aus den assoziierten Kindsegmenten abgeleitet. Diese Kindsegmente müssen eine sequentielle Anordnung haben, so dass die Startzeit wiederum vom ersten Kind abgeleitet wird und die Endzeit vom letzten Kind (vgl. Abbildung 3.2, S. 57).

3.3. EMU Query Language in EBNF

Die EMU Query Language (EQL) ist Teil des EMU Speech Database Systems und stellt die Abfragesprache im Datenbankmanagementsystem dar. In der Literatur (Cassidy und Harrington, 1996; Cassidy und Bird, 2000; Cassidy und Harrington, 2001; Harrington, 2010a) seit den 90er Jahren des 20. Jhdt. wird die EMU Query Language exemplarisch beschrieben. Diese Beschreibungen zeigen anhand von Beispielabfragen auf, welche Assoziationen in EMU-Etikettierungsstrukturen abfragbar sind, um zu zeigen, wie mächtig die Abfragesprache ist. Wie aus Kapitel 2.3 jedoch hervorgeht, definieren sich Sprachen über ihre möglichen Ausdrücke, die mithilfe einer Grammatik bzw. über die Beschreibung der syntaktischen Regeln abgebildet werden können. Die Darstellungen der EQL anhand von Beispielabfragen allein kann nicht alle Ausdrücke der Sprache enthalten. Im Folgenden wird ebenfalls die EQL beschrieben, jedoch auf die in der Literatur fehlenden formalen Art und Weise. Im Rahmen der Dissertation musste diese formale Beschreibung vorgenommen werden, um sich bei der Implementierung des grafischen Interfaces zur Abfragesprache daran orientieren zu können (vgl. Kapitel 4.2). Weiterhin werden die Schwächen der EQL diskutiert und die im Rahmen der Dissertation entwickelten Lösungsansätze gezeigt.

In der Phonetik werden Datenbankabfragen durchgeführt, um zum einen die Existenz von phonetischen Phänomenen in einer Datenbank zu belegen, beispielsweise Aspiration von Lauten in Sprache X, von der Sprachaufnahmen in der Datenbank enthalten sind. Zum Anderen werden Informationen aus der Datenbank extrahiert, wie die zeitliche Position von etikettierten Bereichen, um das assoziierte Signal zu untersuchen, beispielsweise Formantdaten von als Vokal etikettierten Bereichen. Weiterhin werden Datenbanken abgefragt, um den Einfluss der Wort-, Satz- oder Phrasenstruktur als auch der Position eines Tokens im Wort, Satz oder der Phrase auf verschiedene Phänomene hin zu untersuchen. Beispiele hierfür wären die phrasenfinale Längung von Phonen oder das stetige Absinken der Grundfrequenz über eine Phrase hinweg (Deklination). Das Etikettierungsschema und die Etikettierungen selbst bestimmen, welche Abfragen möglich sind. Daher muss bereits bei der konzeptuellen Modellierung des Schemas darauf geachtet werden, dass alle Informationen enthalten sein können, die später abgefragt werden sollen. Um also die Wortstruktur analysieren zu können, muss diese auch in den Etikettierungen enthalten sein, zum Beispiel über die Etikettierung auf Wort-, Silben- und Phonebene.

Das Etikettierungsmodell, das im EMU-System verwendet wird, wurde in Kapitel 3.1.1 beschrieben. Die Funktion der EQL ist es also, die beschriebenen Etikettierungsstrukturen[19] abfragbar zu machen, um Etiketten zu finden, die spezielle Eigenschaften in der Struktur erfüllen. Die Abfrage muss zunächst ermöglichen, die Etiketten auf einer Ebene zu suchen und zu finden. Sie muss somit:

1. einfache Abfragen

gewährleisten.

Die Etiketten auf verschiedenen Ebenen können in unterschiedlichen Beziehungen zueinander stehen. Diese Beziehungen sollten als Einschränkungen für die Suchanfrage formuliert werden können, d. h. Abfragemöglichkeiten für Etiketten in:

2. linearen Beziehungen
3. hierarchischen one-to-many Beziehungen
4. autosegmentellen one-to-many Beziehungen
5. hierarchischen many-to-many Beziehungen
6. autosegmentellen many-to-many Beziehungen

müssen zur Verfügung stehen. Die Etiketten auf einer Ebene sind sequentiell angeordnet. Die Reihenfolge der Etiketten kann ebenfalls ein Suchkriterium/Einschränkungskriterium für die zu suchenden Etiketten darstellen, daher sollen auch:

7. sequentielle Abfragen

möglich sein. Die genannten Abfragenotwendigkeiten leiten sich aus dem Etikettierungsschema ab. Die EMU Query Language ist darauf angepasst, sodass durch die Verwendung der EQL die oben genannten Abfragen (1.–7.) aber auch die folgenden Suchfunktionen (8. und 9.) formuliert werden können.

8. Positionsabfragen
9. Anzahlabfragen

Die EMU Query Language ist als Abfragesprache eine formale Sprache (vgl. Kapitel 2.3), deren mögliche Ausdrücke mit einer gewissen Semantik verbunden sind. ‚Abfrage' wird im weiteren Verlauf synonym mit ‚Abfrageausdruck' verwendet.

[19] Das Abfragen vom Zeitsignal selbst ist nicht Aufgabe der EQL - die momentane Methode in der phonetischen Forschung ist forschungsrelevante Ereignisse im Zeitbereich in der Etikettierung zu markieren. Damit sind die Bereiche schneller wieder zu finden und auch für andere Forscher bereits hervorgehoben.

Ergebnisse der Abfrage werden sowohl in der EMU-Applikation EMU Query Tool als auch in der Abfrage über die EMU/R Schnittstelle in R in Form von Segmentlisten dargestellt (vgl. S. 89).

3.3.1. Alphabet, Syntax und Semantik

Im Folgenden werden Alphabet, Syntax und Semantik der EQL vorgestellt. Hierfür werden die terminalen Symbole der EQL aufgelistet und mögliche Ausdrücke mithilfe der erweiterten Backus-Naur Form (vgl. Kapitel 2.3) zum Aufzeigen der Syntax beschrieben. Weiterhin werden die Semantik und die damit verbundenen Einschränkungen erklärt, als auch mithilfe von Datenbankabfragen und deren Ergebnissen die jeweilige Abfrage exemplarisiert. Als Beispieldatenbank wird das in das EMU-Format überführte Kiel Corpus verwendet. Diese Konvertierung wird in Kapitel 7 beschrieben. Die Produktionsregeln (im Folgenden EQ) werden durch Beispiele erzeugbarer Ausdrücke für die Kiel Corpus Datenbank ergänzt (im Folgenden EQKC). Abfrageergebnisse werden zum Teil als Segmentlisten dargestellt. <Token> und werden im weiteren Verlauf synonym verwendet und bezeichnen jeweils die zu suchende oder gefundene Einheit.

Die Suche in einer EMU-Abfrage setzt das Suchvorhaben voraus, ein Token zu finden, das auf einer Ebene etikettiert wurde, somit ein Etikett und verschiedene andere Eigenschaften besitzt und ggf. bestimmten Auswahlkriterien unterliegt. Das Etikett und auch die Position des etikettierten Tokens sind bereits Auswahlkriterien. Auswahlkriterien können jedoch auch die Anzahl der assoziierten Tokens sein, sowie deren Etikett. Abfragen finden innerhalb der Baumstruktur der Etikettierung statt und zwar zunächst in vertikaler Richtung. In Abbildung A.3 wäre zum Beispiel eine Abfrage der Tokens der `Word`-Ebene, die mit Tokens auf der `Syllable`-Ebene und weiterhin mit bestimmten Tokens auf der `Phonetic`-Ebene assoziiert sind, formulierbar. Nur Positions- und Anzahlabfragen sind inhärent horizontale Suchmuster, so dass im Beispiel die Reihenfolge von Wörtern auf der `Word`-Ebene abgefragt werden kann.

Für die Abfrage selbst können die Ebenennamen aus der Etikettierungsstruktur und die Zeichen aus der Etikettierung mit verschiedenen terminalen Symbolen der EQL, den Operatoren, Klammern und Funktionen unter bestimmten syntaktischen Bedingungen miteinander verbunden werden, um Abfrageausdrücke zu formulieren. In den Tabellen 3.1–3.3 sind die Terminale der EQL aufgelistet. Zusätzlich sind alle

Buchstaben, Zahlen und nicht erwähnte Sonderzeichen ebenfalls terminale Symbole der EQL, die jedoch keine EQL spezifischen Bedeutungen haben und daher nicht aufgelistet sind. Somit gilt: Buchstaben und Zahlen repräsentieren sich selbst. Für Ebenennamen und Zeichen in der Etikettierung werden in der folgenden EBNF Notation keine Produktionsregeln angegeben. Sie werden direkt und ohne Ableitungsregel mit den *Meta-Identifier*n EBENE und ETIKETTIERUNG bezeichnet. Die möglichen Ebenennamen, die mit EBENE gemeint sind, sind die in der EMU-Etikettierungsstruktur definierten Ebenen der Datenbank. Eine ETIKETTIERUNG ist an sich eine freiwählbare Zeichenkette.

Tabelle 3.1.: Terminale Symbole der EQL (Operatoren) und ihre Bedeutung nach absteigenden Bindungsgrad (Priorität).

Operator	Bedeutung
`#`	Ergebnismodifizierer
`,`	Parameterlistentrenner
`=`	Gleichheit
`!=`	Ungleichheit
`>`	mehr als
`>=`	gleich und mehr als
`<`	weniger als
`<=`	gleich und weniger als
`|`	Alternative
`&`	gleichrangige Konjunktion
`^`	Dominanzkonjunktion
`->`	Sequenzoperator

Tabelle 3.2.: Terminale Symbole der EQL (Klammern) und ihre Bedeutung.

Klammer	Bedeutung
`'`	Sonderzeichenanfangs- und Endklammer
`(`	Parameterlistenanfangsklammer
`)`	Parameterlistenendklammer
`[`	Sequenzabfragenanfangsklammer
`]`	Sequenzabfragenendklammer

Eine mögliche Formulierung einer Suche, die alle Abfragen verbindet wäre: *Suche alle zweisilbigen Wörter am Anfang einer prosodischen Phrase, die in der Zitierform einen Schwa-Vokal enthalten, der getilgt ist.* Ausgehend von einem aus linguistischer

Tabelle 3.3.: Terminale Symbole der EQL (Funktionen) und ihre Bedeutung.

Funktionskopf	Bedeutung
`Start`	Anfang
`Medial`	Medial
`End`	Final
`Num`	Anzahl

Sicht intuitiven Etikettierungsschema:
Phrase(-) → Wort(-) → Silbe(-) → Kanonik(-) = Elision(-) → Phon(S)
ist der Abfrageausdruck in EQKC 1 (S. 96) zu erwarten.

Das Kiel Corpus im aufbereiteten EMU-Format hat ein ganz ähnliches Etikettierungsschema wie in Abbildung 3.11 weiter unten zu sehen sein wird. Diese Abfrage auf das Kiel Corpus angewandt, werden 447 Tokens in einer Segmentliste zurückgegeben. Ein Segment, das der Abfrage entspricht, befindet sich in Äußerung g103a010 des Kiel Corpus. Es ist das phraseninitiale zweisilbige Wort <nehmen>, in dem [ə] getilgt ist. Die Phrase, in der das Wort vorkommt, ist in Abbildung 3.11 dargestellt. Die Tilgung des [ə] ist auf der Ebene `SinfoKan`, einem Label-Link zur `Kanonic`-Ebene, mit einem `E` für Elision markiert.

Wird die Abfrage auf: *Suche alle zweisilbigen Wörter am Anfang einer prosodischen Phrase, die in der Zitierform einen Schwa-Vokal enthalten, der getilgt ist, der jedoch nicht vor einem alveolaren Nasal vorkommt.* wie in EQKC 2 eingeschränkt, so werden nur die fünf Tokens (vgl. Tabelle 3.4) gefunden. Das erste Token in der Tabelle ist zusammen mit der gesamten Phrasenetikettierung in Abbildung A.3 abgebildet. Aus dieser rein quantitativen Beobachtung heraus kann man also feststellen, dass /ə/ in einer Folge /ən/ in phraseninitialen zweisilbigen Wörtern sehr oft getilgt wird.

In den beiden Abfragen EQKC 1 und 2 ist nahezu das gesamte Zeicheninventar der EQL, bis auf ein paar Vergleichsoperatoren und Attributen zu Funktionen, gegeben. Es ist überschaubar und dennoch für derartig komplexe Abfragen ausreichend.

EQKC 1.

```
[ Word != g4d6j7 & Start(Phrase,Word)=1 & Num(Word,Syllable)=2
^
Kanonic = @ & SinfoKan = E ]
```

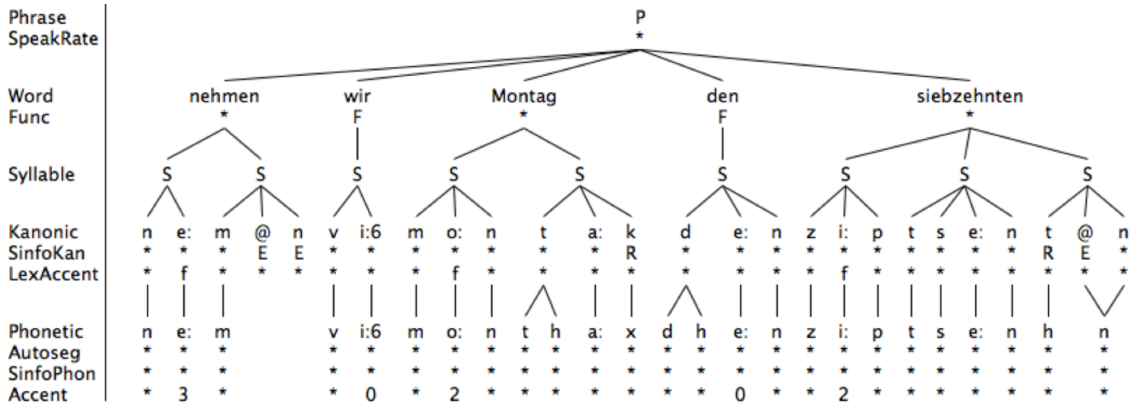


Abbildung 3.11.: Ausschnitt der etikettierten Kiel Corpus Äußerung g103a010 mit einem zweisilbigen phraseninitialen Wort, in dem gegenüber der Zitierform [ə] getilgt ist.

EQKC 2.

```
[ Word != g4d6j7 & Start(Phrase,Word)=1 & Num(Word,Syllable)=2
^
[ Kanonic = @ & SinfoKan=E -> Kanonic != n ]]
```

Tabelle 3.4.: Ergebnis von EQKC 2.

Etikett	Startzeit	Endzeit	Äußerung
freies	3453.5	3764.9370	g11a:g112a:g112a012
seinem	2181.188	2597.25	l01a_l06a:l02a:l021a047
hattest	35.31200	270.0	l01a_l06a:l04a:l041a018
kuschel	21234.25	21561.25	l01a_l06a:l05a:l051a001
hattest	521.938	726.0	l01a_l06a:l05a:l051a040

Die grundlegende Form einer Abfrage ist die Suche nach Tokens mit einem bestimmten Etikett auf einer Ebene. Die notwendige Syntax ist in EQ 1 und ein Datenbankabfragebeispiele in EQKC 3 gegeben.

EQ 1 (ETIKETTA). `ETIKETTA = EBENE,('=' | '!='),ETIKETTIERUNG;`

EQKC 3 (ETIKETTA). `Word = freies` `Word != freies`

Abfragen nach Etiketten werden also formuliert, indem der Ebenenname (`EBENE`) mit einem Etikett über verschiedene Operatoren verknüpft wird. Zur Auswahl stehen hierbei ausschließlich das Gleichheitszeichen oder der Ungleichoperator, der in der EQL mit `!=` ausgedrückt wird. Eine Etikettabfrage mit Ungleichoperator wird Disjunktion genannt. Die Semantik einer Disjunktion ist intuitiv: *Suche auf der Ebene nach Tokens, die nicht das angegebene Etikett haben.* Durch diese Abfrage werden alle Tokens der Ebene außer mit dem angegebenen Etikett in der Abfrage zurückgegeben. Gibt es kein Token mit dem angegeben Etikett werden folglich alle Tokens der Ebene in der Segmentliste erscheinen[20].

Dieses logische Verhalten kann auch in Einschränkungen ausgenutzt werden, die nicht das Etikett der gesuchten Tokens betreffen, sondern das Etikett der assoziierten Segmente. In EQKC 1 werden ohne Etiketteinschränkung alle Wörter gesucht, daher mit der einfachen Abfrage `Word != g4d6j7`, wobei `g4d6j7` eine zufällige Zeichenkette ist, die zweifelsfrei für kein Token auf der `Word`-Ebene verwendet wurde. Welche Wörter genau gesucht werden, wird über andere Beschränkungen eingegrenzt.

Etiketten sind Zeichenketten und stehen in den meisten Abfragesprachen in Anführungszeichen, wie auch in SQL-Anweisungen (wie z. B. `'Hanson'` in SQL-Anweisung 2). In der EQL ist dieses nur für Etiketten notwendig, die Zahlen oder Sonderzeichen enthalten. Abfragen nach dem Diphthong [iːɐ], der im Kiel Corpus mit `i:6` codiert ist, wie im Wortsegment `wir` in Abbildung 3.11 oder die Akzentstufe `0`, die auf der Ebene `Accent` angegeben ist, müssten über die Ausdrücke in EQKC 4 erfolgen. EQ 1 muss also modifiziert werden zu EQ 2.

EQ 2 (ETIKETTA).
```
ETIKETT = ETIKETTIERUNG | ("'",ETIKETTIERUNG,"'");
ETIKETTA = EBENE,('=' | '!='),ETIKETT;
```

EQKC 4 (EA). `Phonetic = 'i:6'` `Accent = '0'`

[20] Die EQL stellt keinen expliziten Abfrageausdruck für die Abfrage aller Tokens auf einer Ebene zur Verfügung, da die Abfragemöglichkeit bereits durch Disjunktionsabfragen vorhanden ist.

Die Abfragen mit EQL können im Etikett auf die legale Zeichenmenge (*legal labels*) der Ebene verweisen, indem die Klasse der Zeichenmenge das Etikett ersetzt. Die Kiel Corpus EMU-Datenbank enthält eine solche Klasse `vowel` für alle Vokale auf der `Kanonic`-Ebene, um die Silbenkerne für die automatische Silbifizierung (vgl. Kapitel 7) bereitzustellen. Eine Abfrage: `Kanonic = vowel` gibt alle mit Vokaletikett versehenen Tokens in einer Segmentliste zurück. EQL unterscheidet in der Implementierung im EMU-System nicht zwischen Etikett `vowel` und einer vowel-Klasse, auch nicht unter Verwendung der Anführungszeichen.

In einer Etikettabfrage kann auch nach mehreren Etiketten gleichzeitig gesucht werden, indem die rechte Seite des Operators mit einer Liste von Etiketten gefüllt wird. Hierbei werden die Etiketten über das `|` (ODER) Zeichen verbunden (vgl. EQ 3 und EQKC 5).

EQ 3 (ETIKETTALT).

```
ETIKETT = ETIKETTIERUNG|("'",ETIKETTIERUNG,"'");
ETIKETTALTERNATIVEN = ETIKETT , {'|',ETIKETT};
ETIKETTA = EBENE,('='|'!='),ETIKETTALTERNATIVEN;
```

EQKC 5 (ETIKETTALT). `Word = nehmen | wir | Montag`

Semantisch heißt es dann: *Suche auf der Ebene alle Tokens mit einem Etikett, das einem Etikett in der Liste entspricht.* Im effektiven Vergleich wird die folgende ODER-Logik verfolgt.

(`Segmentetikett = Etikett`) | (`Segmentetikett = Etikett`) `usw.`

Sobald einer der Vergleiche als wahr ausgewertet wird, ist die gesamte Aussage wahr und somit das Token ein der Abfrage entsprechendes. Für den Ungleichoperator funktioniert diese Verknüpfung syntaktisch auch, ist aber nicht sinnvoll[21]. Die in Analogie zu erwartende Semantik wäre: *Suche alle Tokens mit einem Etikett, das keines der Etiketten in der Liste hat.* Unter Berücksichtigung der klassischen Logik wird jedoch deutlich, warum eine Abfrage nicht die erwarteten Tokens zurückgibt. Der logische Ausdruck ist:

(`Segmentetikett != Etikett`) | (`Segmentetikett != Etikett`) `usw`.

[21] Dieser Umstand wird hier beschrieben, da hierfür bereits Fehlermeldungen von Anwendern gemeldet wurden und auch das Lösen des vermeintlichen Problems im Rahmen dieser Dissertation zur Aufgabe stand. Bei genauer Analyse ist es jedoch kein Fehler in der Implementierung und kann bzw. sollte auch nicht behoben werden.

Dieser Ausdruck wird immer als wahr ausgewertet, ob eines der Etiketten nun dem Segmentetikett entspricht oder nicht. Eine alternative Formulierung einer Abfrage, die die gewünschte Semantik hat, ist unter der Verwendung von verbundenen Abfragen (Konjunktionen[22]) möglich.

Die Etikettabfragen können als einfache Abfragen angesehen werden. Gleiches gilt auch für die Positions- und Anzahlabfragen. EQKC 1 enthält mit dem Ausdruck `Start()` eine Positionsabfrage und eine Anzahlabfrage im Ausdruck `Num()`. Positionsabfragen sind nicht auf die Funktion `Start()` beschränkt, sondern enthalten auch `End()` sowie `Medial()` und können über Parameter modifiziert werden.

EQ 4 (POS).
```
POSFKT = 'Start'|'Medial'|'End';
POSA = POSFKT,'(',EBENE,',',EBENE,')','=','0'| '1';
```

Die Position wird nach EQ 4 als Funktion mit zwei Ebenen als Parameter angegeben, hierbei müssen die beiden Ebenen in hierarchischer bzw. autosegmenteller Beziehung zueinander stehen. Die in der Dominanzbeziehung stehende dominierende Ebene (E_D) steht dabei als erster Parameter der Funktion und die dominierte Ebene (E_d) als zweiter Parameter. Die EQ-B(edingung) 1, die ausdrücklich nicht zur formalen Beschreibung der EQL gehört, soll diese Dominanzbeziehung und die Angabe der Parameter der Funktion in richtiger Reihenfolge veranschaulichen.

EQ-B 1 (POS). `?POSA = POSFKT,'(',`E_D`,','`E_d`,')','=','0'| '1'?;`

Diese Funktion wird mit einem booleschen Wert (1=wahr, 0=falsch) über ein Gleichheitszeichen verbunden. Der Ausdruck wird entsprechend ausgewertet. Befindet sich ein Token an dieser Position, ist der Funktionsausdruck wahr (1). Ist der boolesche Wert ebenfalls 1, so ist der Ausdruck 1=1 wahr und das Token ist ein der Abfrage entsprechender Kandidat. Alle möglichen Positionsabfragen sind am Beispiel des Kiel Corpus und dem Etikettierungstrukturausschnitt `Phrase -> Word` also mit `Phrase` als E_D und `Word` als E_d in EQKC 6 aufgelistet. Um die Semantik der einzelnen Positionsausdrücke genau nachvollziehen zu können, sind in EQKC 6 die Ergebnisse der jeweiligen Abfrage in Segmentlisten jeweils unter jedem Ausdruck angegeben. Die Abfragen beziehen sich auf die Postion von Wörtern in Phrasen in Abbildung 3.11.

Die aufgelistete Positionsfunktion `Medial(`E_D`,`E_d`)=1` ist redundant und ließe sich über die Konjunktion[23] aus `Start` und `End` ausdrücken. So sind die beiden Abfragen `Medial(`E_D`,`E_d`)=1` und `Start(`E_D`,`E_d`)=0 & End(`E_D`,`E_d`)=0` äquivalent. Diese Red-

[22] siehe unten
[23] siehe unten

`Start(Phrase,Word)=1`
„Tokens auf Word, die als erstes Token mit einer Phrase assoziiert sind“

labels	start	end	utts
nehmen	0.1	234.163	g103a010

`Start(Phrase,Word)=0`
„Tokens auf Word, die nicht als erstes Token mit einer Phrase assoziiert sind“

labels	start	end	utts
wir	234.163	375.725	g103a010
Montag	375.725	701.912	g103a010
den	701.912	903.225	g103a010
siebzehnten	903.225	1592.787	g103a010

`End(Phrase,Word)=1`
„Tokens auf Word, die als letztes Token mit einer Phrase assoziiert sind“

labels	start	end	utts
siebzehnten	903.225	1592.787	g103a010

EQKC 6 (POS).

`End(Phrase,Word)=0`
„Tokens auf Word, die nicht als letztes Token mit einer Phrase assoziiert sind“

labels	start	end	utts
nehmen	0.100	234.163	g103a010
wir	234.163	375.725	g103a010
Montag	375.725	701.912	g103a010
den	701.912	903.225	g103a010

`Medial(Phrase,Word)=1`
„Tokens auf Word, die nicht als erstes und nicht als letztes Token mit einer Phrase assoziiert sind“

labels	start	end	utts
wir	234.163	375.725	g103a010
Montag	375.725	701.912	g103a010
den	701.912	903.225	g103a010

`Medial(Phrase,Word)=0`
„Tokens auf Word, die als erstes oder als letztes Token mit einer Phrase assoziiert sind“

labels	start	end	utts
nehmen	0.100	234.163	g103a010
siebzehnten	903.225	1592.787	g103a010

undanz ist eine Folge der Analogie zu `Medial(E`$_D$`,E`$_d$`)=1`. Dieser Ausdruck lässt sich nicht für alle der Semantik entsprechenden Tokens durch eine Konjunktion von `Start` und `End` darstellen, denn die Bedingung, dass ein Token gleichzeitig das erste und letzte Kindsegment eines dominierenden Tokens ist, ist nur erfüllt, wenn es das einzige Kind des Elternsegmentes ist. Die Abfrage `Start(E`$_D$`,E`$_d$`)=1 & End(E`$_D$`,E`$_d$`)=1` ist also eine gesondert zu erwähnende Abfrage für ‚Einzelkinder'.

Als nächstes werden die Anzahlabfragen betrachtet, mit der in EQKC 1 die Anzahl der Silben, die das gesuchte Wort dominieren soll, festgelegt wurde. Die Syntax ist ähnlich den Positionsabfragen in EQ 4. Im Anzahlabfrageausdruck wird die Funktion `Num` mit den beiden Ebenen aus der Dominanzbeziehung in der Reihenfolge dominierende Ebene, dominierte Ebene über einen Vergleichsoperator (VOP) mit einer natürlichen Zahl ($INTPN$) verbunden. Die möglichen Anzahlausdrücke sind in EQ 5 formuliert. Beispielabfragen sind zusammen mit den Segmentlisten in EQKC 7 gegeben, wobei E$_D$ diesmal `Word` und E$_d$ `Syllable` ist. Die Ergebnisse in den Segmentlisten stammen aus den Abfragen auf das Kiel Corpus innerhalb der Phrase, die im Ausschnitt des Kiel Corpus in Abbildung 3.11 dargestellt ist.

In den Beispielen in EQKC 7 werden im ersten Ausdruck zweisilbige, im zweiten Ausdruck nicht zweisilbige Wörter gesucht. Im dritten Ausdruck werden explizit Wörter mit mehr als zwei Silben und im vierten Wörter mit weniger als zwei Silben abgefragt. Die beiden letzten Abfragen enthalten der Semantik zur Folge alle `Word`-Tokens aus der zweiten Abfrage und keines aus der ersten.

EQ 5 (NUM).

```
VOP = '=' | '!=' | '>' | '<' | '<=' | '>=';
DIGIT = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';
INTP = DIGIT-'0',{DIGIT};
INTPN = '0' | INTP;
NUMA = 'Num','(',EBENE,',',EBENE,')',VOP, INTPN;
```

Die Semantik richtet sich nach dem Operator und der natürlichen Zahl. Generell gilt aber, ein Token auf der dominierenden Ebene (E$_D$ = `Word`) ist oder ist nicht (je nach Vergleichsoperator) mit einer gewissen Anzahl $INTPN$ an Tokens auf der dominierten Ebene (E$_d$ = `Syllable`) assoziiert. Das heißt, dass im Gegensatz zu den Positionsabfragen (EQ 4) die Semantik für Anzahlabfragen von der dominierenden Ebene (E$_D$) ausgeht. Dieser Unterschied geht aus dem folgenden Ausschnitt der EQKC 1 hervor:

EQKC 7 (NUM).

```
Num(Word,Syllable)=2
```

labels	start	end	utts
nehmen	0.100	195.663	g103a010
Montag	375.725	701.912	g103a010

```
Num(Word,Syllable)!=2
```

labels	start	end	utts
wir	234.163	375.725	g103a010
den	701.912	903.225	g103a010
siebzehnten	903.225	1592.787	g103a010

```
Num(Word,Syllable)>2
```

labels	start	end	utts
siebzehnten	903.225	1592.787	g103a010

```
Num(Word,Syllable)<2
```

labels	start	end	utts
wir	234.163	375.725	g103a010
den	701.912	903.225	g103a010

```
Word  != g4d6j7 & Start(Phrase,  Word  )=1 & Num(  Word  ,Syllable)=2
  E                                 E_d                 E_D
```

Während in der Positionsabfrage die `Word`-Ebene E_d ist, und Tokens auf dieser Ebene in einer gewissen Position gegenüber E_D (`Phrase`) stehen müssen, ist die `Word`-Ebene in der Anzahlabfrage E_D, und die assoziierten Tokens auf E_d (`Syllable`) müssen in gewisser Anzahl vorhanden sein. Durch beide Ausdrücke werden Wörter gesucht. Eine Anzahlabfrage mit diesem Ausdruck auf das gesamte Kiel Corpus ausgeführt, gibt 1700 `Word`-Tokens in 1042 Äußerungen[24] zurück. Auf die Angabe der Segmentliste wird daher verzichtet.

Die für die Semantik entscheidende Ebene der gesamten Abfrage im Beispiel oben, nämlich die Ebene, welche die Tokens für die Segmentliste enthält, steht in einer einfachen Abfrage am Anfang des gesamten Ausdrucks. In den der einfachen Abfrage folgenden beiden Einschränkungen gibt jeweils die `Word`-Ebene die Semantik des Ausdrucks vor. Aus diesem Grund funktioniert die Konjunktion der Ausdrücke, denn in der Konjunktion dürfen die Ebenen in den unterschiedlichen Abfrageausdrücken nicht in hierarchischer oder gar keiner Beziehung stehen.

Bevor die Konjunktion im Detail betrachtet wird, soll die formale Beschreibung einfacher Abfragen (EA) erfolgen. Wie bereits erwähnt, können Etikettabfragen,

[24] Quellcode C.1 im Anhang enthält die R Befehle zur Analyse der Segmentanzahl.

Positionsabfragen und Anzahlabfragen aus formaler Sicht unter einfache Abfragen zusammengefasst werden. Um diese Tatsache leichter in EBNF formulieren zu können, sollen Postions- und Anzahlabfragen als Funktionsabfragen zusammengefasst werden. Unter Berücksichtigung aller formulierter Abfragen können einfache Abfragen wie in EQ 6 definiert werden.

EQ 6 (EA).

```
ETIKETT = ETIKETTIERUNG | ("'",ETIKETTIERUNG,"'");
ETIKETTALTERNATIVEN = ETIKETT , {'|',ETIKETT};
ETIKETTA = EBENE,('=' | '!='),ETIKETTALTERNATIVEN;
POSFKT = 'Start'|'Medial'|'End';
POSA = POSFKT,'(',EBENE,',',EBENE,')','=','0'| '1';
DIGIT = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';
INTP = DIGIT-'0',{DIGIT};
INTPN = '0' | INTP;
VOP = '=' | '!=' | '>' | '<' | '<=' | '>=';
NUMA = 'Num','(',EBENE,',',EBENE,')',VOP,INTPN;
FUNKA = POSA | NUMA;
EA = ETIKETTA | FUNKA;
```

EQ 7 (KONJ).

```
KONJA = EA,{'&',EA};
```

Wie aus EQ 7 hervorgeht, werden für die Konjunktion einfache Abfragen über den `&`-Operator verbunden. Dieser Operator bedingt eine UND-Logik im effektiven Vergleich. Das heißt, nur wenn alle mit UND verknüpften Abfragen als wahr ausgewertet werden können, ist die Abfrage wahr und das Token entspricht der Abfrage. In der ersten Beispielabfrage der Kiel Corpus Datenbank in EQKC 1 sind alle Eigenschaften des Wortes, am Anfang einer Phrase zu stehen und zwei Silben zu dominieren, über Konjunktionen ausgedrückt:
`Word != g4d6j7 & Start(Phrase,Word)=1 & Num(Word,Syllable)=2`
Semantisch formuliert heißt es dann: *Suche alle Wörter (nicht verwendetes Etikett) UND diese Wörter stehen am Anfang einer Phrase UND das Wort dominiert zwei Silben.* Nur wenn alle Bedingungen erfüllt sind, wird das Token als gefundenes zurückgegeben.

In EQKC 1 wurde über eine solche Konjunktion außerdem die Bedingung formuliert, dass das in der Zitierform enthaltene /ə/ getilgt ist. Die beiden Ebenen `Kanonic` und `SinfoKan` stehen in linearer Beziehung zueinander, dadurch kann der `&`-Operator die beiden Etikettabfragen `Kanonik = @` und `SinfoKan = E` syntaktisch und semantisch

richtig verbinden. An dieser Stelle wird der enge Zusammenhang zwischen Ebene und Label-Link sehr deutlich. Genau wie die Eigenschaften des `Word`-Segments, phraseninitial zu stehen und zwei Silben zu dominieren, ist für den Schwa-Vokal die Markierung der Elision eine segmentinhärente Eigenschaft.

Die Semantik einer Konjunktion ist abhängig vom verwendeten Vergleichsoperator innerhalb der einfachen Abfragen, sofern Etikettabfragen enthalten sind. Eine Semantik: *Suche alle Tokens mit einem Etikett, das keines der Etiketten in der Liste hat.*, wie sie oben für die Etikettalternativen im Zusammenhang mit dem Ungleichoperator erwartet wurde, ist über die folgende Abfrage[25] ausdrückbar. Hierbei ist die Ebene in allen drei einfachen Abfragen dieselbe.

`EBENE != ETIKETT`$_1$ `& EBENE != ETIKETT`$_2$ `& EBENE != ETIKETT`$_3$

Für Abfragen, die Ebenen enthalten, die nicht in linearer Beziehung stehen, aber dennoch assoziiert sind, sieht die EQL eine gesonderte Syntax vor. In Betracht kommen hier alle Ebenen in Dominanzbeziehungen. In diesem Fall sind zwei einfache Abfragen mit dem `^`-Operator verknüpft und der gesamte Ausdruck geklammert wie in EQ 8.

EQ 8 (DOM). `DOMA = '[',(KONJA | DOMA),'^',(KONJA | DOMA),']';`

EQKC 8 (DOM).
```
[ Word != g4d6j7 ^ Kanonic = @ ]
[[ Word != x ^ Kanonic = @ ] ^ Phonetic = @ ]
[[ Word != x ^ Kanonic = @ & SinfoKan = E ]
```

Die Semantik eines solchen Ausdrucks ist: *Finde bestimmte Tokens einer Ebene, die bestimmte Tokens auf einer anderen bestimmten Ebene dominieren.* Die Dominanzabfrage unterscheidet nicht zwischen hierarchischen und autosegmentellen Beziehungen. Diese Unterscheidung ist nur für die Ableitung der Start- und Endzeiten relevant, jedoch in keiner Form für die Abfrage. Eine Dominanzabfrage wurde auch in EQKC 1 verwendet, um das gesuchte Wort auf Wörter einzuschränken, die mit einem /ə/ auf der `Kanonic`-Ebene assoziiert sind (vgl. EQKC 8 erstes Beispiel). Das Wort und auch das /ə/ wurden durch weitere Eigenschaften eingeschränkt, die hier jedoch keine Rolle spielen. Diese Abfrage ergibt auf das Kiel Corpus angewandt 12360 Tokens. Im zweiten Abfrageausdruck von EQKC 8 wurde eine weitere Bedingung in Form einer Dominanzbeziehung eingefügt, es werden nicht nur alle *Wörter, die in der Kanonic ein Schwa enthalten* gesucht, wie im ersten Beispiel, sondern zu-

[25] EBENE und ETIKETT sind entsprechend einzusetzen

sätzlich soll es eingeschränkt sein auf die Wörter, *in denen das Schwa auch realisiert wurde*. Diese Abfrage gibt 5081 Wörter zurück. Beide Abfrageergebnisse zusammen genommen, lassen darauf schließen, dass weniger als die Hälfte aller phonologisch angesetzten Schwas getilgt sind. Im dritten Beispiel von EQKC 8 ist die Etikettabfrage erweitert um die lineare Abfrage, die wiederum die Wörter auf jene einschränkt, *in denen Schwa elidiert ist*. Diese Abfrage ergänzt die anderen beiden und führt folglich zu 7809 zurückzugebenen Tokens.

Alle vorgestellten Abfragen bezogen sich auf das Etikett des Tokens und die Beziehungen verschiedener Tokens auf unterschiedlichen Ebenen. Die einzige Abfrage, die innerhalb einer Ebene vorstellbar ist, ist die Suche nach bestimmten Segmentreihenfolgen. In EQKC 2 wurde die Suche durch den Ausdruck in EQKC 9 auf getilgte /ə/ Tokens eingeschränkt, denen kein alveolarer Nasal /n/ folgt. Formal gesehen ist eine solche Sequenzabfrage die sequenzielle Verbindung zweier einfacher Abfragen über den Sequenzoperator `->` , wobei diese Verbindung geklammert wird. Die Reihenfolge mehrerer Tokens müsste somit wie in EQ 9 formuliert werden, wobei die syntaktisch vorgegebene Reihenfolge der paarweisen Klammerung keine Unterschiede in der Semantik zur Folge hat. Die Ebenennamen, die in den einfachen Abfragen (EA) über den Sequenzoperator miteinander verbunden werden, müssen identisch sein oder die entsprechenden Ebenen in linearer Beziehung stehen. Die Suche nach einer komplexeren Sequenz, wie für die folgende zu suchende Phrase: *Montag, den siebzehnten* ist in EQKC 10 zusammen mit der Ergebnisssegmentliste gegeben.

EQ 9 (SEQ). `SEQA = '[',(KONJA | SEQA),'->',(KONJA | SEQA),']';`

EQKC 9 (SEQ). `[ Kanonic = @ -> Kanonic != n ]`

EQKC 10.

`[[ Word=Montag -> Word=den ] -> Word=siebzehnten ]`

labels	start	end	utts
Montag->den->siebzehnten	4070.12	5287.18	g103a010
Montag->den->siebzehnten	3903.00	4944.93	g123a013
Montag->den->siebzehnten	4275.00	5333.18	g123a020
Montag->den->siebzehnten	5778.25	7030.06	g143a003
Montag->den->siebzehnten	1922.75	2976.81	g143a004
Montag->den->siebzehnten	17773.68	19002.56	g216a005

Die bisher vorgestellten Ausdrücke können als isolierte Abfragen formuliert werden. Das Ergebnis der Abfrage sind jeweils die Tokens auf der Ebene, die in der ersten einfachen Abfrage steht, wobei bei Positions-, und Anzahlabfragen die Antwortebene

jene ist, von der die Semantik ausgeht (für NUM - E_D; für POSFKT - E_d). So könnte [`Word != g4d6j7 ^ Kanonic = @`] zu [`Kanonic = @ ^ Word != g4d6j7`]

umgestellt werden, um nicht die `Word`-Tokens in der Rückgabe der Abfrage zu erhalten, sondern die `Kanonic`-Tokens. Das Gleiche funktioniert für die Verbindungen mit dem `&`-Operator, nicht jedoch in den Sequenzabfragen. Hier gilt die Reihenfolge der einfachen Abfragen als Suchkriterium, das nicht geändert werden kann. Die Semantik verbietet somit eine Umstellung.

EQKC 11 (#). `[ Kanonic = @ -> #Kanonic != n ]`

Wenn die Reihenfolge, die Ebene für die Antwortsegmente an den Anfang zu stellen, aufgrund semantischer oder syntaktischer Beschränkungen nicht einhaltbar ist, wie für die Sequenzabfragen, stellt die EQL den `#`-Operator zur Verfügung, der die Ebene markiert, aus der die zurückzugebenden Tokens stammen sollen. Die Raute in EQKC 11 bedingt die Rückgabe der Tokens an zweiter Stelle der gefundenen Reihenfolge an Segmenten. Im Beispiel ist es somit das Token, das /ə/ folgt und kein alveolarer Nasal ist. Da der verwendete Operator grundlegend in Etikettabfragen vor der Ebene auftreten darf, müssen diese Abfragen formal dementsprechend angepasst werden, wie in EQ 10.

EQ 10 (#). `ETIKETTA = ['#'],EBENE,('=' | '!='),ETIKETTALTERNATIVEN;`

Die EQKC 1 und 2 Ausdrücke (S. 96) zeigten bereits, wie die einzelnen Typen von Abfragen zu komplexen Abfragen kombiniert werden können. Mithilfe des EQ Alphabets kann mit einer Abfrage die gesamte Baumstruktur der Etikettierung unter Verwendung der Sequenz- und Dominanzabfragen durchlaufen werden. Durch die Verwendung von Konjunktion können weitere Eigenschaften des Tokens an der jeweiligen Stelle im Etikettierungsbaum hinzugefügt werden. Die Verkettung mehrerer Dominanz- und Sequenzabfragen wurde ebenfalls bereits gezeigt. Diese Verkettung geht jedoch weiter, sodass Sequenzabfragen auch innerhalb von Dominanzabfragen und umgekehrt stehen können. Die formale Darstellung dieser Tatsachen ist in EQ 11 und 12 zu finden[26] .

EQ 11 (DOM). `DOMA='[',(KONJA|DOMA|SEQA),'^',(KONJA|DOMA|SEQA),']';`

[26] Eine Zusammenfassung beider Definitionslisten zu `KONDOMSEQA = '[',(KONJA | SEQA | DOMA),('^','->') ,(KONJA | SEQA | DOMA),']';)` wäre hier denkbar, für die formale Beschreibung jedoch aufgrund der paarweisen Klammerung schwierig in der vollständigen formalen Definition der EQL unterzubringen.

EQ 12 (SEQ). `SEQA='[',(KONJA|SEQA|DOMA),'->',(KONJA|SEQA|DOMA),']';`

Aus allen genannten Gründen ist die EMU Query Language mit EQ 13 vollständig definiert. Dieser EBNF-Ausdruck verdeutlicht, dass die EQL:

- Tokens bzw. ihr Etikett,
- deren linear assoziierte Eigenschaften,
- deren hierarchische Beziehungen,
- aus der hierarchischen Beziehung ableitbare Eigenschaften (Position und Anzahl) und
- deren sequentielle Anordnung

als einfache sowie als kombinierte Suchkriterien berücksichtigt.

EQ 13 (EQL). `EQL = KONJA | SEQA | DOMA;`

3.3.2. Vollständige Formale Beschreibung der EMU Query Language

Die in dieser Dissertation als notwendige Vorarbeit für die Entwicklung des graphischen Interfaces zur Abfragesprache erarbeitete formale Beschreibung der EMU Query Language ist im Folgenden in der erweiterten Backus-Naur Form *top-down*[27] dargestellt.

[27] Die Beschreibung beginnt anders als in anderen Beschreibungen in dieser Arbeit mit einem Ausdruck, der ausschließlich nicht terminale Symbole enthält.

```
EQL
   = KONJA | SEQA | DOMA;
DOMA
   ='[',(KONJA|DOMA|SEQA),'^',(KONJA|DOMA|SEQA),']';
   (*Ebenen müssen hierarchisch oder autosegmentell assoziert sein*)
SEQA
   ='[',(KONJA|SEQA|DOMA),'->',(KONJA|SEQA|DOMA),']';
   (*Ebenen müssen linear assoziert sein*)
KONJA
   = EA,{'&',EA};
   (*Ebenen müssen linear assoziert sein*)
EA
  = ETIKETTA | FUNKA;
ETIKETTA
  = ['#'],EBENE,('='| '!='),ETIKETTALTERNATIVEN;
FUNKA
   = POSA | NUMA;
POSA
   = POSFKT,'(',EBENE,',',EBENE,')','=','0'| '1';
   (*Ebenen müssen hierarchisch oder autosegmentell assoziiert sein*)
   (*zweite Ebene gibt Semantik vor*)
NUMA
   = 'Num','(',EBENE,',',EBENE,')',VOP,INTPN;
   (*Ebenen müssen hierarchisch oder autosegmentell assoziiert sein*)
   (*erste Ebene gibt Semantik vor*)
ETIKETTALTERNATIVEN = ETIKETT , {'|',ETIKETT};
ETIKETT = ETIKETTIERUNG | ("'",ETIKETTIERUNG,"'");
(*EBENE sind Ebenen aus der Etikettierungsstruktur der Datenbank*)
(*ETIKETTIERUNG ist eine freiwählbare Zeichenkette bzw.
  eine legal labels Klasse aus dem Etikettierungsschema.*)
POSFKT = 'Start'|'Medial'|'End';
VOP = '=' | '!=' | '>' | '<' | '<=' | '>=';
INTPN = '0' | INTP;
INTP = DIGIT-'0',{DIGIT};
DIGIT = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';
```

3.3.3. Diskussion und Fazit

Die EMU Query Language ist eine Abfragesprache mit einem überschaubaren Inventar an Zeichen (Alphabet) und einer einfach handhabbaren und formell darstellbaren Syntax. Die in EQL möglichen Ausdrücke zu formalisieren, reicht jedoch nicht aus, um semantisch korrekte Ausdrücke zu erstellen, da eine Abfrage nur im Zusammenhang mit einer gegebenen Etikettierungsstruktur einer Datenbank semantisch korrekt bzw. inkorrekt sein kann. So wären z. B. Dominanzabfragen syntaktisch richtig formulierbar. Aber wenn die im Abfrageausdruck stehenden Ebenen in der Etikettierungsstruktur nicht miteinander in einer hierarchischen oder autosegmentellen Beziehung stehen, ist der Ausdruck semantisch falsch. Derartige semantische Bedingungen können jedoch aus den Kommentaren innerhalb der formalen Beschreibung gelesen werden.

Ein Abfrageausdruck kann in natürlichsprachlichen Sätzen ausgedrückt werden, wie aus der jeweils angegebenen Semantik der einzelnen vorgestellten Ausdrücke hervorgeht. Die EQL bietet für jede Struktur, die das EMU-Etikettierungsmodell ermöglicht, eine Abfragemöglichkeit und erweitert die Abfragemöglichkeiten um Positions- und Anzahlabfragen, die beliebig kombiniert werden können. Auf diese Weise können Einschränkungen für jedes Token innerhalb einer zusammenhängenden Baumstruktur der Etikettierung gesetzt werden. Obwohl die Möglichkeiten der Abfrage vielfältig und dem Etikettierungsmodell angemessen sind, zeigt die EQL auch Schwächen, wie u. a. Cassidy und Bird (2000) erwähnen, die im Folgenden dargestellt werden sollen. Es werden aber auch Alternativlösungen vorgeschlagen, die im Rahmen der Dissertation für die Beantwortung von Benutzerproblemen und für die eigene Forschung erarbeitet wurden.

EQL-Abfragen, die sich an der Baumstruktur entlang hangeln, sind immer auf Tokens angewiesen, die über einfache Abfragen mit einem oder keinem bestimmten Etikett versehen sind. Das Etikett muss vollständig bestimmt sein. Die EQL sieht keine Verwendung von regulären Ausdrücken (The Open Group, 2003) vor, die als Filter, wie z. B. *wildcards* (Platzhalter) oder Schablonen, formuliert werden. Hierfür könnte die EQL in Zukunft erweitert werden, was ein anzustrebendes Ziel sein sollte. Eine Überführung der Etikettierung in eine relationale Datenbank, wie sie in Kapitel 8 beschrieben und diskutiert wird, kann die Verwendung von regulären Ausdrücken ebenfalls gewährleisten. Hierfür eignet sich die für relationale Datenbanken standardisierte ‘Structured Query Language’ (SQL), die reguläre Ausdrücke zulässt und auswertet, wie aus SQ 3 auf S. 45 in Kapitel 2.2 hervorgeht.

Für Benutzer anderer Software ist das Fehlen von regulären Ausdrücken ein sehr hoch eingestuftes Problem. Das EMU Speech Database System ist jedoch für den Aufbau und die Analyse von Sprachdatenbanken mit einem individuell gestaltbaren Etikettierungsschema, entworfen worden. In den meisten Fällen, in denen EMU in der Forschung verwendet wird, werden Sprachdatenbanken aufgebaut, die für eine konkrete Forschungsfrage gestaltet sind. Somit ist das Etikettierungsschema unter dem Aspekt der benötigten Abfragemöglichkeiten zu entwerfen. Hierbei können Strukturen gewählt werden, welche die Verwendung von regulären Ausdrücken unnötig machen.

Das Kiel Corpus ist kein individuell aufgebautes Korpus, dennoch ist z. B. für die Abfrage von Wörtern, die eine <nt> Graphemreihenfolge enthalten, nicht zwingend ein regulärer Ausdruck notwendig. Beachtet man die im Deutschen vorherrschende Phonem-Graphem Korrespondenz (Prinz und Wiese, 1990), so sollte ein solches Wort auf Phonemebene eine Folge /nt/ enthalten. Die eng phonematische Zitierform ist im Kiel Corpus auf der Ebene `Kanonic` abgebildet, so dass die Abfrage <nt> enthaltener Wörter über eine Verbindung einer Dominanzabfrage mit einer Sequenzabfrage erfolgen kann. Es müssten alle Wörter mit beliebigem Etikett gesucht werden, die eine Sequenz: „Token mit dem Etikett `n` gefolgt von einem Token mit dem Etikett `t`“ dominiert. Eine analoge Abfrage wurde zu Beginn des Kapitels formuliert und ist somit umsetzbar.

Ein Gegenbeispiel für das Funktionieren dieser Methode ist die Suche von Wörtern mit Doppelkonsonanten, zu denen es keine unterschiedliche Phonementsprechung im Vergleich zu Einzelkonsonanten gibt. Das Vorkommen ist aber phonologisch vorhersagbar, ein Umstand der in der Etikettierung berücksichtigt und modelliert werden könnte. Für eine andere Methode ist zu bedenken, dass die Analyse der Daten in der Regel in einer Analysesoftware, wie zum Beispiel R (R Development Core Team, 2011) erfolgt, die weiterführende Funktionen bietet, welche auch für die Abfrage der Datenbank im Sinne einer post-hoc Verarbeitung genutzt werden können. Es gibt somit auch die Möglichkeit, mithilfe der Funktionen der Analysesoftware die Abfrageergebnisse weiter einzuschränken. So könnte für die Suche von Wörtern mit der Graphemfolge <nn> eine Abfrage von allen Wörtern mit assoziierten /n/-Phonem mit der EQL an die Datenbank gerichtet und die resultierende Segmentliste der Wörter anschließend über das Ergebnis eines regulären Ausdrucks auf die Wörter, die orthographisch den Doppelkonsonanten enthalten, reduziert werden. Die Etikettspalte kann dafür in Hinblick auf Wörter mit einem zweifachen zusammenhängenden

Vorkommen von <n>[28] analysiert werden. Die Segmentliste wird anschließend auf die Reihen beschränkt, die Etiketten enthalten, für die der reguläre Ausdruck ein Vorkommen an einer Stelle[29] gefunden hat. Eine beispielhafte Umsetzung in R ist im Anhang C.2 zu finden.

Die fehlende Möglichkeit, reguläre Ausdrücke zu verwenden, ist ein großes Problem in den Positionsabfragen. Auch ohne reguläre Ausdrücke können zwar für verschiedene Positionen Platzhalter formuliert werden, indem in den enthaltenen einzelnen einfachen Abfragen alle Etiketten zugelassen werden. Das funktioniert jedoch nur, wenn die Anzahl an Positionen, für die das Etikett des Tokens beliebig ist, zwischen zwei für die Abfrage relevanten Etiketten bestimmt bzw. bekannt ist. Eine unbestimmte Anzahl von zwischenliegenden Tokens ist nicht formulierbar. Eine wenig elegante, aber einsetzbare Lösung kann wiederum die Analysesoftware bieten. Eine Vielzahl von Positionsabfragen, die sukzessive jeweils einen Platzhalter hinzufügen, können die Datenbank abfragen. Die resultierenden Segmentlisten können im Anschluss zusammengefügt werden. Ein Beispiel für die Umsetzung in R ist im Anhang C.3 angefügt. Dieses Vorgehen ist nur möglich, wenn die Anzahl der unbekannten Etiketten in der Sequenz durch den Forscher aus theoretischer Sicht in sinnvoller Weise eingeschränkt werden kann.

Insgesamt bietet die EMU Query Language zusammen mit den Funktionen der EMU/R Bibliothek und R selbst ein ausreichend umfangreiches und handhabbares Werkzeug für die Abfrage von EMU-Sprachdatenbanken. Um es noch handhabbarer zu machen, wurde im Rahmen dieser Dissertation eine grafische Oberfläche entworfen, wofür die hier verwendete formale Beschreibung zwingend notwendig war. Details über die Entwicklung des queryGUIs sind in Kapitel 4.2 erläutert.

Weiterentwicklungen sowohl der EQL selbst als auch des queryGUIs sind niemals auszuschließen und gerade in Hinblick auf reguläre Ausdrücke notwendig. Für neue und weitere Anforderungen an die EMU Query Language muss von Entwicklerseite auf die Erfahrungsberichte und Anforderungen der Benutzer zurückgegriffen werden können. Unter anderem hierfür stehen den Benutzern des EMU-Systems der *Feature Request Tracker* (EMU Developers, 2000) zur Verfügung.

[28] Regulärer Ausdruck: n{2,2}; die übrigen Zeichen im Wort sind unspezifiziert
[29] Vorkommen > 0

4. Benutzerfreundlichkeit

Der Anspruch des Benutzers an eine Software ist der einfache Zugriff auf die gesamten oder zumindest die grundlegenden Funktionalitäten der Software, sowie deren einfache Verwendung. Sowohl die Darstellung als auch der Zugriff auf Funktionen und die einfache Verwendung dieser kann durch grafische Benutzerschnittstellen (GUI[1]) realisiert werden.

Nach Galitz (2007, S. 54) gibt es in der Oberflächenprogrammierung das Prinzip des "Positive First Impression". Gefördert wird ein guter erster Eindruck u. a. durch eine übersichtliche Oberfläche, die eine schnelle Orientierung und einen klaren Überblick über vorhandene Funktionalität bietet. Mit dem grafischen Interface des EMU Speech Database Systems (siehe Abbildung 4.1) soll dieser Anspruch erfüllt sein.

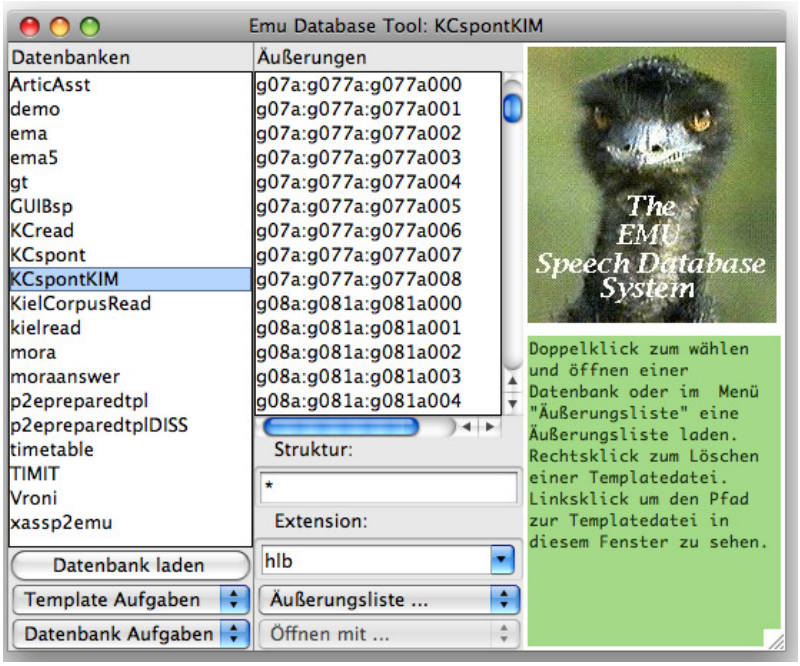


Abbildung 4.1.: Das grafische Interface des EMU-Systems.

[1]Graphical User Interface

Die primäre Benutzerschnittstelle der EMU-Software (EMU Database Tool) zeigt auf den ersten Blick, dass Datenbanken verarbeitet werden und dass diese Datenbanken Äußerungen enthalten (sollten). Das Programmmenü selbst zeigt, dass weitere Operationen zur Verfügung stehen. Unterhalb der Datenbanken und Äußerungen zeigen Menütasten direkte Optionen zur Auswahl an. Nicht zuletzt steht dem Benutzer das Emu[2] zur Seite, das genauere Informationen über die dargestellten Einheiten gibt.

Die in Abbildung 4.1 dargestellte Oberfläche ist eine Tcl/Tk Applikation. Sie wurde vom gesamten Entwicklerteam konzeptualisiert und schließlich zunächst von Lasse Bombien (Bombien, Cassidy, Harrington, John und Palethorpe, 2006) implementiert. Da sie die primäre Schnittstelle des EMU-Systems auf externer Ebene darstellt, ist sie jedoch gleichzeitig auch die Applikation, in der alle anderen EMU-Applikationen durch den Programmierer eingebunden werden müssen. Insofern wurde im Rahmen dieser Dissertation ebenfalls an der Funktionalität dieser Applikation gearbeitet.

Im Database Tool sind mit den oben genannten Eigenschaften einige Konzepte umgesetzt, die die Benutzerfreundlichkeit des Systems gleich bei der Erstverwendung ohne Vorkenntnisse erhöhen. Dieses Prinzip: ohne Vorkenntnisse bestehende Funktionalitäten verwenden zu können, war vor der Weiterentwicklung des Systems, wie die eigene Erfahrung zeigte, besonders für die Abfrage und die Definition des Datenbankschemas nahezu ausgeschlossen. Diese Tatsache allein machte das System gegenüber anderen Programmen wie Praat benutzerunfreundlich. Diese Tatsache sollte mit der Bereitstellung grafischer Oberflächen und Dokumentationen behoben werden. In der Konzipierung der grafischen Oberflächen wurde sich an den Grundprinzipien und vorgeschlagenen Methoden für die Umsetzung, die in Galitz (2007) aus verschiedenen Quellen zusammengefasst sind, orientiert[3].

In diesem Kapitel wird die Umsetzung des Zieles vorhandene Funktionalität des EMU-Systems benutzerfreundlich zu machen, erläutert.

[2] der Vogel

[3] Die Grundprinzipien der Oberflächenprogrammierung wurden nur als Hilfestellung verwendet. Sie sind an sich nicht Teil dieser Dissertation, daher wird auf eine Diskussion der Grundprinzipien verzichtet und für eine detaillierte Übersicht auf die Literatur verwiesen.

4.1. Dokumentation des Systems

Neben dem Anspruch von Benutzern an ein sprachverarbeitendes Programm, Daten darstellen, etikettieren sowie abfragen zu können, ist ein wesentlicher Anspruch des Benutzers eine Dokumentation zum Programm.

EMU als *open source* Programm verfügt nicht über das Personal und die Zeit, um ausführliche Dokumentationen zu verfassen. Somit wurden im EMU-System andere Strategien der Dokumentation entwickelt. Diese Strategien wurden mit dem gesamten Entwicklerteam überlegt und nach und nach umgesetzt.

Im Rahmen dieser Dissertation wurden im Zuge der Dokumentation für die meisten grafischen Benutzeroberflächen in eigener Arbeit Videotutorials erstellt, auf die in Harrington (2010a) verwiesen wird. Sie sind auf den für das Buch in ebenfalls eigener Arbeit im Auftrag des Wiley-Blackwell Verlages erstellten Internetseiten (John, 2010l) bereitgestellt. Die Seiten basieren auf PHP[4] (The PHP Group, 2011) mit CSS[5] (W3C, 2011), und sind in der Struktur den Wiley-Blackwell Internetseiten (John Wiley & Sons, 2011) nachempfunden worden. Ausschnitte der Internetseite und der Videos sind in Abbildung 4.1 zu sehen.

Für die Aufnahmen der Videotutorials wurde ein drehbuchbasierter Desktopbeschrifter in Tcl/Tk implementiert, nachdem mit Freeware und *open source* Programmen keine zufriedenstellenden Ergebnisse erzeugt werden konnten. Für die Aufnahme selbst wurde das sehr gut entwickelte Desktopaufnahmeprogramm Jing Pro (TechSmith, 2009) verwendet. Die Verwendung verschiedener Applikation des EMU-Systems wurden als Video aufgezeichnet. Die Zeitmarken, an denen Etiketten erscheinen sollten, wurden in einer Segmentationsliste notiert und mit einem Etikett versehen. Ein jedes Etikett konnte aus einem Satz an Auswahlmöglichkeiten verschiedener in der Gestaltung vordefinierter Canvas[6] gewählt werden, zu denen die Attribute Bildschirmkoordinaten, Text und Endzeit im Signal angegeben werden mussten. Die Angabe der Bildschirmkoordinaten war notwendig, um die Etiketten innerhalb des Videos an der richtigen Stelle zu positionieren. Die Positionierung konnte mit einem interaktiven Canvas zuvor durch händisches Bewegen auf dem Bildschirm bestimmt werden. Die Angabe der Endzeit war notwendig, da Etiketten in ihrer Darstellungsdauer überlappen konnten. Diese Etikettierungsdatei wur-

[4] Hypertext Preprocessor
[5] Cascading Style Sheets
[6] Ein Canvas ist ein Leinwandobjekt in der grafischen Oberflächenprogrammierung, vergleichbar mit einem rahmenlosen Fenster zum Einfügen von grafischen Elementen oder auch Text.

de zusammen mit der Signaldatei von einer Programmfunktion eingelesen und das Videosignal zusammen mit der Etikettierung dargestellt. Aus den Endzeiten der Etiketten wurden automatisch die Dauern berechnet. Diese kontinuierliche Darstellung von Videosignal und Etikettierung wurde über Jing erneut abgefilmt.

Die Videos sind somit ein ähnliches Resultat wie die Etikettierung eines Sprachsignals und die Darstellung von beidem. Der Unterschied liegt jedoch in der diskreten und kontinuierlichen Darstellung sowie der indirekten und direkten Etikettierung im Signal selbst. Die Videos wurden dateibasiert segmentiert und etikettiert und stellen nur eine schreibgeschützte Variante der etikettierten Signale dar.

Neben den Videos ist mit der neuen primären Schnittstelle (EMU Database Tool), die in Abbildung 4.1 (S. 113) zu sehen ist, eine neue Dokumentationsmethode in das System eingekehrt. Das dargestellte Emu und das darunter zu sehende Textfeld ist eine dynamische Hilfe, die in kleineren Applikationen wie dem autoDBinstaller oder dem Database Tool selbst dem Benutzer stets zur Seite steht und bei der Mausbewegung auf eine Eingabefläche oder einen Aktionsknopf die Aktion erklärt. In Abbildung 4.1 ist diese Hilfe in englischer Sprache, aber wie mit einem Blick auf die Darstellung der autoDBinstaller Oberfläche in Abbildung 5.10 (S. 166) zu sehen sein wird, ist diese Hilfe nicht nur dynamisch, sondern auch zweisprachig.

Das Hilfesystem ist eine eigene Bibliothek und kann von jedem Entwickler in neu entwickelte Applikationen leicht eingebunden werden. So kann direkt bei der Implementierung die Hilfe berücksichtigt werden. Die Dokumentation obliegt somit jedem einzelnen Entwickler selbst. Es entsteht eine Arbeitsteilung. Jeder EMU-Anwender kann auch Entwickler sein. Er dokumentiert seine eigene Applikation und die hauptverantwortlichen Entwickler binden nur die bereits dokumentierte Applikation ein. Alternativ stellen umfangreichere Applikationen eine browserähnliche Hilfe zur Verfügung (Tkassp, GTemplate Editor), die in der Regel über eine Aktionsfläche oder das Menü aufgerufen werden kann. Auch diese Hilfe ist eine leicht einbindbare Bibliothek und somit gelten die gleichen Vorteile wie auch für die dynamische Hilfe.

Die Zweisprachigkeit auch der Eingabeflächen ist eine Weiterentwicklung des Systems, die im Rahmen dieser Dissertation angeregt und umgesetzt wurde und gilt nunmehr für die meisten Applikationen im EMU-System.

Nicht durch die bereits genannten Methoden dokumentierbar sind Programmierschnittstellen, sog. API's[7]. Diese sind vorrangig für Entwickler notwendige Doku-

[7] Application Programming Interface

mentationen. Sie sind jedoch auch für jeden EMU-Anwender von Bedeutung, um über die zur Verfügung gestellten *Scripting*-Möglichkeiten das System an die individuellen Ansprüche anpassen zu können. Neuere Applikationen sind in dieser Richtung noch nicht dokumentiert. Für den EMU-core sowie für die im Rahmen dieser Dissertation entstandenen Applikationen labConvert und dem GTemplate Editor gibt es online-Dokumentationen auf der EMU-Internetseite (EMU Developers, 2011a). Auch die Modellierung von systemeigenen Konzepten und die formalen Beschreibungen, die in Kapitel 3 für Teile des EMU-Systems im Rahmen der Dissertation entstanden sind, tragen zur Dokumentation des Systems bei, die wiederum sowohl für Entwickler als auch für EMU-Anwender als Werkzeug zur Verfügung stehen.

Neben der Dokumentation der Software für den Benutzer benötigen die Entwickler wiederum die Rückmeldung durch den Anwender über die Leichtigkeit bzw. Komplexität der Benutzung, auftretende Probleme, Verbesserungsvorschläge und auch neue Bedürfnisse an Funktionalität. Über entsprechende Anpassungen der Software müssen die Benutzer wiederum informiert werden können. Ein guter Weg des Informationsaustausches stellen Foren dar, wie in Covi und Ackerman (1995) diskutiert wird. Lakhani und von Hippel (2003) beschreiben die Vorteile von Benutzerforen und betonen, dass der größte Gewinn in der Unterstützung der Benutzer untereinander in Bezug auf Anwendungsfragen, mögliche Softwarefehler und dergleichen liegt. Eine gute Dokumentation oder allgemein Erklärungen der Funktionsweise können nicht zuletzt auch über ein aktives Hilfeforum, in denen Benutzer untereinander Erfahrungen austauschen können, erfolgen.

Ein solches Forum wird von SourceForge (Geeknet, Inc., 2011) bereitgestellt. In den letzten Jahren hat sich der EMU-Internetauftritt innerhalb der SourceForge Plattform zur weiteren Unterstützung der Benutzer gewandelt. Hier können grundsätzliche Information und Neuigkeiten über das System erfahren werden. Es werden über die SourceForge Plattform verschiedene *Tracker* zur Verfügung gestellt, z. B. für Benutzeranfragen (*Feature Request Tracker*) oder zum Melden von Problemen (*Bug Tracker*). Mit derartigen *Trackern* können Anregungen und Wünsche nicht nur gestellt, sondern deren Bearbeitung auch verfolgt werden.

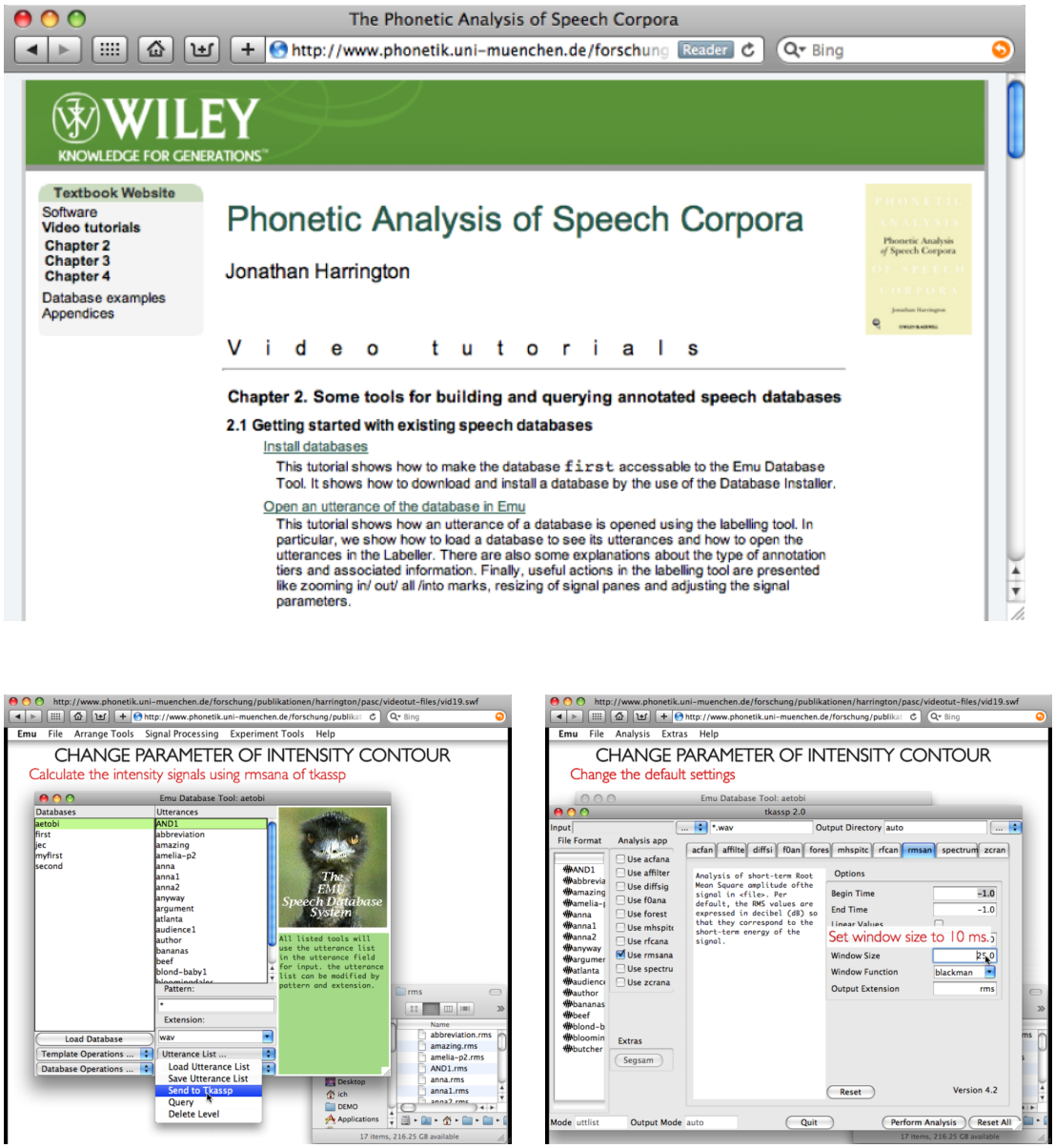


Abbildung 4.2.: Ausschnitt der Internetseite zu Harrington (2010a) für den Menüpunkt Videotutorials (oben) und zwei Ausschnitte eines Videos (John, 2010c) mit unterschiedlichen Etikettierungscanvas zu unterschiedlichen Zeiten für die Ableitung der rms Energy aus dem Sprachsignal unter Verwendung der EMU-Applikation Tkassp.

4.2. Benutzerfreundliche Abfrage ohne EQL

Die EMU Query Language als Abfragesprache im EMU Speech Database System wurde in Kapitel 3.3 vorgestellt. Wie aus der Diskussion des Kapitels hervorgeht, können mithilfe weniger terminaler Symbole und einer überschaubaren Syntax sehr komplexe Abfrageausdrücke formuliert werden. Mit der EQL sind alle Etikettierungsstrukturen, die das Modell vorgibt, abfragbar. Obwohl Phonetiker und Linguisten im Wissenschaftsschwerpunkt mit Sprache umgehen, ist die Verwendung einer formalen Sprache zur Abfrage einer Datenbank keine intuitive Arbeit des Forschers. Um den Benutzern die Abfrage von Datenbanken zu erleichtern, wurde im Rahmen dieser Dissertation eine grafische Benutzeroberfläche (queryGUI) entworfen und implementiert, die bei der Erzeugung des gewünschten Abfrageausdrucks Unterstützung leisten soll. Eine vollständige Darstellung der Benutzung des queryGUI anhand einer bestehenden Datenbank ist im Anhang in Abbildung A.4 zur Übersicht zusammengestellt. Die Entwicklung der grafischen Oberfläche wird in diesem Kapitel vorgestellt.

In der Entwurfsphase mussten die folgenden Punkte grundsätzlich berücksichtigt werden:

1. die möglichen Abfragearten und die Syntax der EQL
2. die möglichen Etikettierungsstrukturen
3. der Zugang für den Benutzer
4. die Dokumentation

Die Aufgabe bestand nun darin, das Etikettierungsschema und die Abfragemöglichkeiten grafisch miteinander zu verbinden, sodass jede Abfrageart für jede Ebene aus dem Etikettierungsschema für den Benutzer grafisch zugänglich ist. Weiterhin mussten die Eingaben in der grafischen Oberfläche so weiterverarbeitet werden, dass ein syntaktisch aber auch semantisch richtiger Abfrageausdruck erzeugt wird. Ebenfalls Teil der Aufgabe war es, sowohl die grafische Oberfläche als auch den erzeugten Abfrageausdruck für den Benutzer auf sinnvolle Art und Weise zugänglich zu machen.

4.2.1. Vorarbeiten

Um Etikettierungsschemata und Abfragemöglichkeiten miteinander sinnvoll grafisch verbinden zu können, mussten alle möglichen Arten von Etikettierungsstrukturen und Abfragen bekannt sein. Hierfür wurde zunächst mithilfe der Beschreibungen

in der Literatur (Cassidy und Harrington, 1996; Cassidy und Bird, 2000; Cassidy und Harrington, 2001; Harrington, 2010a) und durch die eigene Erfahrung sowie durch eruieren des EMU-core Quellcodes (EMU Developers, 2011a) die EMU Query Language formal beschrieben. Die erfolgte und verwendete formale Beschreibung wurde für diese Arbeit in die erweiterte Backus-Naur Form überführt und in Kapitel 3.3.2 dargestellt. Die möglichen Etikettierungsstrukturen und die Notation für die Konzeptualisierung wurden ebenfalls aus der Literatur, aus der eigenen Erfahrung als auch aus dem EMU-core Quellcode abgeleitet. Eine angepasste Notation wurde in Kapitel 3.1.2 vorgestellt.

4.2.2. Einbinden der EQL-Möglichkeiten

Für die Umsetzung des Einbindens der EQL-Möglichkeiten wurden zunächst einfache Abfragen nach EQ 6 auf S. 103 mit der Modifikation in EQ 10 auf S. 106 berücksichtigt, die zur besseren Übersicht in EQ 14 noch einmal formal dargestellt sind. Einfache Abfragen gehen grundsätzlich von einer Ebene aus, die auch gleichzeitig die Ebene ist, von der gefundene Tokens in der Segmentliste gesammelt zurückgegeben werden und damit das Ergebnis der Abfrage präsentieren. Die Umsetzung von EQ 14 in eine grafische Oberfläche ist in Abbildung 4.3 dargestellt.

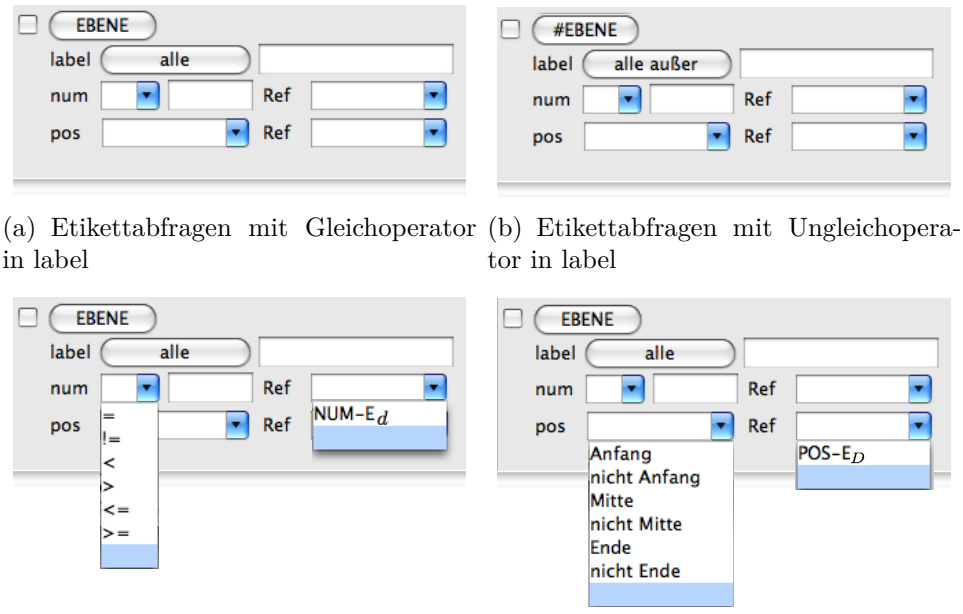
EQ 14 (EA).

```
ETIKETT = ETIKETTIERUNG | ("'",ETIKETTIERUNG,"'");
ETIKETTALTERNATIVEN = ETIKETT , {'|',ETIKETT};
ETIKETTA = ['#'],EBENE,('=' | '!='),ETIKETTALTERNATIVEN;
POSFKT = 'Start'|'Medial'|'End';
POSA = POSFKT,'(',EBENE,',',EBENE,')','=','0'| '1';
DIGIT = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';
INTP = DIGIT-'0',{DIGIT};
INTPN = '0' | INTP;
VOP = '=' | '!=' | '>' | '<' | '<=' | '>=';
NUMA = 'Num','(',EBENE,',',EBENE,')',VOP,INTPN;
FUNKA = POSA | NUMA;
EA = ETIKETTA | FUNKA;
```

Für Abfrageausdrücke nach `ETIKETTA` aus EQ 14 steht für jede Ebene die mit ‚label' annotierte Zeile in Abbildung 4.3 zur Verfügung. Das leere Eingabefeld fungiert als Eingabemöglichkeit für `ETIKETTALTERNATIVEN`-Ausdrücke, wobei die einfache Aneinanderreihung von `ETIKETT`-Ausdrücken mit Freizeichen vorgesehen ist. Der

Gleich- und Ungleichoperator aus den Etikettabfragen ist über den ‚alle'- bzw. ‚alle außer'-*Button*[8] realisiert, während der Ebenenname ebenfalls die Beschriftung eines *Button* darstellt, der das Setzen des Ergebnismodifizierers (#, vgl. Tabelle 3.1, S. 94 und EQ 10, sowie EQKC 11, Seite 106) ermöglicht.

Für die Anzahlabfragen und Positionsabfragen (`NUMA` und `POSA` in EQ 14) werden zwei Ebenen für einen syntaktisch richtigen Ausdruck benötigt. Für die semantische Richtigkeit müssen diese beiden Ebenen in einer Dominanzbeziehung stehen. Wie aus Kapitel 3.3 hervorgeht, gibt die erste Ebene im `NUMA`-Ausdruck die Semantik der Abfrage vor aber im `POSA`-Ausdruck die zweite Ebene. Das Wissen dieser Formalia sollte beim Benutzer nicht vorausgesetzt werden müssen und musste daher in der Umsetzung des GUIs berücksichtigt werden. Die beiden Funktionsabfragen wurden dafür jeweils der semantikvorgebenden Ebene untergeordnet, sodass diese nicht explizit angegeben werden muss und kann.



(a) Etikettabfragen mit Gleichoperator in label

(b) Etikettabfragen mit Ungleichoperator in label

(c) Anzahlabfragen in num

(d) Positionsabfragen in pos

Abbildung 4.3.: Darstellung der grafischen Umsetzung einfacher Abfragen am abstrakten Etikettierungsschema POS-E$_D$(-) → EBENE(-) → NUM-E$_d$(S).

Für die Anzahlabfragen steht die mit ‚num' annotierte Zeile in Abbildung 4.3 zur Verfügung, wobei `VOP` aus EQ 14 in Form einer Auswahlliste, `INT` durch ein frei ausfüllbares Eingabefeld und die dominierte Ebene (`E`$_d$ vgl. S. 101) wiederum in Form einer Auswahlliste umgesetzt ist. Die Zeile ‚pos' steht für Positionsabfragen zur

[8] engl. für Knopf; Schaltfläche

Verfügung, wobei sowohl die Position aus einer Auswahlliste gewählt werden kann als auch die dominierende Ebene (E_D vgl. S. 99). Für die Positionsauswahlliste wurden alle möglichen Ausdrücke für `POSA` in EQKC 6 (S. 100) mit zwei beliebigen Ebenen semantisch interpretiert und alle Kombinationen aus `POSFKT` und dem Ausdruck `('0' | '1')` als Position formuliert. Die Auswahllisten für die Ebenen sind bereits so aufbereitet, dass nur die Ebenen zur Verfügung stehen, die auch zu semantisch richtigen Ausdrücken führen.

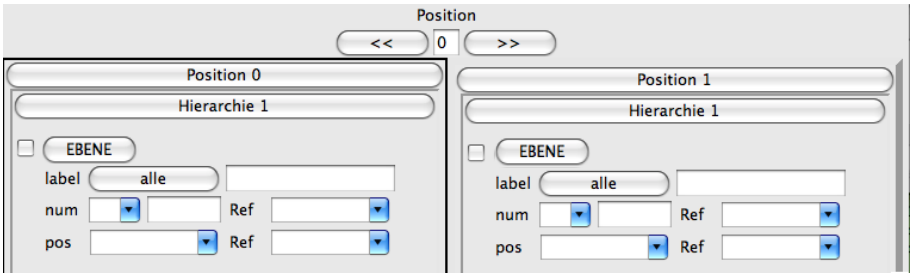


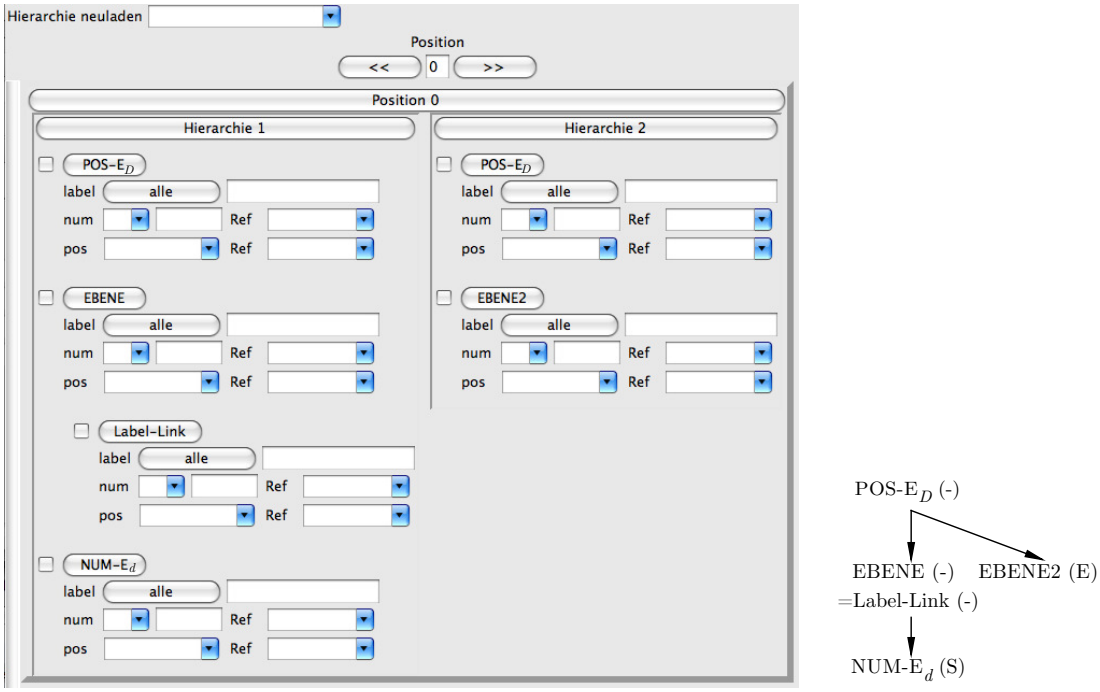
Abbildung 4.4.: Darstellung der grafischen Umsetzung von Sequenzabfragen am abstrakten Etikettierungsschema EBENE(-).

Sequenzabfragen gehen auf eine spezielle Verknüpfung einfacher Abfragen zurück. Diese Verknüpfung ist jedoch nicht implizit, sondern sie muss vom Benutzer definiert werden. Für die grafische Umsetzung bedeutet das, alle Abfragemöglichkeiten müssen an verschiedenen Positionen für eine Sequenz angeboten werden, wobei die Anzahl an Positionen vom Benutzer frei bestimmbar sein muss. Abbildung 4.4 zeigt diese Umsetzung. Über die Positionspfeile können verschiedene Positionen in einer Sequenz angelegt bzw. ausgewählt werden. Für jede Position werden die gleichen Eingabefelder zur Verfügung gestellt.

Für die Konjunktionsabfragen (`KONJA`) und Dominanzabfragen (`DOMA`) werden keine expliziten grafischen Umsetzungen benötigt, denn sie verknüpfen in letzter Instanz die einfachen Abfragen in unterschiedlichen Formen, wobei die Verknüpfungsart von den Beziehungen zwischen den Ebenen der einfachen Abfragen und somit vom Etikettierungsschema bestimmt ist. Entscheidend für die Wahl der Operatoren ist jedoch die Ebene, aus der die Tokens in der resultierenden Segmentliste stammen sollen. Das GUI muss eine entsprechende Markierung anbieten. Hierfür wurde ein so genanntes Kontrollkästchen jedem Ebenen-*Button* vorangestellt.

4.2.3. Einbinden des Etikettierungsschemas

Wie aus Kapitel 3.1.1 hervorgeht, sind im Etikettierungsschema verschiedene Ebenen mit verschiedenen Beziehungen in verschiedenen Ketten definierbar, wobei eine Kette die Hauptkette zur Vererbung der Zeiten aus ggf. vorhandenen zeitgebundenen Ebenen darstellt. Weiterhin ist aus Kapitel 3.3 bekannt, dass Abfragen nur innerhalb einer Kette formuliert werden können. Diese Punkte mussten bei der Umsetzung des graphischen Interface ebenfalls berücksichtigt werden.



(a) (b)

Abbildung 4.5.: Darstellung des Etikettierungsschemas mit mehreren Ketten (Hierarchie 1 und 2) in der grafischen Oberfläche (a) und das grafisch umgesetzte Etikettierungsschema in (b).

In der Notation von Etikettierungsschemata (vgl. Kapitel 3.1.2) werden die Ebenen einer Kette untereinander geschrieben und mit entsprechender Markierung der Beziehung miteinander verbunden. In der Umsetzung des grafischen Interface zur EMU Query Language wurde diese Struktur übernommen, wobei außer für Label-Links auf die explizite Markierung der Beziehung der Ebenen untereinander verzichtet wurde. Die Unterscheidung der Beziehungen one-to-many, many-to-many, hierarchisch oder autosegmentell spielen für die Abfrage selbst keine Rolle. Nicht assoziierte Ebenen befinden sich in unterschiedlichen Ketten. Label-Links sind für eine bessere Übersicht

und Orientierung eingerückt, wie in Abbildung 4.5 demonstriert ist. Jede einzelne Kette im Etikettierungsschema wird im grafischen Interface auf diese Weise, jedoch getrennt von anderen Ketten, dargestellt.

4.2.4. Implementierung

Mit der Bereitstellung des Konzepts der grafischen Oberfläche an sich war die Aufgabe jedoch nicht erfüllt. Das GUI und verschiedene Funktionen mussten implementiert werden. Hierfür wurde sich am modularen Aufbau des Systems orientiert und die Tcl/Tk Schnittstelle[9] genutzt um die separate EMU-Applikation `queryGUI` zu implementieren.

Für die grafische Unterscheidung der einzelnen Ketten wurde im Rahmen der Oberflächenentwicklung ein Algorithmus entworfen (EMU Developers, 2011a). Da die Etikettierungsstruktur nicht in Form eines Schemadiagramms für die Datenbanken vorliegt, musste der Algorithmus die einzelnen Ketten sondieren. Die DML gibt auf Anfrage die Beziehungen der Ebenen zueinander zurück, aus denen die Ketten dann abgeleitet werden können. Außerdem mussten in der Applikation Algorithmen zur Verfügung gestellt werden, die die Benutzereingaben so aufbereiten, dass ein syntaktisch und semantisch korrekter Abfrageausdruck zurückgegeben werden kann. Hierfür wurde ein Algorithmus entworfen, der zunächst alle Eingaben aus dem GUI sammelt, überprüft und gemäß der EQL-Syntax (vgl. Kapitel 3.3.1) zusammensetzt.

Etwas anders als in der eigens erstellten formalen Beschreibung (vgl. Kapitel 3.3.2) geht der Algorithmus von einer Sequenzabfrage aus, wobei diese auch nur mit einer Position besetzt sein kann. Zu jeder Position und jeder Ebene in der Sequenz, werden die Etikett- und die Funktionsabfragen syntaktisch richtig formuliert. Hierfür werden die Etikettalternativen gemäß syntaktischer Vorschrift mit dem `|`-Operator verbunden und die Funktionen umgesetzt in die Form von `NUMA` und `FUNKA`. Alle diese einfachen Abfragen werden durch den `&`-Operator als `KONJA` verknüpft. Die Reihenfolge der Verknüpfung wird über die vorher evaluierte Antwortebene vorgegeben. Die so entstandenen Konjunktionen verschiedener Ebenen werden anschließend gemäß `DOMA` Produktionsregel mit dem `^`-Operator verbunden und geklammert. Gibt es mehrere Positionen in der Sequenz werden diese schließlich nach Produktionsregel `SEQA` miteinander verknüpft. Der Algorithmus ist im EMU-Quellcode (EMU Developers, 2011a) nachzulesen.

[9] vgl. Kapitel 1.3

4.2.5. Zugang für den Benutzer

An dieser Stelle steht dem Benutzer eine grafische Benutzeroberfläche zur Eingabe von Abfragen zur Verfügung, die einen Abfrageausdruck erzeugt. Es wurde nun überlegt, wie der Benutzer auf das grafische Interface hingewiesen werden kann und wie der erzeugte Abfrageausdruck zu seiner eigentlichen Verwendung kommt.

Die eigene Erfahrung im Umgang mit Software-Updates aber auch die Erfahrung mit EMU-Nutzern zeigt, dass sich über Neuerungen nicht unbedingt informiert wird. Aus diesem Grund sollte das queryGUI so eingebunden werden, dass es überaus offensichtlich als vorhanden erkannt wird. In der Umsetzung wurde die Schnittstelle zum queryGUI direkt an der Stelle im System eingefügt, an der die Abfrageausdrücke üblicherweise eingetragen werden. Abbildung 4.6 auf der nächsten Seite zeigt das EMU Query Tool mit der eingebundenen Schnittstelle zum queryGUI in Form eines *Buttons* mit der zur Motivation bestimmten Beschriftung ‘graphical query‘.

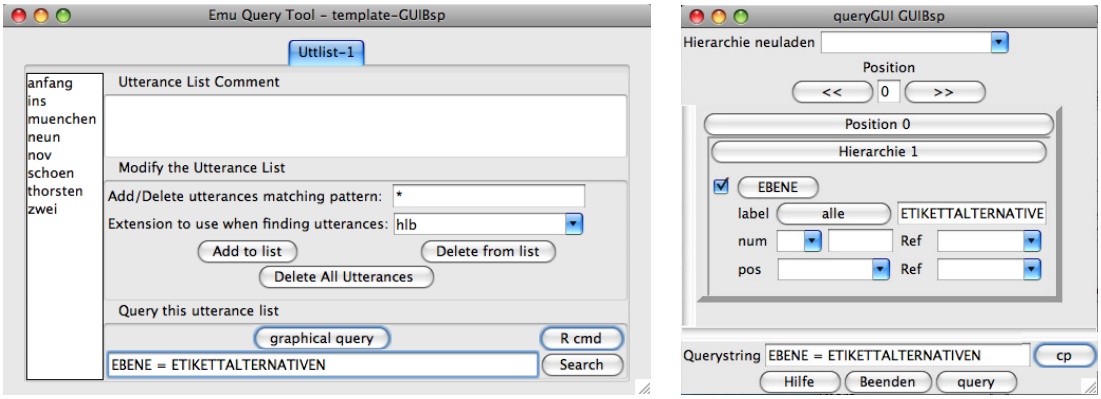
Als explizite Abfrageapplikation mit einem Eingabefeld für Abfrageausdrücke wurde das EMU Query Tool auch als geeigneter Ort für das Einsetzen des Abfrageausdrucks gewählt. In der Umsetzung wird nach dem Erzeugen des Abfrageausdrucks im queryGUI dieser direkt in das EMU Query Tool übertragen und dieses wird auf dem Bildschirm fokussiert, während das queryGUI minimiert wird. Da es aber auch andere Möglichkeiten gibt, den Abfrageausdruck zu verwenden, z. B. in anderen Applikationen oder externen Programmen, wird der erzeugte Abfrageausdruck automatisch in die Zwischenablage kopiert und könnte an beliebiger Stelle eingefügt werden. Für die gleiche Funktionalität wurde jedoch auch ein *Button* im queryGUI (vgl. Abbildung 4.6) bereitgestellt.

Eines der externen Programme, die für die Abfrage vorgesehen sind, ist die R Umgebung, in der mithilfe einer Funktion (emu.query) Abfragen an das EMU-DMS gesendet werden. Um auch diesen Arbeitsschritt benutzerfreundlicher zu gestalten, wurde dem GUI ein ‘R cmd’-*Button* hinzugefügt, der den R-Befehl[10] erzeugt und in die Zwischenablage legt.

4.2.6. Dokumentation

Die fertige Applikation wurde im letzten Schritt dokumentiert. Hierfür wurde eine Kurzanleitung für die Bedienung für den EMU typischen Hilfebrowser erzeugt und

[10] für das Beispiel: emu.query("GUIBsp","*","EBENE = ETIKETTALTERNATIVEN")



(a) (b)

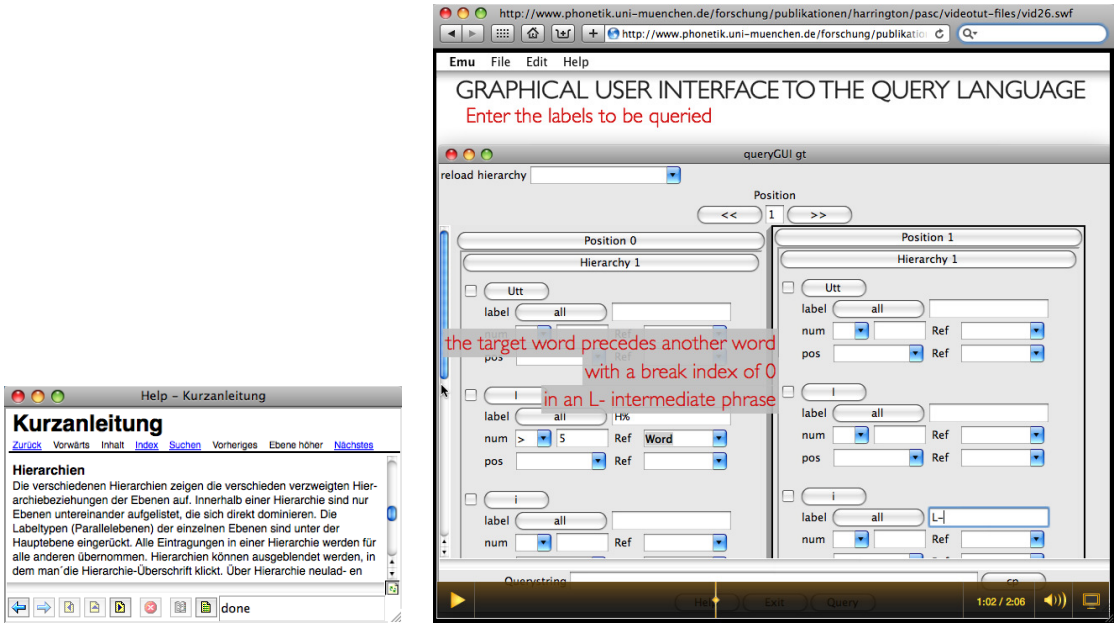
Abbildung 4.6.: Darstellung des möglichen Zugriffs des Benutzers auf das queryGUI vom EMU Query Tool aus, sowie der Zugriff auf die erzeugten Abfrageausdrücke in beiden Applikationen.

eine Dokumentation anhand verschiedener Beispiele geschrieben (John, 2003b). Ein Videotutorial (John, 2010h) wurde ebenfalls explizit für Kapitel 4.6 aus Harrington (2010a) erstellt. Die beiden grafischen Dokumentationswege sind in Abbildung 4.7 beispielhaft dargestellt.

4.2.7. Fazit

Das queryGUI stellt das Etikettierungsschema als Eingabemaske dar, in der die gewünschten Etiketten nur eingetragen werden müssen. Der Abfrageausdruck wird aufgrund der Eingaben von der Applikation selbst erzeugt. Damit ist es dem Nutzer ganz ohne Kenntnisse der EQL möglich, Abfrageausdrücke zu formulieren, wie Williams (2008) bestätigt. Durch die Anzeige des Abfrageausdrucks kann die EQL sehr schnell erlernt werden.

Die regelbasierte automatische Erzeugung des Abfrageausdrucks führt zu einem weiteren Vorteil: der Verhinderung syntaktisch oder semantisch falscher Ausdrücke. Dennoch kann der Benutzer die Semantik selbst bestimmen, wobei menschliche Fehler hierbei auch durch die grafische Eingabemöglichkeit nicht verhindert werden können. Das queryGUI deckt alle Abfragearten ab und erstellt ausschließlich syntaktisch wohlgeformte Abfrageausdrücke. Einschränkungen gibt es nur für die speziellere Semantik sehr komplexer Abfragen. An dieser Stelle muss der Benutzer den vorgegebenen Abfrageausdruck leicht verändern. So würde eine Abfrage: *Suche*



(a) (b)

ABBILDUNG 4.7.: Dokumentation des queryGUIs als Kurzanleitung für die Bedienung (a) und als Videotutorial in (b).

alle zweisilbigen Wörter am Anfang einer prosodischen Phrase, die in der Zitierform einen Schwa–Vokal enthalten, der getilgt ist, der jedoch nicht vor einem alveolaren Nasal vorkommt. durch die damit verbundene Sequenzabfrage für die Folge /əX/, wobei X nicht /n/ sein darf, automatisch einen Abfrageausdruck erzeugen, der diese Sequenz auch auf die damit verbundenen Wörter bezieht (vgl. EQKC 12).

EQKC 12.

```
[[ Word != g4d6j7 & Start(Phrase,Word)=1 & Num(Word,Syllable)=2
^
Kanonic = @ & SinfoKan = E ] -> [ Word !=g4d6j7 ^ Kanonic !=n ] ]
```

Im Abfrageausdruck müsste die Sequenzabfrage der Wörter als Korrektur gelöscht und die Klammerung gemäß syntaktischer Vorschrift angepasst werden, sodass der Abfrageausdruck in EQKC 2 (S. 96) entsteht. Das ist die einzige bekannte, nicht direkt formulierbare Semantik. Das GUI könnte jedoch dahingehend modifiziert werden. Modifikationen werden zur Zeit diskutiert. Grundsätzlich müsste das GUI Eingabefelder für eine Sequenzabfrage zusätzlich in der grafischen Umsetzung der

einfachen Abfrage einbinden. Da Sequenzabfragen wiederum alle anderen Abfragemöglichkeiten enthalten müssen, ist es keine triviale Angelegenheit eine sinnvolle grafische Lösung zu finden, die für den Benutzer als benutzerfreundlich erachtet wird und leichter ist als die Klammerung entsprechend zu ändern.

In der Diskussion steht auch eine Abfragemöglichkeit, in der der Benutzer den Etikettierungsbaum, der aus der Datenbank extrahiert werden soll, grafisch erzeugt – genauso als würde die Etikettierung gerade vorgenommen. Hierfür könnten die vorhandenen Etikettierungsfunktionen genutzt werden. Der so erstellte Etikettierungsbaum würde dann in die EQL übersetzt. Bisher wurde von der Umsetzung abgesehen, da es seit der Bereitstellung des queryGUIs von Benutzerseite keine negative Kritik bezüglich großer Schwierigkeiten für die Benutzung der Abfragemöglichkeit in EMU gibt. Die Notwendigkeit wird daher als gering eingeschätzt und anderen Weiterentwicklungen nachgestellt.

Aus einer anderen Perspektive bietet die modulare Art und Weise der Implementierung Vorteile, wenn Änderungen an der Syntax der EMU-Query Language vorgenommen werden oder diese Sprache durch eine andere Abfragesprache wie zum Beispiel SQL ersetzt wird. Denn es muss nur die Abbildung (*Mapping*) zwischen GUI-Einträgen und der Abfragesprache neu implementiert werden. Auf den Benutzer hat es bis auf die ggf. hinzugekommenen nutzbaren neuen Abfragemöglichkeiten soweit keinen Einfluss.

Insgesamt stellt das im Rahmen dieser Dissertation entwickelte queryGUI ein sowohl für den Benutzer als auch für den Programmierer benutzerfreundliches Tool zur Abfragesprache dar und trägt somit maßgeblich zur besseren Benutzbarkeit einer Funktionalität des EMU-Systems bei, womit es sich gegenüber anderen Tools abhebt. Die Die höhere Benutzerfreundlichkeit des Tools gegenüber der Verwendung der EQL zeigte sich u. a. bereits in der eigenen Erfahrung als auch in der eigenen Lehrtätigkeit mit Studierenden der Phonetik und der empirischen Linguistik.

4.3. Benutzerfreundliches Datenbanktemplate ohne EMU-DDL

Eine benutzerfreundlichere Erstellung eines Datenbankschemas, in der die Verwendung der DDL vermieden werden kann, wurde durch die EMU-Applikation GTemplate Editor realisiert, dessen Entwicklung im Folgenden beschrieben wird. Ein Benutzungsbeispiel des graphischen Template Editor anhand des Datenbankschemas in John (2004) ist im Anhang in Abbildung A.2 zu sehen.

4.3.1. Motivation

In Kapitel 3.2.2 wurde die Sprache, die in der Datenbanktemplate verwendet wird, um die Daten und Datenstrukturen auf externer Ebene zu definieren, vorgestellt. Die Datenbeschreibung obliegt dem Benutzer, der eine Textdatei selbst editiert, wobei die Definitionen sehr strengen syntaktischen Regeln unterliegen. Weiterhin werden eine Vielzahl von Einzeldefinitionen benötigt, die zum Teil voneinander abhängen. Der EMU-Nutzer ist somit bereits vor der ersten Benutzung des Systems nicht nur mit dem völlig anderen Ansatz des Programms im Vergleich zu anderen Sprachverarbeitungsprogrammen konfrontiert, sondern außerdem mit der Datendefinitionssprache, die zu erlernen ist. Diese Punkte tragen nicht zur Benutzerfreundlichkeit des Systems bei. Um dem Benutzer die Datendefinition zu erleichtern, sollte im Rahmen dieser Dissertation eine grafische Oberfläche konzipiert, implementiert und im System integriert werden. Eine geeignete Umsetzung sollte alle möglichen Definitionsmöglichkeiten visuell darbieten und dabei Eingabemöglichkeiten zur Verfügung stellen. Aus den Benutzereingaben sollten sowohl syntaktisch als auch semantisch korrekte Datenbankschemata automatisch erzeugt und gespeichert werden.

4.3.2. Vorarbeiten

Für die Konzeptualisierung wurde mithilfe der Literatur, einer älteren EMU-Dokumentation (Cassidy, 2004), dem EMU-Quellcode sowie der eigenen Erfahrung eruiert, welche Definitionen in einem Datenbankschema auf externer Ebene enthalten sein müssen und können, welche semantischen Zusammenhänge die definierten Teile haben sowie in welcher Form Definitionen formuliert werden müssen. Notwendige Definitionen und deren Zusammenhänge wurden in Form eines Templateschemadia-

grammes als zusammenfassende Orientierung und somit als Werkzeug aufbereitet, wie in Kapitel 3.2.2 vorgestellt und erläutert. Die Datendefinitionssprache selbst ist in Kapitel 3.2.3 zusammen mit der für diese Dissertation erstellten formalen Beschreibung der EMU-DDL aufgeführt. Die formale Beschreibung war für die automatische Erzeugung syntaktisch richtiger Datenbankschemata notwendig.

4.3.3. Grafische Umsetzung

Wie aus Kapitel 3.2.2 hervorgeht, enthält ein Datenbankschema jegliche Definitionen zu Etikettierungsstrukturen, Signalen, Speicherinformationen, Systemkonfigurationen und Applikationseigenschaften. In der Umsetzung wurden diese verschiedenen Teile als getrennt voneinander dargestellt, um Zusammenhänge bzw. Zusammenhanglosigkeiten hervorzuheben. Hierfür wurde sich in der groben Struktur der grafischen Oberfläche für eine Applikation mit verschiedenen Registerkarten entschieden, die nach Priorität geordnet sind. Mit höchster Priorität eingeschätzt wurden zunächst die Definitionen für die in der Datenbank enthaltenen Etikettierungen und Signale. Abbildung 4.8 zeigt diese Umsetzung.

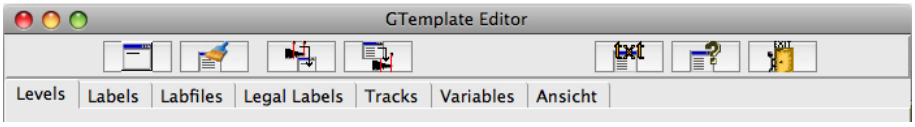


Abbildung 4.8.: Registerkarten in der grafischen Oberfläche der EMU-Applikation GTemplate Editor (GTed).

Dem folgenden Zitat für die Umsetzung des *Familiarity*-Prinzips für die Oberflächenprogrammierung Folge leistend, wurde sich bei der Benennung der einzelnen Registerkarten an den dazugehörigen Deklarationen der DDL orientiert. Diese Entscheidung sollte auch eine gewisse Kompatibilität[11] für frühere EMU-Nutzer, die die EMU-DDL bereits erlernt haben, darstellen.

> "Build on the user´s exisiting knowledge [...]. Build into the interface concepts, terminology, workflows, and spatial arrangements that are already familiar to the user." (Galitz, 2007, S. 51)

[11] ebenfalls ein Grundprinzip in der Oberflächenprogrammierung nach (Galitz, 2007)

Für jeden Entitätstypen mit Ausnahme von **path**, **hlb** und **primary** im Templateschemadiagramm, dass noch einmal in Abbildung 4.9 dargestellt ist, wurde eine Registerkarte (vgl. Abbildung 4.8) angelegt.

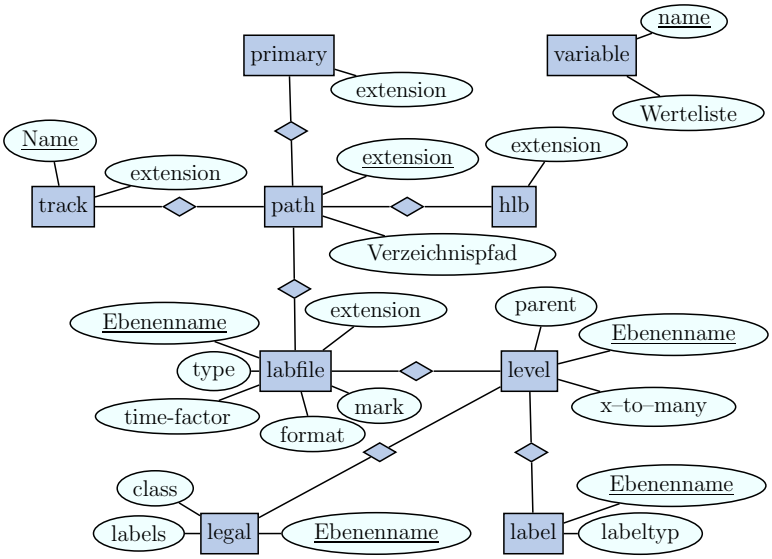
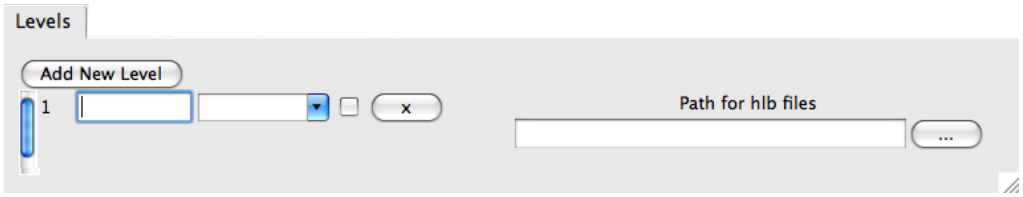


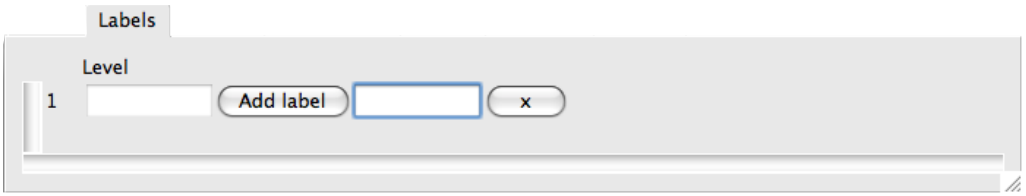
Abbildung 4.9.: Vereinfachtes Templateschemadiagramm im ER-Modell.

Wie bereits aus dem Templateschemadiagramm hervorgeht, bilden **path** und **hlb** einen Beziehungstypen, wobei **path** weitere Beziehungstypen mit **labfile** und **track** bildet. Es wäre möglich gewesen, auch für die Pfadangaben in **path** eine eigene Registerkarte anzulegen, wobei auf dieser dann zwischen Pfaden zu Signalen (**track** Entitäten), der *hlb*-Dateien und der externen Etikettierungsdateien unterschieden werden hätte müssen. Darauf wurde jedoch zugunsten einer anderen Lösung verzichtet. Alle Attribute der Entitätstypen in Abbildung 3.7 sind wichtige durch den Benutzer zu definierende Informationen. Neben den Registerkarten zur Orientierung mussten hierfür somit Eingabemöglichkeiten zur Verfügung gestellt werden.

Für die einfache Ebenendefinition (vgl. **level** im Templateschemadiagramm) wird in der grafischen Umsetzung das Eingabefeld für das Attribut **Ebenenname** zur Verfügung gestellt, wie in Abbildung 4.10(a) zu sehen ist. Die dem Feld folgende Auswahlliste setzt das **parent**-Attribut um, wobei die Auswahlliste nur bereits definierte Ebenen zur Auswahl anbietet, um die semantische Korrektheit des Datenbankschemas zu wahren. Das **x-to-many**-Attribut geht von der Default-Ausprägung one-to-many aus und es wurde zum Definieren der many-to-many Beziehung das Kontrollkästchen vorgesehen. Auf der Levels-Registerkarte können somit alle Ebe-



(a) Ebenen



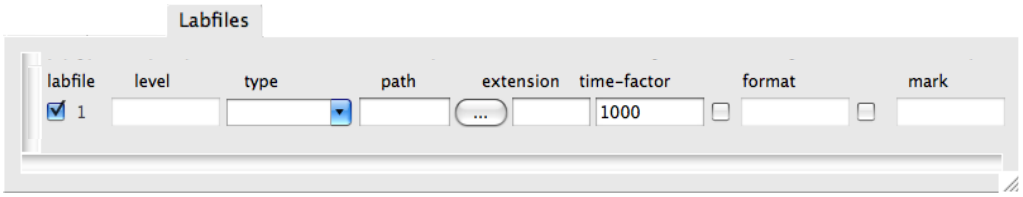
(b) Label-Links

Abbildung 4.10.: Registerkarten für die Etikettierungsstrukturen Ebene (a) und Label-Link (b) mit den benötigten Eingabefeldern.

nen und deren hierarchische Beziehungen definiert werden. Für Label-Links (Abbildung 4.10(b)) wird nur ein Eingabefeld für das Attribut **labeltyp** auf der Labels-Registerkarte zur Verfügung gestellt. Das erste Eingabefeld ist mit dem Ebenennamen-Eingabefeld verknüpft.

Die Zeitgebundenheit einer Ebene und die damit ggf. implizit entstehenden autosegmentellen Beziehungen zwischen Ebenen gehen einher mit externen Etikettierungsdateien (labfiles), die durch den Entitätstypen **labfile** im Templateschemadiagramm mit mehreren Attributen modelliert sind. Für alle Attribute stellt die Labfiles-Registerkarte (vgl. Abbildung 4.11(a)) Eingabefelder bereit, wobei das Attribut **type** wiederum als Auswahlliste umgesetzt wurde und nur die beiden Ausprägungen SEGMENT und EVENT zur Auswahl anbietet. Nach dem selben Prinzip werden auch für die legalen Zeichenmengen pro Ebene Eingabefelder auf der Legal Labels-Registerkarte (in der selben Abbildung (b)) zur Verfügung gestellt.

Wie aus Kapitel 3.2.2.2 hervorgeht, werden Etikettierungen in verschiedenen Dateien und u.U. an verschiedenen Dateispeicherorten abgelegt. Hierzu gehören die *hlb*-Dateien und die externen Etikettierungsdateien. Während die Dateinamen durch das interne Schema (vgl. Abbildung 3.8) durch den Entitätstypen **HIERARCHY** bestimmt sind und für zusammengehörige Datensätze identisch sind, müssen laut Templateschemadiagramm die Attribute **Verzeichnispfad** und **extension** des Entitätstypen **path** mit Ausprägungen versehen werden.



(a) zeitgebundene Ebenen

(b) legale Zeichenketten

Abbildung 4.11.: Registerkarten für die Etikettierungsstrukturen zeitgebundene Ebene (a) und legale Zeichenketten (b) mit den benötigten Eingabefeldern.

Da **path** sowohl mit **labfile** als auch mit **hlb** Beziehungstypen bildet, müssen diese Angaben auch hierfür durch den Benutzer im GUI eintragbar sein. In der Umsetzung wurde auf der Levels-Registerkarte nur ein Eingabefeld für den Dateipfad zu den *hlb*-Dateien angegeben, da die Dateiextension bekannt ist[12]. Die Wahl dieser Registerkarte beruht auf der Überlegung, dass ein Datenbankschema wohlgeformt ist, auch wenn es ausschließlich die Informationen auf der ersten Registerkarte enthält. Diese Tatsache geht aus der in der Vorbereitung entstandenen formalen Beschreibung der EMU-DDL auf S. 87 in Kapitel 3.2.3 hervor[13].

Für die externen Etikettierungsdateien sind sowohl Pfad als auch Extension frei wählbar, sofern nicht bereits Etikettierungsdateien durch andere Umstände vorhanden sind. In der Umsetzung wurden für beide Informationen Eingabefelder zu Verfügung gestellt (vgl. Abbildung 4.11(a)). Um die Eingabe der Verzeichnispfade benutzerfreundlich zu gestalten, wurde eine Aufrufbarkeit zum Dateibrowser mittels eines *Buttons* eingefügt.

Unter allen genannten Aspekten wurde auch die Definierbarkeit von Datenbanksignalen benutzerfreundlich gestaltet, wie aus Abbildung 4.12 hervorgeht. Der bereits vorhandene Eintrag ist eine Ausprägung des **Name**-Attributes und bezeichnet Sprachsignale. Er wird als Defaultangabe im GUI vorgegeben unter der Prämisse,

[12] hlb

[13] TPLLEV
= {LEVEL}- ,LF,HLBPATH,LF,{LABEL},LF,{LEGAL},LF,{TPLLAB},LF,{TPLTRACK};
TPL = TPLLEV | TPLTRACK;

dass EMU-Sprachdatenbanken Sprachsignale enthalten, die etikettiert und analysiert werden sollen.

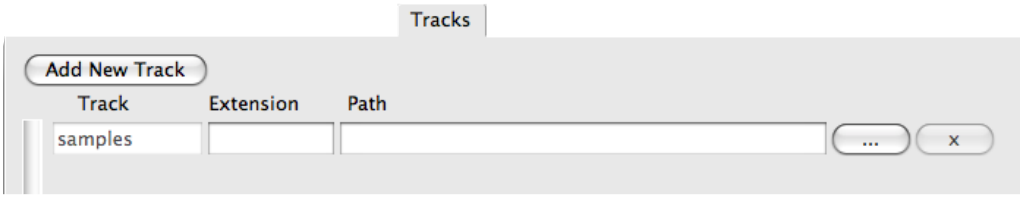
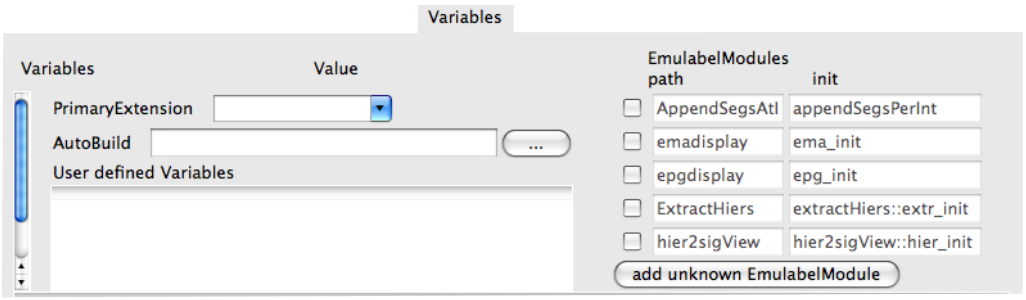


Abbildung 4.12.: Registerkarte für die Signale.



(a) Variables

(b) Applikationsparameter

Abbildung 4.13.: Registerkarten für Variablen (a) und Applikationsparameter (b) mit den benötigten Eingabefeldern.

Abbildung 4.13 zeigt die Umsetzung für die Definition anwendbarer Module und zu möglichen Variablen sowie für die Sichten in der EMU-Applikation `labeller2`, dem Etikettierungstool. Für die benutzerfreundliche Definition, welche Module im Etikettierungstool zur Verfügung gestellt werden sollen, wurde für die Umsetzung entschieden, alle verfügbaren EMU-Labeller Module aufzulisten und über Kontrollkästchen auswählbar zu machen. An dieser Stelle wird das Prinzip verfolgt, dass sich die Applikation nicht ändern muss, wenn sich das interne System verändert. Denn welche Module verfügbar sind, wird bei jedem Start des GTemplate Editors evaluiert, indem zugrundeliegende Dateien im EMU-System gesucht und auf Schlüsselwörter hin überprüft werden. Benutzerdefinierte Module können auf diese Weise berücksichtigt werden, sodass auch diese auswählbar gemacht werden können.

Die primäre Extension[14] als Variable PrimaryExtension ist wie auf S. 77 in Kapitel 3.2.2.1 mit einer Ausprägung von **extension** aus dem Templateschemadiagramm definierbar, was direkten Einfluss auf die interne Struktur der Datenbank hat. In der grafischen Umsetzung der Variablendefinition wird daher eine Auswahlliste bereitgestellt, die nur alle definierten Extensionen zur Auswahl anbietet.

In der AutoBuild Variable wird ein Skript angegeben, das auf die Datenbank auf verschiedene Weisen ausführbar ist. Skripte liegen in Dateiform vor. Für die benutzerfreundliche Deklaration eines Skriptes in der Template reichte es wiederum ein Eingabefeld mit verknüpften Dateibrowser zur Verfügung zu stellen. Über benutzerdefinierte Variablen konnten keine Annahmen gemacht werden, in welcher Form diese Eingaben durch den Benutzer gemacht werden würden, sodass nur ein Textfeld als Eingabemöglichkeit in Betracht kam. Für die Sichtendefinition ist nur die Angabe der zu betrachtenden Ebenen und Signale von Bedeutung, sodass das einfache Auflisten der Möglichkeiten und der Auswahlmöglichkeit durch Kontrollkästchen als ausreichend benutzerfreundlich erachtet wurde.

Für die Gestaltung des grafischen Template-Editors wurden nicht nur die Eingabemöglichkeiten berücksichtigt, sondern auch die Modifikationsmöglichkeiten. Denn sowohl schon bei der Konzipierung des Datenbankschemas als auch bei der grafisch unterstützen Eingabe können Fehler unterlaufen, die nicht vorhersehbar aber änderbar sind. Für Änderungen, wie das Löschen von Definitionen, wurden zu Gunsten der Benutzerfreundlichkeit Kontrollkästchen vor den Eingabefeldern oder Entfernen-*Buttons* nach den Eingabefeldern eingefügt, über die Definitionen schnell und unkompliziert ausgeschaltet werden können.

Für das Einfügen von Ebenen in einer bestehenden Kette und um vorzubeugen, dass Definitionen nicht vom grafischen Interface abgedeckt werden, wurde eine Schnittstelle zum textbasierten Template Editor, der alle vorgenommenen Definitionen in EMU-DDL ausdrückt, konzipiert. Die beiden Editoren sind über die Schnittstelle miteinander verknüpft und können sich auf diese Weise wechselseitig updaten, sofern es vom Benutzer erwünscht ist.

[14] `PRIM` in der formalen Beschreibung auf S. 87

4.3.4. Implementierung

Der grafische Template Editor wurde als Tcl/Tk Applikation `GTed` implementiert (vgl. EMU-Quellcode (EMU Developers, 2011a)) und ins EMU-System über eine Schnittstelle zur `dbemu`-Applikation eingebunden[15].

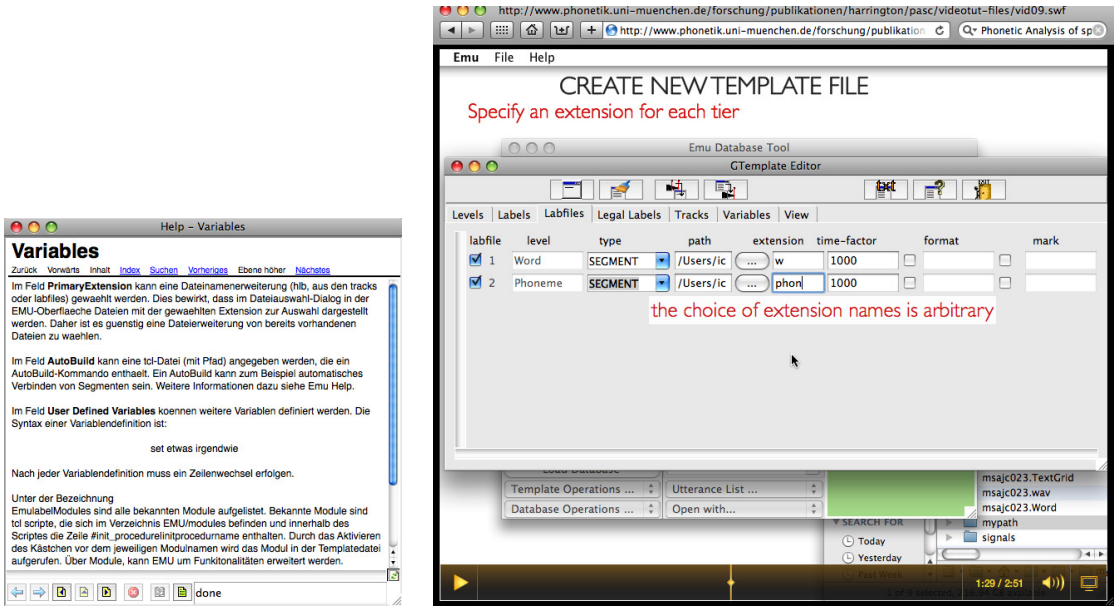
Neben der Programmierung der einzelnen grafischen Komponenten, die für die einzelnen Datenbankschemateile benötigt wurden, mussten Routinen entwickelt werden, die das GUI mit Eingaben füllen, wenn eine Datenbanktemplate zur Überarbeitung eingelesen wird. Hierfür wiederum wurde ein Algorithmus entwickelt, der vorhandene Definitionen über das Datenbankmanagementsystem gezielt anfragt und entsprechend automatisch aufbereitet.

Außerdem wurden Algorithmen benötigt, die die Benutzereingaben auf Vollständigkeit prüfen und Fehler dokumentieren. Um vollständige Benutzereingaben in eine vom EMU-System verarbeitbare Form zu bringen, wurde eine Routine bereitgestellt, die die Eingaben in die EMU-DDL übersetzt. Für die Übersetzung wurde sich an der formalen Beschreibung in Kapitel 3.2.3 auf S. 87 orientiert. Der implementierte Algorithmus bereitet die Benutzereingaben in durchzählbare zusammenhängende Datenstrukturen auf und erstellt zunächst die Ausdrücke der Ebenendefinitionen, der Signaldefinitionen und schließlich die Definitionen der Variablen und Sichten. Die wohlgeformte und semantisch richtige Templatedatei wird am benutzerdefinierten Speicherort anschließend abgelegt und ist bei geeigneter Konfiguration des EMU-Systems im EMU Database Tool verfügbar.

4.3.5. Dokumentation

Als Dokumentationsmittel wurde der EMU typische Hilfebrowser gewählt, der innerhalb der Applikation aufgerufen werden kann. Weiterhin wurden Videodokumentationen (John, 2010g; John, 2010e) für Kapitel 2.5 und 2.7 in Harrington (2010a) aufgenommen, die die Benutzung und die Umsetzung dokumentieren. Abbildung 4.14 stellt beide Dokumentationsarten beispielhaft dar.

[15] Für die unterschiedlichen Benennungen von Programmname und Programmpaket vgl. Tabelle B.1



(a) (b)

Abbildung 4.14.: Dokumentation des GTemplate Editors als Browserhilfe und als Videotutorial.

4.3.6. Fazit

Der GTemplate Editor bietet jedem EMU-Anwender die Möglichkeit, auf einfachen Weg eine Datenbanktemplate zu erzeugen. Die Komplexität wird durch die grafische Umsetzung nicht eingeschränkt, da alle Konzepte grafisch umgesetzt sind. Der versierte Benutzer hat zudem die Freiheit, textbasiert, unter Verwendung der DDL, Deklarationen zu schreiben, ohne die Vorteile der Plausibilitätsprüfung, die die grafische Oberfläche bietet, zu verlieren. Damit ist auch das Flexibilitätsprinzip, dass nach Galitz (2007) (vgl. folgendes Zitat) für die Oberflächenprogrammierung angesetzt wird, erfüllt.

> "Permit people to choose the method of interaction that is most appropriate to their situation. [...] Flexibility is accomplished by providing multiple ways to access application functions and perform tasks. " (Galitz, 2007, S. 51)

Schwierigkeiten in der Verwendung des Template Editors liegen ferner an der grafischen Umsetzung selbst als am Konzept, das EMU zugrundeliegt. Trotz einer benutzerfreundlichen Eingabemöglichkeit, kann auch dieser Editor den Benutzer nicht

darin unterstützen, den datenbankähnlichen Ansatz des Systems gänzlich zu verstehen. Er unterstützt ihn jedoch dabei, Zusammenhänge zu erkennen, da zusammenhängende Teile entsprechend grafisch als Einheit dargestellt sind.

Die größte Unterstützung im Zuge der Benutzerfreundlichkeit des Editors liegt darin, den Anwendern das Erlernen der EMU-DDL zu ersparen und vermeidet die Erstellung fehlerhafter Templatedateien, die allein durch Tippfehler oder Syntaxfehler entstehen können. Somit sind zwei wesentliche Fehlerquellen in der Erzeugung des unbedingt notwendigen Datenbankschemas aufgehoben. Wie auch bereits für die benutzerfreundliche Abfrage wurde die bessere Benutzerfreundlichkeit des GUI's gegenüber der textbasierten Definition in der eigenen Forschung und Lehrtätigkeit mit Studierenden bestätigt.

5. Funktionalität

> "Certain features of EMU (e.g., segmental annotation by hand, creation of data files) are comparable to the features offered by other software, such as Praat. However, other features of EMU represent particular strengths, [...]." (Williams, 2008, S. 170)

> "As well as the two major 'views' [...], EMU contains several other tools and utilities (including a utility that converts label files to and from Praat TextGrid format). [..] this review cannot cover the full breadth of functionality available." (Williams, 2008, S. 168)

> "The semi-automatic annotation at higher levels is a significant strong point for a database consisting of speech [...]." (Williams, 2008, S. 172)

> "Compared to Praat, EMU offers less scope for creating and running user-written scripts. EMU 's user-scripting facilities are confined to the AutoBuild feature, for building up levels of annotation automatically." (Williams, 2008, S. 173)

> "There is no equivalent to the more extensive soundfile manipulation features of Praat, for such operations as splitting a long file into sub-files and saving them, [...]." (Williams, 2008, S. 173)

Wie aus den aufgeführten Zitaten hervorgeht, ist die Funktionalität des EMU-Systems zum einen vergleichbar mit anderen Programmen, ist zum anderen jedoch auch breit gefächert und verschieden von anderen Tools. Schwächen werden im Bezug auf *Scripting*-Möglichkeiten gesehen.

Die vorliegende EMU-Kritik[1] bezieht sich auf eine EMU-Version die bereits einige aber nicht alle Weiterentwicklungen aus dieser Dissertation enthält. Im Folgenden werden die Weiterentwicklungen der Funktionalität näher betrachtet. Ohne ausschweifende Diskussion und Fazit wird sich zeigen, dass die Funktionalität des Sys-

[1] *Review*

tems im Rahmen dieser Dissertation wesentlich erweitert wurde und auch die Kritik der zu wenig vorhandenen *Scripting*-Möglichkeiten und der fehlenden Funktion zum Schneiden von Signalen aufgrund der unternommenen Arbeiten nicht mehr haltbar ist. Die Nützlichkeit jeder neuen Funktion wird jeweils für die Umsetzung diskutiert und zusammengefasst.

5.1. Benutzerdefinierter EMU-Templatepfad

Mit der Zeit hat sich für Betriebssysteme eine Benutzerverwaltung mit verschiedenen Sicherheitsstufen durchgesetzt, obgleich fast jeder forschereigene Heim-PC oder Laptop nur von einer einzigen Person verwendet wird[2]. Die Betriebssysteme schützen sich heutzutage vor ihren Nutzern, was ohne Diskussion von Vorteil sein kann.

Vor der Weiterentwicklung des EMU-Systems war der Speicherort für Templatedateien fest definiert und zwar individuell für jede Plattform. Im Programmverzeichnis sowie im Homeverzeichnis des Benutzers konnten Datenbanktemplates abgelegt werden. Programmverzeichnisse sind heutzutage auf jedem Betriebssystem vor dem Zugriff durch den Benutzer geschützt. Insofern ist dieser Speicherort außerhalb des Mehrbenutzerbetriebs nicht günstig, da er nicht umstandsfrei nutzbar ist.

Die zentrale Speicherung der Datenbanktemplates hat den Vorteil, dass die Datenbankschemata im System integriert scheinen. Die Datenbankausprägungen können jedoch nach Kapitel 3.2.2.2 in anderen Verzeichnissen und auch auf ganz anderen Medien gespeichert sein. Die Verteilung der zur Datenbank gehörenden physisch vorhandenen Dateien hat Nachteile für den Benutzer. Die Datenbankarchivierung ist aufwendig, da Datenbankschemata und Ausprägungen aus unterschiedlichen Verzeichnissen zusammengesucht werden müssen. Datenbankausprägungen, die aus Backup-Gründen bereits auf externen Medien gespeichert sind, werden unbrauchbar, wenn das Datenbanktemplate aufgrund einer Neuinstallation des lokalen Betriebssystems nicht mehr vorhanden ist. Diese Schwierigkeiten lassen sich zwar durch ein Backup des Datenbankschemas überbrücken, jedoch bringt die doppelte Speicherung gleicher Daten immer wieder den Nachteil der Inkonsistenz mit sich. Eine bessere Lösung wäre, die Datenbanktemplates zusammen mit der Datenbankausprägung speichern zu können.

Hierbei sei angemerkt, dass das Datenbanktemplate auch manuell aus den Ausprägungen wiederhergestellt werden könnte. Für diese Rettungsmaßnahme wurden in eigener Arbeit Algorithmen implementiert. Anstatt jedoch dem Benutzer Datenbankrettungsmaßnahmen anzubieten, wurde eine geeignete benutzerfreundliche Lösung für das Speichern von Datenbanktemplatedateien in verschiedenen und vor allem beliebigen Verzeichnissen im Rahmen dieser Dissertation angestrebt. Die entwickelte Lösung und ihre Umsetzung sind im Folgenden beschrieben.

[2]Das lässt sich vermutlich auf die Arbeitszeit des Wissenschaftlers zurückführen.

5.1.1. Methode

Bevor die eigentliche Konzipierung neuer Lösungen beginnen konnte, musste die im System vorgesehene Datenspeicherung mithilfe der Literatur und dem EMU-Quellcode eruiert werden. Die Ergebnisse sind in Kapitel 3.2.2.2 zusammengefasst.

Nach Kapitel 3.2.2.2 sind die beiden Verzeichnispfade zu den Datenbanktemplatedateien im EMU-core (vgl. Kapitel 1.3) festgeschrieben. Das Programm kennt also mindestens ein systemeigenes Verzeichnis, in dem es nach Datenbankschemata sucht. Es sollte jedoch viele benutzerdefinierte Verzeichnisse für Datenbanktemplates kennen. Die Lösung für dieses Problem wurde im Problem selbst gefunden. Das System sollte in den bekannten systemeigenen Verzeichnissen nicht nach Datenbankschemata selbst suchen, sondern nach ihren Speicherorten.

In der Umsetzung wurde folgende Strategie implementiert. Wie die meisten bekannten Programme wurde nun auch das EMU-System mit einer Konfigurationsmöglichkeit ausgestattet. Im Backend wird eine Konfigurationsdatei verwaltet, die im bekannten systemeigenen Benutzerverzeichnis vorliegt. Die editierbare Textdatei kann eine beliebige Anzahl an benutzerdefinierten EMU-Templatepfaden enthalten. Abbildung 5.1 zeigt beispielhaft eine solche Datei. Da die Konfigurationsdatei auch für

```
1
2 #EMU_TEMPLATE_PATH=/Users/Natty/dbs/KCspontKIM:/Users/Natty/dbs/kielread:/Users/Natty/dbs/
      timetable:/Users/Natty/dbs/mora:/Users/Natty/dbs/gt:/Users/Natty/dbs/KCread:/Users/Natty/
      Documents/DISS/J_smith_1:/Users/Natty/Documents/DISS/J_smith_1/test:/Users/Natty/Documents/
      DISS/e2p:/Users/Natty/Documents/Interspeech2011:/Users/Natty/dbs/KielCorpusRead:/Users/Natty
      /dbs/ema:/Users/Natty/Documents/DISS/e2p/KCShort:/Users/Natty/first:/Users/Natty/second
3
4 #PRAAT_SEP_INFOMSG=0
```

Abbildung 5.1.: Beispielhafter Inhalt der EMU-Konfigurationsdatei.

weitere Konfigurationen genutzt werden sollte, wurde für Templatepfade ein Schlüsselwort entworfen, dem dann die Angaben mit einem Pfadtrenner folgen sollen. Die formale Beschreibung in der Backus-Naur Form[3] in FB 2 zeigt die vorgesehene entwickelte Syntax[4].

[3] Eine Übersicht dieser formalen Sprache wurde in Kapitel 2.3 gegeben.

[4] Das Schlüsselwort `#PRAAT_SEP_INFOMSG` setzt eine Variable, die das Ein- und Ausblenden von Hilfedialogen bei der Verwendung der direkten Schnittstelle zu Praat konfiguriert.

FB 2 (TPLPATH).

```
(* PFAD, WERT - semantisch sinnvolle frei
wählbare Zeichenkette *)
(* LF - Zeilenwechsel *)
TPLSW = '#EMU_TEMPLATE_PATH';
SW = '#PRAAT_SEP_INFOMSG';
TPD = TPLSW,'=',[PFAD,{':',PFAD}],LF;
VARS = SW,'=',{WERT},LF;
EMUCONF = [TPD],{VARS};
```

Der `emu-core` wurde modifiziert, sodass beim Programmstart die Konfigurationsdatei eingelesen und das Schlüsselwort (`TPLSW` in FB 2) in der Datei gesucht wird. Alle Pfadangaben werden im `emu-core` den bereits festgeschriebenen Templatepfaden für die Laufzeit des Programms hinzugefügt. So sucht das EMU-DBMS auf Anfrage alle Templatedateien in allen angegebenen Verzeichnissen.

Eine manuell editierbare Textdatei birgt Risiken, sodass auch für das Anlegen dieser Datei eine benutzerfreundliche sowie programmierfreundliche Lösung gefunden und umgesetzt werden musste. Für die benutzerfreundliche Umsetzung der Funktionalität wurde eine EMU-Applikation `emuconf` implementiert und im Dateimenü der primären Schnittstelle des Systems (`dbemu`) eingebunden[5]. Mit der verwendeten Tcl/Tk Programmiersprache wurde das GUI in Abbildung 5.3(b) erstellt, welches im Backend alle benötigten Funktionen enthält. Hierzu gehören die Funktionen zum Einlesen der Konfigurationsdatei, zur grafischen Darstellung der Dateiinhalte, zum Abgleichen der Benutzereingaben mit vorhandenen Inhalten und zum Speichern der neuen Verzeichnispfade. Die Implementierung ist im EMU-Quellcode (EMU Developers, 2011a) verfügbar.

Dokumentiert wurde die Applikation für die Bedienung durch das Emu[6], das dem Anwender zur Seite steht (vgl. Kapitel 4.1) und weiterhin einem Videotutorial (John, 2010f) für Kapitel 2.7 in Harrington (2010a). Abbildung 5.3(a) zeigt einen Ausschnitt des Videotutorials mit einer älteren Version der Applikation. Eine kleine ebenfalls entworfene `emuconf`-API kann den Dateipfad zur Konfigurationsdatei zurückgeben und einen weiteren Pfad hinzufügen. Die API ist in Abbildung 5.2 dokumentiert.

[5] `emuconf` ist im Benutzer-Frontend der EMU Konfigurationseditor und `dbemu` das EMU Database Tool, vgl. Tabelle B.1.

[6] die Abbildung des Tiers

```
1 # loads applications package
2 package require conf
3
4 # returns file path of emuconf file
5 emu::conf::get_emuconffile
6
7 # adds path PATH to emuconf
8 emu::conf::addtplpath PATH
```

Abbildung 5.2.: Dokumentation der `emuconf`-API.

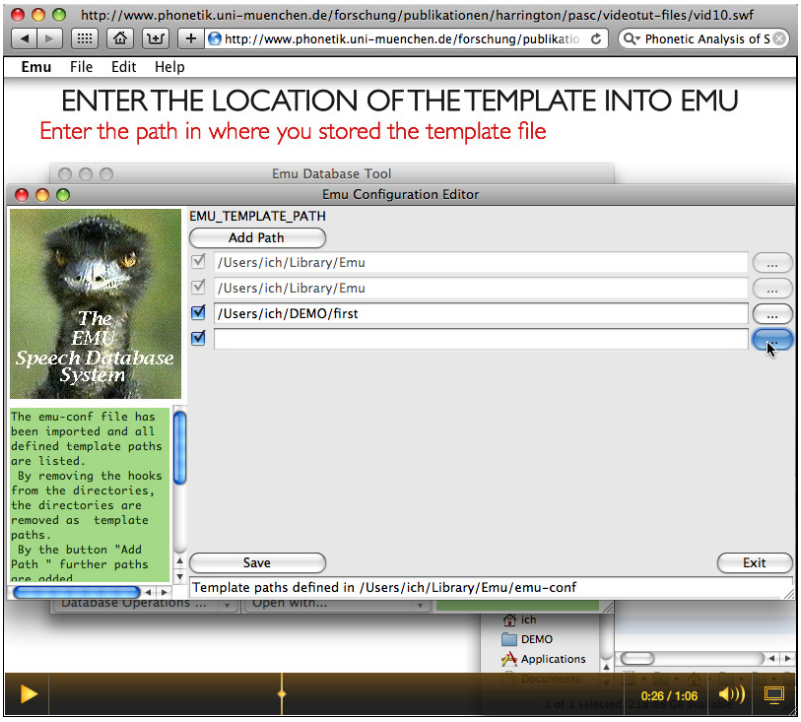
5.1.2. Ergebnis und Fazit

Mit der in Tcl/Tk implementierten EMU-Applikation `emuconf` konnte ein benutzerfreundlicher Konfigurationseditor für die Konfiguration des EMU-Systems bereitgestellt werden. Neben der grundlegenden Neuerung, das EMU-System unabhängig von Datenbankschemata konfigurieren zu können, ist der bisher größte Nutzen der Applikation die Freiheit, unabhängige Templatepfade definieren zu können. Datenbankteile müssen nicht mehr zwangsläufig auf dem System verteilt liegen, sondern können in einem lokalen Verzeichnis als vollständige Datenbank im Sinne von Kemper und Eickler (2006) gespeichert werden.

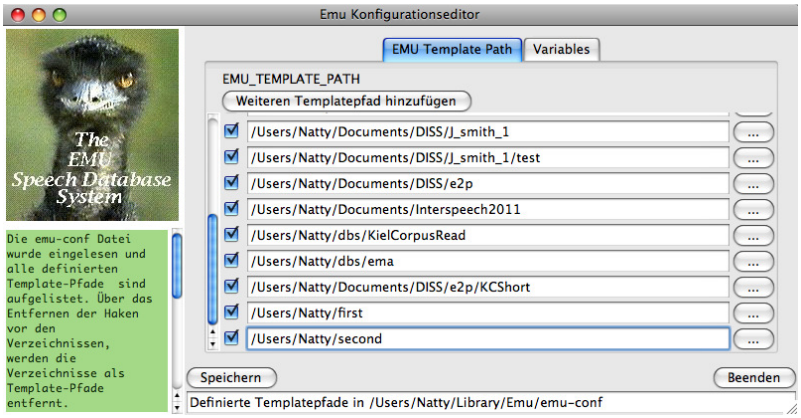
EMU-Sprachdatenbanken können auf diese Weise sehr leicht archiviert und auch transferiert werden. Auch wenn die Applikation eher eine funktionale Erweiterung des EMU-Systems darstellt, trägt sie jedoch auch zur Weiterentwicklung der Benutzerfreundlichkeit bei, nicht nur in Bezug auf die Archivierung und den Transfer. Das Konzept der Datenbank kann bei einer derartigen Organisation der Daten vom Benutzer besser verstanden werden, denn mit dem Blick auf ein Datenbankverzeichnis kann die Ausprägung und das Datenbankschema voneinander unterschieden werden. Der Bezug dieser beiden Teile ist somit besser gegeben.

Der einzige Nachteil könnte bei der Verwendung an sich gesehen werden. Denn, da Datenbankspeicherorte benutzerdefiniert sein können, müssen sie auch vom Benutzer definiert werden, wenn Datenbanken in der oben beschriebenen Form organisiert sein sollen. Die Alternative der zentralen Speicherung an einem bereits vordefinierten Ort wurde jedoch nicht entfernt, so ist der Anwender frei in der Wahl seiner Strategie.

Wie sich in Kapitel 5.4 zeigen wird, eröffnete `emuconf` zusätzlich die Möglichkeit zur Integration einer ganz neuen Funktionalität im EMU-Systems.



(a) Videotutorial



(b) GUI der Applikation im derzeitigen Entwicklungsstand

Abbildung 5.3.: EMU-Applikation Konfigurationseditor mit Angaben verschiedener Speicherorte für Datenbanktemplatedateien in einem Ausschnitt aus dem Videotutorial (a) und nur das GUI der Applikation in erweiterter aktuellen Form (b).

5.2. Schneiden, Darstellen und Etikettieren

5.2.1. Darstellen und Etikettieren

> "The primary interface for creating annotations within the Emu system is the graphical Emu Labeller" (Cassidy und Harrington, 2001, S. 65)

Die von Cassidy und Harrington (2001) im Zitat erwähnte primäre grafische Benutzerschnittstelle wurde in der Weiterentwicklung vom EMU Database Tool (`dbemu`) abgelöst. Dennoch ist der EMU Labeller immer noch die Applikation im System zur Darstellung von Signalen und Etikettierungen sowie zum Etikettieren selbst.

> "The labeller can display any time series data as an x-y graph [...]. The displays can be zoomed and scrolled [...]." (Cassidy und Harrington, 2001, S. 66)

Neben der im Zitat erwähnten Signaldarstellung konnte in 2001 auch ein Sonagramm aus den Zeitsignalen abgeleitet und dargestellt werden. Die Sonagrammdarstellung teilte jedoch nicht die Eigenschaft, zoom- und scrollbar zu sein. Andere vergleichbare Tools waren mit dieser Funktionalität jedoch bereits ausgestattet, die dem EMU-System als Etikettierungstool aus diesem Grund vorgezogen wurden. WaveSurfer (Sjölander und Beskow, 2000) war eines dieser Sprachverarbeitungsprogramme. Die Signaldarstellung in dieser Software basiert auf einer *open source* Tcl/Tk Erweiterung, der `snack` Bibliothek (vgl. Kapitel 1.3).

Auch das EMU-System war mit einer Tcl/Tk Schnittstelle ausgestattet und ist es immer noch. Da eine zoom- und scrollbare Sonagrammdarstellung zeitgemäß ist, sollte auch EMU diese Funktionalität bereitstellen und zwar unter der Verwendung der `snack` Bibliothek. Weiterhin ist nach Kapitel 1.3 eine Anforderung an das System, plattformunabhängig zu sein. Versuche, den Quellcode des Systems derartig anzupassen, dass alle damaligen Teile des Systems (`emusf libemu padgraph ssff tkemu`) auf allen Plattformen kompilierbar sind, scheiterte an der `padgraph`-Bibliothek, die die Signaldarstellung für die EMU Labeller Applikation `labeller` enthielt.

Um also das System sowohl plattformunabhängig zu machen, und um ein zoombares Sonagramm bereitstellen zu können, musste die komplette Signaldarstellung neu implementiert werden. Die Implementierung war eines der Ziele in dieser Dissertation. Die Umsetzung des Zieles wird im Folgenden dargestellt.

5.2.1.1. Methode

Die Arbeit begann mit dem Sichten des Quellcodes des damaligen Labellers, um ihn modifizieren zu können. Es stellte sich sehr schnell heraus, dass die nicht auf allen Plattformen kompilierbare `padgraph` Bibliothek nicht nur die Signaldarstellung selbst sondern auch die Synchronisation der einzelnen Darstellungsfenster übernahm.

Da `padgraph` jedoch nicht nutzbar war, wurde nicht nur die Signaldarstellung neu implementiert, sondern der gesamte EMU Labeller. Es entstand eine von Grund auf neue Implementierung, die im graphischen Design an den Vorgänger angepasst ist. Nur prozedurale Algorithmen, die z. B. Berechnungen durchführen oder verschiedene Initialisierungen tätigen, wurden unverändert aus dem alten Quellcode übernommen.

Weiterhin wurde die Signaldarstellung anders als zuvor im Backend des Labellers eingebunden. Auf eine externe Applikation wurde verzichtet, da eine solche Darstellung außerhalb des Labellers bzw. ohne Einbettung in eine globalere Applikation bisher keine Verwendung fand. Es wurde jedoch eine API entwickelt, wodurch auch außerhalb des Labeller GUI's Signaldarstellungen möglich sind. Sie war notwendig, für die Applikation EMU Segmenter (vgl. Kapitel 5.2.2) sowie für die neu hinzugefügte Funktion im EMU Query Tool, die gefundene Tokens direkt im Labeller öffnet (vgl. Abbildung A.4 *Open in Labeller*) und in den entsprechenden Zeitbereich des Tokens hineinzoomt.

Die Signaldarstellung wurde, wie vorher diskutiert, mithilfe der `snack` Bibliothek realisiert. Für die Darstellung der Signale selbst ist die Bibliothek hervorragend nutzbar. Probleme gab es mit den jeweiligen Signal- und Achsbeschriftungen. Diese wurden daher von Hand nachgerüstet und mit zoombaren Signalbeschriftungen versehen, die sich je nach Fenstergröße anpassen, um auch bei kleiner Darstellung Signale anhand der Namen unterscheidbar zu machen. Weiterhin wurden Möglichkeiten zum Skalieren der Achsen über grafische Bedienelemente bereitgestellt, wobei ohne benutzerdefinierte Skalierung der genaue Maximalwert des Signals immer auf der Achse angegeben ist. Als Funktionalität hinzugefügt wurde außerdem:

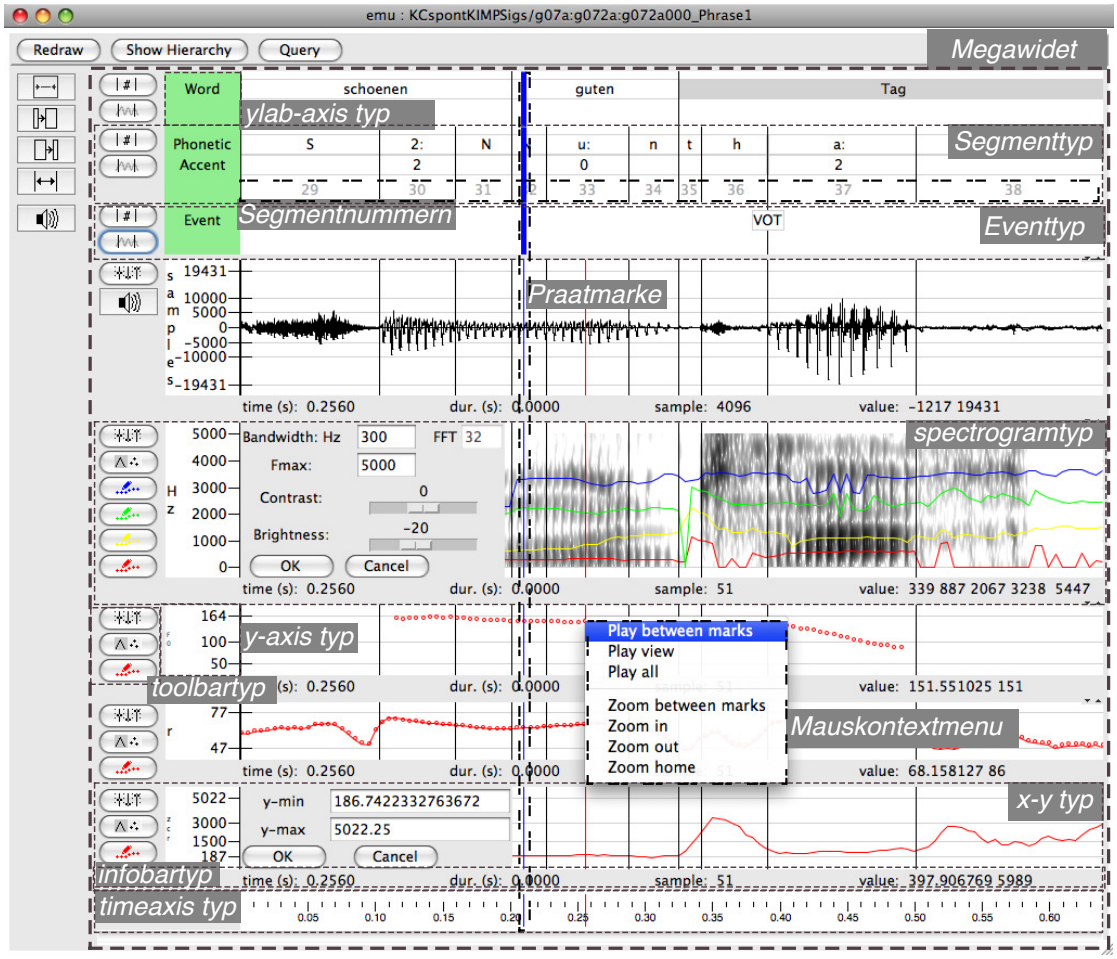
- die Angabe pro Signal vom Zeitpunkt des Mauscursors im Signal in Sekunden
- der Abtastpunkt des oder der Werte an diesem Zeitpunkt sowie der Stelle des Mauscursors
- die Angabe der Dauer eines markierten Bereiches

In der ersten Version des Labellers wurden Signale mit geringer Abtastrate nicht als durchgehende Linien sondern als Punkte dargestellt. Diese Punkte führen bei großen Datenmengen in einer Tcl/Tk Umgebung zu Effizienzproblemen, daher wurde diese Darstellung nur optional, wiederum über ein grafisches Bedienelement, integriert. Weiterhin war es in der alten Version möglich, Formantdaten zu manipulieren. Auch diese Funktionalität musste wieder bereitgestellt werden. Da einzelne Abtastpunkte manipuliert werden, wurden also die Punktdarstellung und das Bewegen der Punkte benötigt. Beides wurde nicht eingeschränkt auf Formanten sondern für alle Signaldarstellungen implementiert.

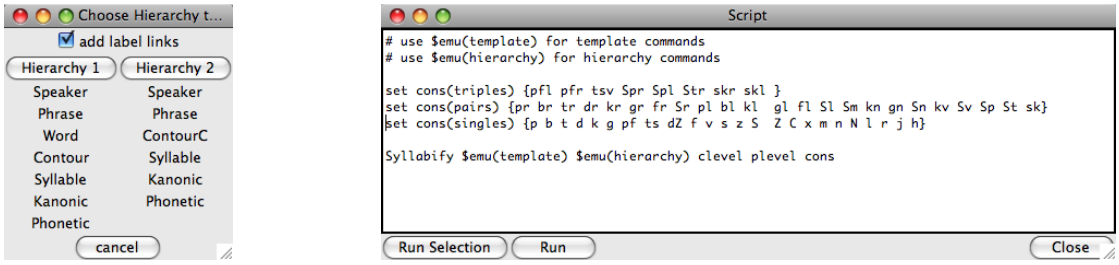
Im Grunde gibt es nur zwei verschiedene Darstellungsmöglichkeiten für Signale. Zum einen für die Signale die Werte über der Zeit abbilden und zum anderen das Sonagramm, das eine dritte Dimension enthält. Von diesen verschiedenen Arten von Signalen können jedoch wiederum mehrere unterschiedliche Instanzen in der Datenbank vorhanden sein und im Labeller dargestellt werden wollen. Für zeitgebundene Etikettierungsebenen gilt ähnliches. Es gibt zwei Arten intervall- und event-zeitgebundene Ebenen (Segment- und Eventebenen) und davon können jeweils mehrere existieren. Aus diesem Grund bietet sich in der Implementierung eine Art Objektorientiertheit an.

Für die Kombination von objektorientierter Programmierung und graphischen Interfaces bietet sich durch die *open source* Tcl/Tk Bibliothek `snit` (Duquette, 2005) eine geeignete Lösung an, die aufgrund von Recherchen gefunden wurde. Mithilfe dieser Bibliothek konnten nicht nur die beiden Signaldarstellungen sondern auch die Darstellungen der beiden zeitgebundenen Etikettierungsebenen sowie ein Container zur Synchronisation aller zeitgebundenen Objekte in einem `snit` *megawidget* implementiert werden. Abbildung 5.4(a) zeigt das neue GUI der `labeller2` Applikation mit der Beschriftung der einzelnen Komponenten aus dem die Oberfläche zusammengesetzt ist. Über eine API können die einzelnen Komponenten außerhalb des Megawidgets angesprochen werden, was die Einbettung in die restliche Struktur des Labellers, der in Abbildung 5.4(a) dargestellt ist, mit Menüfunktionen und externen globalen Bedienelementen möglich machte.

Weiterhin wurde für die bessere Verwendbarkeit und damit einhergehend die neue Funktionalität der an Praat angelehnten Etikettierung eingefügt, sodass ein in Praat üblicher blauer vertikaler Balken zum Setzen von Segmentgrenzen genutzt werden kann. Weiterhin analog zur Praat-Funktionalität wurde ein *Scripting Interface* im



(a) Signal View



(b) Funktionen

Abbildung 5.4.: Das Ergebnis der Neuimplementierung des EMU-Labellers für die Signaldarstellung in (a) mit den Markierungen der einzelnen Bausteine und den neuen Funktionen in (b).

Labeller integriert, das in Abbildung 5.4 dargestellt ist und in Kapitel 5.3.3 näher betrachtet wird.

Für die schelle Verwendung wurden ein Vielzahl an Tastenkombinationen und Menüs bereitgestellt, die mit einem Mausrechtsklick (plattformabhängig) zu erreichen und mit Funktionen versehen sind. Dokumentiert sind diese Bedienungsschnittstellen ebenfalls. In Abbildung 5.5 ist die Dokumentation dargestellt, wobei auch die Tastenkombination und deren Funktion aus der Abbildung hervorgehen.

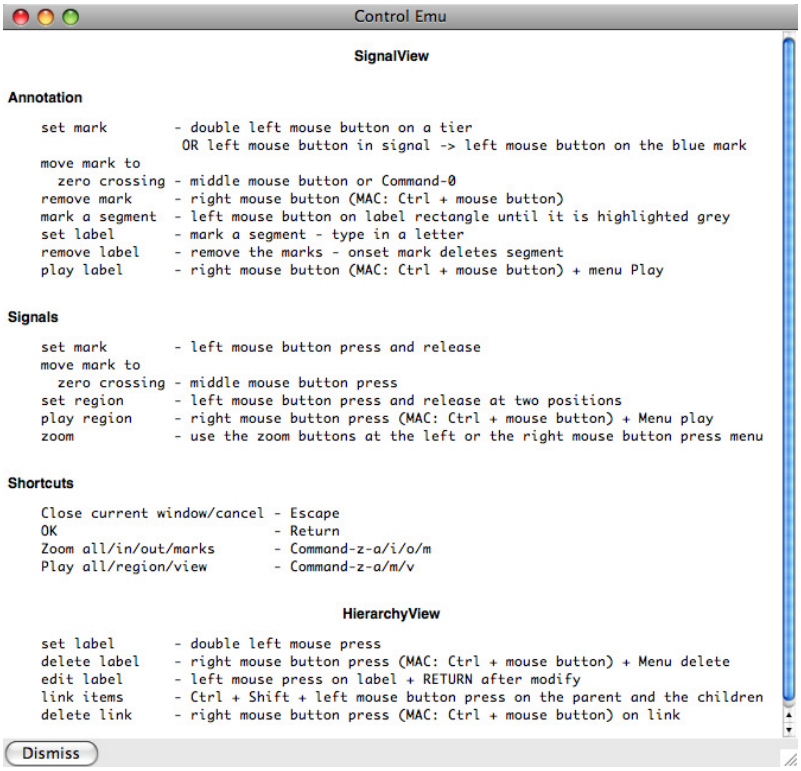


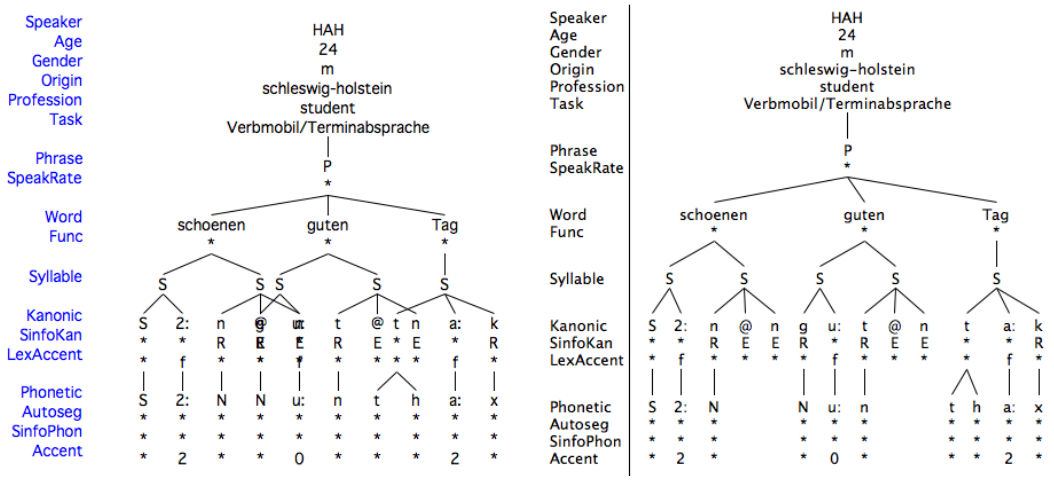
Abbildung 5.5.: Dokumentation der Bedienung des EMU Labellers.

So wurde die Funktion des Setzens der Etikettierungsmarke auf einen Nulldurchgang im Signal hinzugefügt. Hierfür wurde ein entsprechender Algorithmus entwickelt, der anhand des `snack`-Sound-Objektes, der internen Repräsentation des Signals, den nächsten Nulldurchgang sucht. Die Suche erfolgt in beide Richtung vom angegebenen Sample aus. Das dichteste Sample das ein anderes Vorzeichen hat als das ursprüngliche markiert einen Nulldurchgang. Beim Fund vor dem Ursprungssample wird der Abtastpunkt vor dem Nulldurchgang gewählt, während bei einem näheren Fund auf der anderen Seite der Abtastpunkt nach dem Nulldurchgang gewählt wird. Da die Wahrscheinlichkeit eines Samples mit Nullamplitude wenig wahrscheinlich ist,

bietet diese Lösung eine gewisse Konsistenz unter Berücksichtigung von Etikettierungskriterien, wie das Setzen einer Anfangsmarke am Beginn einer Periode.

Alle menügesteuerten Funktionen wurden in den neuen Labeller wieder eingebaut, bis auf die Möglichkeit, Signale einzeln ein- und auszoomen zu können. Dies wurde als nicht nötig erachtet, da mit der Entwicklung des EMU-Datenbank Tools[7], die Möglichkeit gegeben ist, mehrere Labeller gleichzeitig zu öffnen, sodass unterschiedliche Zoomgrade der Signale als Referenz in neuen Fenstern geöffnet werden könnten.

Neben der neuen Signaldarstellung wurde auch die Darstellung der Etikettierungsbäume (Hierarchy View) neu implementiert und mit neuen dokumentierten Bedienungen (vgl. Abbildung 5.5) versehen. So wurde das Verbinden verschiedener Tokens über das Ziehen von Linien durch Tastenkürzel ersetzt, womit gleichzeitig mehrere dominierte Tokens einem dominierenden Token zugeordnet werden können.



(a) Klassischer Hierarchy View

(b) Neuer Hierarchy View

Abbildung 5.6.: Das Ergebnis der Neuimplementierung des EMU-Labeller Hierarchy Views in (b) im Vergleich zur vorangegangenen Version in (a).

Die Darstellung sollte zum einen kompakter aber auch für komplexe sowie sehr einfache Etikettierungsstrukturen symmetrischer sein. Abbildung 5.6 zeigt die Resultate der klassischen und der neuen Implementierungen des Hierarchy Views im Vergleich. In der alten Version gab es horizontale Verschiebungen der Etiketten auf verschiede-

[7]Im Zuge der Entwicklung und im Rahmen dieser Dissertation wurde die Strategie umgesetzt, jede Applikation in einem eigenen Interpreter zu starten. Diese Möglichkeit war vor der Weiterentwicklung nicht gegeben.

nen Ebenen, wenn keine Assoziationen vorhanden waren sowie Überlappungen der Etikettierungen, wenn sie zu viele Zeichen enthielten. Das machte die Arbeit für die manuelle Etikettierung zeitlich langer Äußerungen sehr mühsam, im Besonderen das Verbinden von Tokens[8].

Während in der alten Version des Labellers feste Etikett- und Baumstrukturbreiten sowie ein fester Faktor für das Verbreitern der Bäume definiert sind, wurde in der Umsetzung im neuen Labeller ein Algorithmus entworfen, der aus den Längen der Etiketten, die maximale Breite von Minibäumen berechnet. Minibäume bestehen jeweils aus einem dominierenden Token und seiner Kinder. Hierbei musste beachtet werden, dass das dominierende Token u. U. breiter ist als die aneinandergereihten Kindtokens. Diese Minibäume werden rekursiv wieder als Etiketten angesehen und mit weiteren assoziierten Tokens zu neuen Minibäumen zusammengefasst, deren Breite wiederum berechnet wird. Die Berechnungen werden nach Ketten getrennt von der untersten Ebene zur obersten Ebene vorgenommen. Die einzelnen Baumstrukturen werden dann sequentiell aneinandergereiht unter der Berücksichtigung der Breite nicht assoziierter Tokens oder nicht über die gesamte Struktur assoziierter Minibäume. Eine weitere Berechnung kalkuliert für jedes Token jeweils die Mitte des maximal verfügbaren Bereiches zur Platzierung.

Für die leichtere Bedienbarkeit der im Hierarchy View dargestellten Etikettierungen wurde weiterhin eine neue Auswahlmethode für die darzustellenden Etikettierungsebenen (vgl. Abbildung 5.4(b) links) entwickelt und bereitgestellt. Die im Etikettierungsschema definierten zusammenhängenden Ketten werden zur Auswahl dargestellt, sodass die Ebenen nicht einzeln ausgesucht werden müssen. Der Algorithmus hierfür verwendet den Hierarchiesuchalgorithmus, der für das queryGUI in Kapitel 4.2 entwickelt wurde.

Aufgrund der Unterschiede in der Bedienung und zur Sicherheit von noch nicht berücksichtigten Fällen, ist die neue Darstellung des Hierarchy View nur als Alternative zur originalen Darstellung in den Labeller eingebettet. So können die Benutzer selbst wählen aber auch die neue Darstellung testen und gegebenenfalls Probleme im *Bug Tracker* des EMU-Systems (EMU Developers, 2000) berichten. Ergänzt wurde die Implementierung des neuen Labellers durch eine Dokumentation in Form eines Videotutorials (John, 2010j) für Kapitel 2.1 in (Harrington, 2010a).

[8] So zeigte es die eigene Erfahrung.

5.2.1.2. Fazit

Abbildung 5.4(a) zeigt die grafische Oberfläche des neuen EMU Labellers. Abbildung 5.4(b) zeigt die beiden neuen Funktionen außerhalb der Etikettierung- und Signalansichtsfunktionen. Abbildung 5.5 zeigt schließlich die Dokumentation der Bedienung. Aus diesen Abbildungen geht bereits hervor, dass die Funktionalität des Labellers allein schon anhand der sichtbaren Bedienmöglichkeiten und Informationsfelder enorm erweitert wurde. Bereits durch die neue Darstellung und den Möglichkeiten, welche die `snack` Bibliothek für Signale bereithält, wird die Funktionalität erweitert.

Durch die grundsätzlich beibehaltene Grundstruktur wird ein generelles Prinzip der Oberflächenprogrammierung, die Produktkompatibilität, umgesetzt, zu der Galitz folgendes formuliert:

> "[. . .] combatibility across products must always be considered in relation to improving interfaces, making new systems compatible with existing systems will take advantage of what users already know and reduce the necessity for new learning." (Galitz, 2007, S. 47)

Die steile Lernkurve für die Verwendung des EMU-Systems, die als Ziel dieser Dissertation abgeflacht werden soll, wurde durch die gegebene Kompatibilität und der dadurch reduzierten Notwendigkeit des Neulernens der Bedienung nicht steiler gemacht. Durch die neue Auswahlmöglichkeit zur ursprünglichen Darstellung des Hierarchy Views, der neuen Tastenkombinationen und verschiedenen Einstellungsmöglichkeiten sollte der Labeller im Gegenteil in der Anwendung nicht nur funktionaler sondern auch benutzerfreundlicher sein. Diese Art von umgesetzter Benutzerfreundlichkeit befolgt das Prinzip der "Efficiency" (Galitz, 2007, S. 50). Es schlägt vor, notwendige Augen- und Handbewegungen sowie generell den Bedienungsaufwand so gering wie möglich zu gestalten. Das Einbinden der Etikettierungsmethode nach Praat als weitere Kompatibilitätsmaßnahme erleichtert die ‚Interoperabilität' der Anwender. Nach Hawkins (2011) stehen Überlegungen aus, alle Programme mit einer vergleichbaren Bedienbarkeit auszustatten. Der neue Labeller macht einen Schritt in diese Richtung.

Die Umsetzung des neuen Labellers war im Gegensatz zu anderen Arbeiten eine sehr zeitintensive Aufgabe, da vorhandene Funktionalitäten, die zunächst vollständig bekannt sein mussten, und vorhandenes Design, das den Rahmen vorgab und somit wenig Freiheiten ließ, mit neuen, bisher nicht verwendeten Konzepten nicht nur in der Programmierung umgesetzt werden mussten. Der neue Labeller sowie die damit

einhergehende neue Signaldarstellung für die Zoombarkeit des Sonagramms sowie die Plattformunabhängkeit ist als erfolgreich umgesetzt zu bewerten.

5.2.2. Schneiden

Der EMU Segmenter ist eine der grafischen Benutzerschnittstellen, die bereits vor der Weiterentwicklung vorhanden waren. Sie bietet die Möglichkeit zum Schneiden von Signalen mit der Funktion des automatischen Detektierens geeigneter Signalstücke. Weiterhin unterstützt die Applikation die automatische Vergabe von strukturierten Dateinamen für die extrahierten Signalstücke. Auch die Orientierung an Wortlisten ist möglich. Diese Schnittstelle war in Vergessenheit geraten und dementsprechend nicht zusammen mit den anderen Funktionen weiterentwickelt worden.

Aufgrund von Benutzeranfragen wurde diese bereits vorhandene `capture` Applikation überarbeitet und an das neue EMU-System funktional sowie grafisch angepasst und integriert. Da es sich bei der `capture` Applikation in der Funktionalität um das Schneiden und somit auch das Darstellen von Signalen handelt, basierte es in der alten Version auf der `padgraph` Bibliothek, dessen Funktionalität bereits im Labeller (vgl. Kapitel 5.2.1) ausgetauscht werden musste. Die Methode der Neuimplementierung mit allen notwendigen Vorarbeiten wurde daher auch hierfür gewählt.

Für den Segmenter, der in Abbildung 5.7 dargestellt ist, wurde auf die für den Labeller implementierte Signaldarstellung im `snit` Megawidget zurückgegriffen. Alle Funktionen, die aus dem Quellcode (EMU Developers, 2011a) und der Dokumentation (Simpson, 2003) eruiert werden konnten, wurden im neuen Interface wieder integriert.

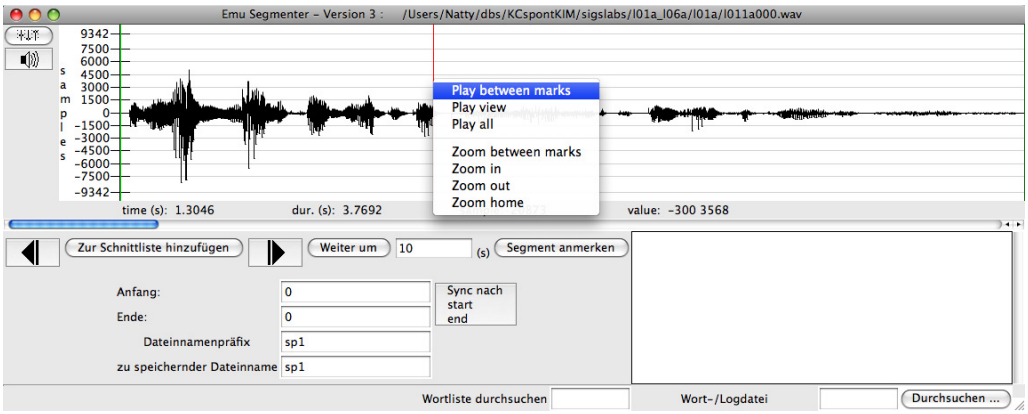


Abbildung 5.7.: Das Ergebnis der Neuimplementierung des EMU Segmenters.

Für die Dokumentation wurde eine ganz neue Methode gewählt, es wurde keine EMU-Browserhilfe umgesetzt, sondern eine World Wide Web Onlinehilfe. Hierfür konnten die bereits für den autoDBinstaller (vgl. Kapitel 5.4) für den Zugriff auf die Datenbanklistendatei implementierten Methoden verwendet werden. Die Onlinehilfe stellt die Hilfe des Segmenters in der ursprünglichen Dokumentation dar, die auf der EMU-Internetseite zur Verfügung steht. Das ist insofern eine geeignete Lösung, da die Neuimplementierung genau diese Funktionalität hat. Als weitere Dokumentation kommen Videotutorials in Frage.

Bisher ist der neue Segmenter noch nicht im EMU-Release vorhanden, aber bereits im *open source* des Programms (EMU Developers, 2011a) verfügbar. In der Testphase hat der EMU Segmenter mit seiner Funktion des automatischen Detektierens von geeigneten Schnittstellen und seiner grundlegenden Funktionen überzeugt. Da diese Arbeit aufgrund von Benutzeranfragen unternommen wurde, kann man auch durch den wieder verfügbaren Segmenter das System als funktional erweitert ansehen.

5.3. Scripten mit Tcl/Tk und EMU-DML

5.3.1. EMU Labeller Module

EMU Labeller Module werden in Cassidy und Harrington (2001, S. 66) als "plugin interface" bezeichnet und sind in Tcl/Tk oder wahlweise C/C++ programmierte funktionale Erweiterungen zum EMU-Labeller (vgl. Kapitel 5.2.1). Unter Verwendung der `labeller2`-API kann u. a. jede Art von grafischer Veränderung am GUI vorgenommen werden. Die im Rahmen dieser Arbeit implementierten Module, die aufgrund von Benutzeranfragen entstanden sind und derzeit mit dem System ausgeliefert werden, sind in Tabelle 5.1 zusammen mit ihrer Funktion aufgelistet.

Das Modul `ExtractHiers` wurde für die Erstellung übersichtlicher Abbildungen in der vorliegenden Arbeit sehr häufig benutzt, um Phrasen aus Äußerungen des Kiel Corpus' zu extrahieren. Das Modul hat sich auch bereits im Rahmen der Lehre sehr bewährt, für das Extrahieren bzw. Separieren guter Sprachbeispiele aus dem größeren Kontext für Transkriptions- und Hörübungen.

Ein Beispiel für die Funktion des `hier2sigView` Modules ist in Abbildung 5.4(a) mit der sichtbaren `Word`-Ebene in der Signalansicht des EMU-Labellers gegeben. Diese

Tabelle 5.1.: Neue EMU Labeller Module und ihre Funktion.

Modul	Funktion
`AppendSegsAtInts`	Tokens können in definierten Zeitabständen automatisch auf einer zeitgebundenen Ebene hinzugefügt werden.
`ExtractHiers`	Kleine Etikettierungsstrukturen können aus Äußerungen extrahiert werden. Die extrahierten Signale und Etikettierungsstrukturen werden direkt in die Datenbank eingepflegt.
`hier2sigView`	Nicht explizit mit der Zeit assoziierte Tokens auf Ebenen können zeitassoziiert dargestellt werden, soweit die Zeiten aus der Etikettierungsstruktur ableitbar sind.

Ebene ist in der Kiel Corpus Datenbank[9] zeitlos und wird trotzdem als vermeintliche intervall-zeitgebundene Ebene dargestellt. Diese Darstellung ist sehr nützlich, wenn die Orthografie die phonetische Transkription unterstützen soll. Das ist besonders bei der Transkription bzw. beim Lesen der Transkription von fremdsprachlichem Material sehr hilfreich. Somit hat es sich ebenfalls sowohl in Lehre und Forschung gut bewähren können.

Das `AppendSegsAtInts` Modul kann Verwendung finden und wurde auch speziell dafür implementiert, um größere Signaldateien in Kombination mit dem `ExtractHiers` Modul zu schneiden. Dieses Verfahren bietet Vorteile gegenüber anderen Methoden, durch die sofortige Bereitstellung der geschnittenen Abschnitte als Äußerungen der Datenbank und ist auch eine skriptbasierte Alternative zum EMU Segmenter (vgl. Kapitel 5.2.2).

Neben den von Entwicklerseite bereitgestellten Modulen sollte jedoch auch das Erweitern des EMU-Labellers für den Anwender benutzerfreundlich gestaltet werden. Hierfür wurde die folgende Strategie entwickelt und auch dokumentiert. Benutzerdefinierte Module sind einfache Tcl-Skripte und werden als EMU Labeller Modul erkannt, wenn sie eine `ELM` Markierung nach FB 3 enthalten und im Emulabel-Modules-Pfad gespeichert sind.

FB 3 (Modulmarke).

```
(* MAIN - freiwählbare Zeichenkette; semantisch
sinnvoll der Funktionsname der Funktion, die das
Modul startet *)
ELM = '#init_procedure|',MAIN;
```

[9] aus der die abgebildete Äußerung stammt

Hinter diesem Pfad verbirgt sich ein Programmverzeichnis, was aufgrund der erhöhten Sicherheitsmaßnahmen der Betriebssysteme sehr nachteilig für den Benutzer ist. In der Datenbanktemplate können jedoch auch benutzerdefinierte Module deklariert werden (vgl. `VAR` Deklaration auf S. 87). Ein weiterer Nachteil für den Anwender ergibt sich durch die fehlende Dokumentation der Programmierschnittstelle. Die Dokumentation muss jedoch ein nahes Ziel in der Weiterentwicklung des Systems sein.

In Zukunft soll auch der EmulabelModules-Pfad im EMU-System konfigurierbar sein. Für die Konfiguration wird der EMU-Konfigurationseditor (vgl. Kapitel 5.1) verwendet werden. Systembekannte Module haben den Vorteil, dass ihre Deklaration im EMU-Template für mehrere Datenbanken sehr einfach ist, da sie im Grafischen Template Editor (vgl. Kapitel 4.3) automatisch zur Auswahl bereitgestellt werden. Ein Nachteil ergibt sich durch den zentralen Modulpfad dennoch, denn das Modul ist dann von der Datenbank separiert. Der Benutzer bleibt jedoch frei in der Wahl seiner Methode.

Die Methode EMU Labeller Module zu verwenden, hat sich bewährt und unterstützt nicht nur die Darstellung spezifischer Daten, wie von Cassidy und Harrington (2001) vorgesehen, sondern sie unterstützen den Benutzer auch direkt bei der Etikettierung der Daten. Videodaten in der Zukunft auch mithilfe von Erweiterungen zum EMU Labeller zu visualisieren, wurde in der Vergangenheit diskutiert, jedoch noch nicht in Angriff genommen. Statt eines EMU Labeller Modules wäre zu überlegen, eine verlustfreie Interoperabilität mit einem multimodalen Programm wie ELAN (Wittenburg, Brugman, Russel, Klassmann und Sloetjes, 2006) umzusetzen, da unterschiedliche Tools unterschiedliche Anforderungen erfüllen und auf diese spezialisiert sind. Derartige Spezialisierungen könnten ausgenutzt werden, anstatt sie selbst neu zu implementieren.

5.3.2. Stapelverarbeitung von AutoBuild Skripten

AutoBuild Skript AutoBuild Skripte sind benutzereditierte Tcl-Skripte, mit deren Hilfe Etikettierungsstrukturen automatisch aufgebaut werden können. So können Tokens auf verschiedenen zeitgebundenen Ebenen in Abhängigkeit ihrer Zeitmarken oder gänzlich ohne Berücksichtigung der Zeitmarken über ein Lexikon oder aufgrund vorgegebener Assoziationen miteinander verknüpft werden. Weiterhin können Tokens auf einer Ebene automatisch erzeugt und mit vorhandenen Tokens auf einer anderen Ebene assoziiert werden, z. B. zum automatischen Silbifizieren von

Phonemen[10] oder zum Erzeugen von Wörtern. Hierfür werden Ableitungsregeln für dominierende oder zu dominierende Tokens verwendet. Das EMU-System stellt diese Funktionalität in EMU-DML Kommandos bereit. Für eine ausführliche Darstellung sei auf Harrington (2010b) verwiesen.

Aufgrund von Benutzeranfragen wurden die EMU-DML Kommandos in Tabelle 5.2 im Rahmen dieser Arbeit erstellt und in das EMU-System eingebunden. Außerdem wurde die bereits vorhandene Möglichkeit, eine Silbenebene mit Silbentokens automatisch aus Phontokens einer Phonetikebene abzuleiten, um die Funktionalität erweitert, voretikettierte Wortgrenzen durch Worttokens auf einer Wortebene zu berücksichtigen.

Tabelle 5.2.: EMU-DML Kommandos und ihre Funktion.

EMU-DML Prozedur	Funktion
`AddLabelsFromFile`	Einfügen von Tokens auf einer zeitlosen Ebene für mehrere Äußerungen auf Grundlage einer externen Textdatei pro Datenbankäußerung
`AppendTokensFromTSFile`	Einfügen von Tokens auf einer Ebene und deren Label-Links auf Grundlage Tab-separierter Informationen aus einer externen Textdatei pro Äußerung.
`ExtractHierarchy`	Extraktion von Etikettierungsstrukturen mit den alignierten Zeitsignalen und Einpflegen der neuen Äußerung in die Datenbank.

Für die Verwendung von AutoBuild Skripten wurde analog zu Kapitel 4.8 in Harrington (2010a) ein Videotutorial (John, 2010b) erstellt. Es zeigt das Schreiben eines Skriptes zum automatischen Verbinden von Segmenten auf unterschiedlichen Ebenen unter Berücksichtigung ihrer Zeitmarken. Weiterhin wird gezeigt, wie das AutoBuild Skript in der Datenbanktemplate Variable `AutoBuild` wie sie in Kapitel 3.2.3.1 auf S. 87 in der `SCRIPT` Deklaration enthalten ist, deklariert wird.

Stapelverarbeitung AutoBuild Skripte waren vor der Weiterentwicklung des Systems aufgrund der Deklaration im Datenbanktemplate innerhalb des EMU-Labellers

[10] Zur Vereinfachung der Formulierung und zum besseren Verständnis wird bewusst auf die Metasprache verzichtet. Metasprachlich richtig ausgedrückt können Tokens (Silbenabbildungen) auf einer dominierenden Ebene (mit dem Namen Silbe) durch Ableitungsregeln, die von Tokens (Phonemabbildungen) auf einer dominierten Ebene (mit dem Namen Phonem) ausgehen, erzeugt werden.

ausführbar. Da derartige Skripte zum Aufbau von Etikettierungen benutzt werden, ist diese Schnittstelle naheliegend. Der automatische Aufbau von Etikettierungen ist jedoch üblicherweise für die gesamte Datenbank erwünscht. Im Rahmen dieser Dissertation sollte daher eine Stapelverarbeitung für die AutoBuild Funktionalität bereitgestellt werden. Umgesetzt wurde diese Anforderung in der EMU-Applikation

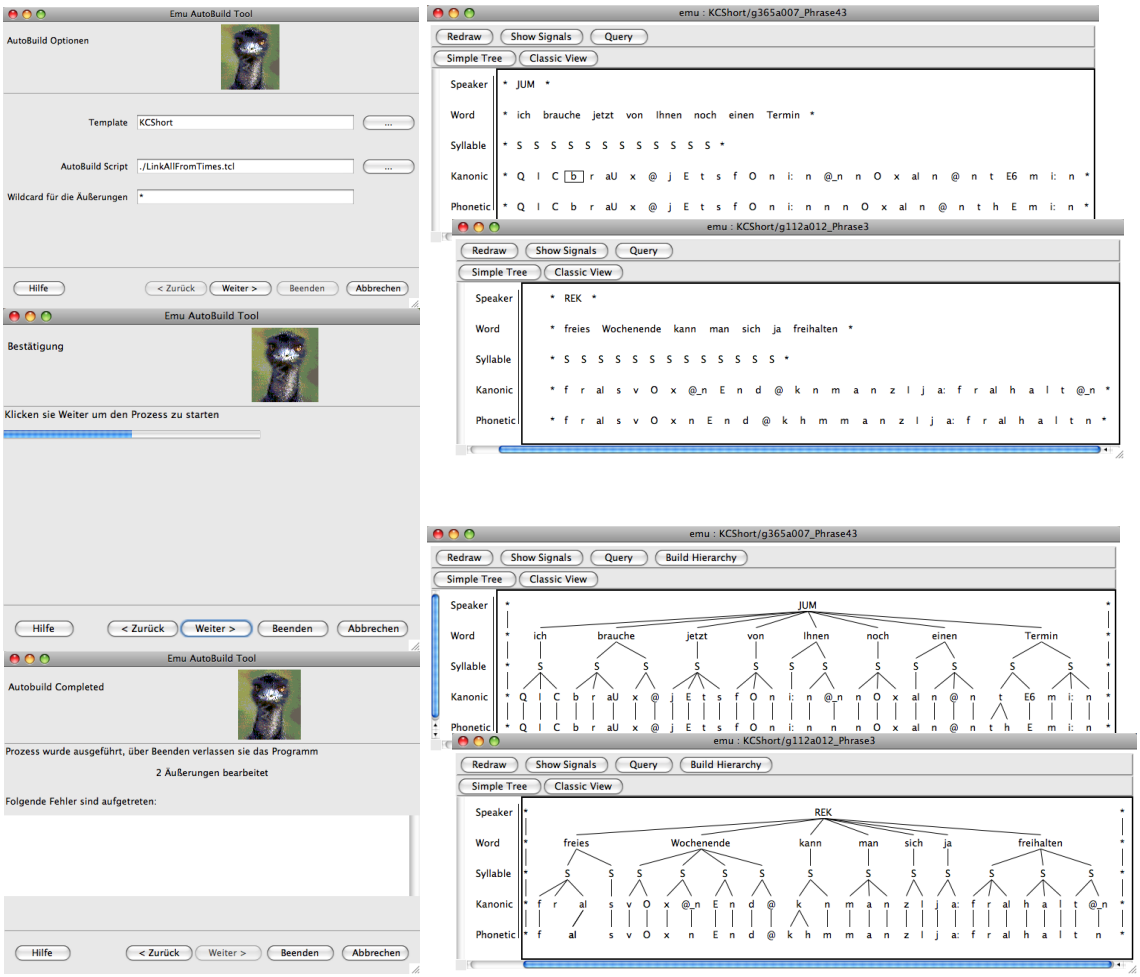


ABBILDUNG 5.8.: Der AutoBuild Wizard mit den verschiedenen Stufen anhand des `LinkAllFromTimes` Autobuild Skript mit dem Ergebnis beispielhaft dargestellt anhand zweier gekürzter Kiel Corpus Äußerungen.

AutoBuild Wizard im `autobuild` Paket (vgl. EMU-Quellcode (EMU Developers, 2011a)), einem Wizard, der durch die notwendigen Schritte führt. Abbildung 5.8 zeigt den AutoBuild Wizard. In der Implementierung wurde die EMU-DML für die Anfragen der Datenbankäußerungen verwendet. Jede Äußerung wird über DML Befehle ins DBMS geladen und das AutoBuild ausgeführt. Mit grundlegenden Tcl/Tk Befehlen wurden die Applikation und die grafische Oberfläche gestaltet.

Durch die Bereitstellung des grafischen Interfaces konnten Benutzereingaben berücksichtigt werden. Dazu gehören die Auswahl eines AutoBuild Skripts, welches nicht in der Datenbanktemplate deklariert ist und die Auswahl der Datenbankäußerungen, auf die das Skript angewendet werden soll. Für den Benutzer ist die Anwendung des GUIs durch die Auswahl der Applikation an sich im Prinzip schon erklärt. Aus diesem Grund wurde auf eine textliche Dokumentation über die EMU-Browserhilfe oder ähnliches verzichtet. Es wurde zur Dokumentation ausschließlich ein Videotutorial (John, 2010a) erstellt.

Sowohl die im Rahmen dieser Dissertation entstandenen Prozeduren als auch die Stapelverarbeitungsfunktionalität haben sich besonders beim Aufbau der Kiel Corpus EMU-Sprachdatenbank, wie in Kapitel 7 erläutert wird, bewährt.

5.3.3. EMU-Script

Um Skriptlösungen mit AutoBuild-Funktionen auch äußerungsindividuell und direkt ohne externe Skripte und Deklarationen in der Templatedatei vornehmen zu müssen, wurde die auf Tcl/Tk basierende EMU-Applikation als `EMU-script` im Rahmen dieser Dissertation entwickelt und im EMU-System integriert (vgl. Quellcode (EMU Developers, 2011a)). Sie wurde direkt im EMU Labeller integriert und orientiert sich an der Funktion des Skriptens in Praat (Boersma, 2002).

Es wurde also eine grafische Oberfläche entwickelt (vgl. Abbildung 5.4(b)), die einen Editor darstellt, in den Befehle eingegeben werden können. Die Funktionalität wurde so implementiert, dass geschriebene Skripte, aber auch nur Teile von Skripten, direkt über die Oberfläche ausgeführt werden können. Das Resultat bei Befehlen, welche die Datenbankausprägung ändern, kann damit direkt im EMU Labeller sichtbar gemacht werden. Für diese Zwecke wurde der Editor mit einer direkten Schnittstelle zur DML bzw. zum DBMS versehen. Auf eine explizite Dokumentation wurde bisher verzichtet, da der Editor selbsterklärend sein sollte.

Mit dieser ad-hoc *Scripting*-Möglichkeit können Funktionen, die später in AutoBuild Skripten auf die gesamte Datenbank angewendet werden sollen, zunächst getestet werden. Aber auch für äußerungsindividuelle Änderungen in der Etikettierung, die manuell aufwendig sind, ist diese Applikation sehr nützlich.

5.4. Installation und Transfer von EMU-Datenbanken

EMU–Sprachdatenbanken müssen portabel sein. Dieser Fakt lässt sich leicht begründen, ohne in die Tiefen der Arbeitsstrategien und Kooperationen von Forschern in der Phonetik und Linguistik gehen zu müssen. EMU–Sprachdatenbanken werden meist auf lokalen Rechnern erstellt, wobei die Sprachsignale aus externen Quellen, wie dem Tonstudioaufnahmerechner oder ähnlichen stammen können. Somit ist bereits nach einer Neuinstallation des Betriebssystems des lokalen Systems eine Neuinstallation der Datenbanksoftware als auch der Datenbank selbst aus einer Datensicherung notwendig.

Die Erstellung etikettierter Korpora bzw. Datenbanken ist sehr zeitintensiv, sodass meist eine Archivierung angestrebt wird. Außerdem werden derartige Daten unter Forschern ausgetauscht bzw. sie sollten ausgetauscht werden können.

EMU–Sprachdatenbanken waren aufgrund der Organisation der Daten in frei zugänglichen Dateien bereits vor der Weiterentwicklung der Software zwischen Systemen transportierbar. Um eine Datenbank auf einer neuen Maschine zu installieren und in das lokale EMU-System einzupflegen, sind folgende Schritte notwendig, wenn davon ausgegangen wird, dass beim Überführen von mehreren Dateien ein komprimiertes Archiv verwendet wird:

1. das Archiv in ein Verzeichnis speichern
2. das Archiv in ein Verzeichnis entpacken
3. die Datenbanktemplate aus dem Verzeichnis suchen
4. sämtliche Pfadangaben in der Templatedatei ändern
5. das Datenbanktemplate in ein Verzeichnis kopieren, in dem das EMU-DBMS Templates sucht bzw. das Verzeichnis als EMU-Templatepfad im EMU Konfigurationseditor konfigurieren

Diese Schritte sind von jedem Benutzer des EMU-Systems ohne Probleme durchführbar, wenn die Vorgehensweise bekannt ist. Bereits jedoch am Wissen über dieses Vorgehen könnte der Benutzer scheitern. Außerdem ist die Abfolge der zu erledigenden Aufgaben algorithmusähnlich immer die gleiche. Somit liegt es nahe, diesen Algorithmus zu implementieren und dem Benutzer einen benutzerfreundlichen Austausch von EMU-Datenbanken zu ermöglichen.

Eine weitere Motivation für die Bereitstellung einer benutzerfreundlichen Austauschmethode war die Idee, Beispielsprachdatenbanken als Dokumentation des EMU-

Systems bereitzustellen. Die Auslieferung der EMU-Software mit bereits integrierten Datenbanken stellte sich jedoch aufgrund des damit einhergehenden Umfangs der Datenmenge als schwierig heraus. Das EMU-System wird als *open source* Software nur zum Download angeboten. Die Beispieldatenbanken sollten dann als separate Downloads verfügbar gemacht werden, womit wiederum das Problem verbunden ist, dass das Installieren vorhandener Datenbanken mühsam ist.

Die Bereitstellung einer EMU-Applikation zum Datenbankaustausch und das zur Verfügungstellen von Beispieldatenbanken, die mithilfe der Applikation installierbar sind, war eines der Ziele dieser Dissertation. Die Umsetzung wird im Folgenden erläutert.

5.4.1. Methode

Für die Umsetzung musste zunächst eine geeignete einheitliche Struktur von EMU-Sprachdatenbanken entwickelt werden, damit sie alle gleichermaßen bearbeitet werden können. Weiterhin musste ein Algorithmus implementiert werden, der die in der Aufzählung aus S. 161 bereits dargestellten benötigten einzelnen Schritte zur Installation einer Datenbank umsetzt. Der Zugang zu dieser Funktionalität musste benutzerfreundlich gestaltet werden.

Dem modularen Aufbau des EMU-Systems nachkommend, wurde eine EMU-Applikation als `autoDBinstaller` Paket unter Verwendung der Tcl/Tk Programmiersprache, die in EMU eingebunden ist, implementiert und dokumentiert. Die Applikation bietet eine grafische Benutzeroberfläche an, die in Abbildung 5.10, S. 166 dargestellt ist.

Weiterhin musste eine geeignete Methode für die Bereitstellung der Beispieldatenbanken zum Download entwickelt werden. Hauptsächlich zu überlegen war hierbei, wie der Benutzer ohne weitere Recherchen die Information über angebotene Datenbanken erhält. Zu vermeiden sollte außerdem sein, dass eine Datenbank manuell von einem Server auf das System heruntergeladen, auf dem eigenen System abgespeichert und dieser Speicherort dann in der Applikation ausgewählt werden muss.

Wenn man der Auffassung von Kemper und Eickler (2006) über eine Datenbank folgt (vgl. Kapitel 2.1), besteht die Datenbank aus der Datenbasis selbst und dem Datenbankschema. Für EMU-Datenbanken (vgl. Kapitel 3.2) bedeutet dies, dass ein Austausch von Datenbanken durch die Übergabe des Datenbanktemplates, der

Etikettierungen und der Signale, alles als physische Dateien, in einer vom Datenbanktemplate vorgegebenen Speicherstruktur bewerkstelligt werden kann.

Dieser Ansatz wurde in der EMU-Applikation autoDBinstaller umgesetzt, wobei eine geeignete Organisation der Daten konzipiert werden musste. Für das Konzipieren einer einheitlichen Datenbankstruktur mussten diese für das EMU-System bekannt sein. In den vorbereitenden Arbeiten wurden die Konzepte eruiert und als Arbeitsgrundlage modelliert sowie grafisch aufbereitet. Die Vorarbeiten sind in Kapitel 3 dargestellt.

Es wurde sich für die folgende Struktur der Dateienorganisation entschieden: Alle Dateien, die den Datenbankzustand physisch speichern, müssen in einer Verzeichnisstruktur liegen, die ein gemeinsames Wurzelverzeichnis teilt. Dieses Wurzelverzeichnis muss außerdem die Datenbanktemplatedatei enthalten. Für den Austausch über das Internet oder andere Medien kann das Wurzelverzeichnis zip-komprimiert (PKWARE, 2004) werden. Eine zip-Komprimierung ist, soweit bekannt, auf allen Plattformen verfügbar.

Unter der Bedingung dieser Datenbankstruktur wurden nun jeweils getrennte Algorithmen für jeden Schritt der bekannten Vorgehensweise zum Installieren (vgl. oben) von Datenbanken entwickelt. Weitere Algorithmen implementieren die Reihenfolge der durchzuführenden Schritte, die unter Umständen verschieden sein können, was sich durch die unterschiedlichen Möglichkeiten des Datenaustausches erklären lässt.

Schritt 1 ist für die automatische Installation von Datenbanken anders zu formulieren nämlich zu: Speicherort des Archivs vom Anwender erfragen. Dieses geschieht in der Umsetzung über die Eingabemaske des autoDBinstallers. Da sich innerhalb eines Verzeichnisses mehrere Datenbanken befinden können, muss auch die Angabe der Datenbank getrennt erfolgen. Dieses Vorgehen hat den Vorteil, dass zip-Archive und Verzeichnisse ohne Komprimierung gleichermaßen als Datenbank behandelt werden können, wobei für letztere Schritt 2 entfällt.

Für die Umsetzung von **Schritt 2**, der Dekomprimierung von zip-Archiven, konnte auf ein Tcl/Tk-Erweiterungspaket zurückgegriffen werden, das durch Recherchen gefunden wurde. **Schritt 3** ist durch die entworfene Datenbankdatenorganisation trivial. Es werden die Dateien verwendet, die im Wurzelverzeichnis die datenbanktemplateübliche Dateiextension (*tpl*) tragen. Mehrere Datenbankschemata auf einer Datenbasis sind möglich, z. B. für die Verwendung der selben Signale für unterschiedliche Untersuchungen, die unterschiedliche Etikettierungsstrukturen vorsehen.

Für das in **Schritt 4** durchzuführende Anpassen der Dateipfade, die im Datenbanktemplate deklariert sind, musste eine Routine implementiert werden, die unter Berücksichtigung der Deklaration, der im Archiv vorhandenen Dateien und dem neuen Speicherort der Daten die passenden neuen Verzeichnispfade findet. Hierfür wurden für jede Verzeichnistiefe, die sich aus den Angaben in der Template ergibt, alle Pfade um die größte gemeinsame Zeichenkette reduziert und um die Zeichenkette, die das neue Wurzelverzeichnis ausdrückt, erweitert. Hierbei kann es zu Überschneidungen kommen. Um trotzdem die richtigen Verzeichnispfade zu finden, werden die neuen Pfade auf Richtigkeit überprüft, indem nach zu erwartbaren Dateien im Verzeichnis gesucht wird. Kann der Verzeichnispfad nicht eindeutig bestimmt werden, wird die Originalangabe beibehalten und der Benutzer entsprechend informiert.

Um den Benutzer niemals mit der nicht erfolgten Anpassung der Verzeichnispfade in der Templatedatei zu konfrontieren, wurden die Ansprüche an die Datenorganisation speziell für Beispieldatenbanken erweitert, und zwar um die alternative Verwendung von relativen Pfaden in der Deklaration. Diese Alternative kann direkt beim Datenbankaufbau berücksichtigt werden. Jedoch ist es auch möglich, vor einem Austausch der Datenbank die Pfadangaben in relative Pfade umzuwandeln. Relativ ist hierbei in Bezug auf den Speicherort der Datenbanktemplatedatei zu verstehen, der für den Austausch über den autoDBinstaller vorgeschrieben ist, nämlich das Wurzelverzeichnis des Archivs.

Das Datenbanktemplate bleibt im Wurzelverzeichnis der Datenbank, das für **Schritt 5** über die `emuconf`-API (vgl. Abbildung 5.2, S. 144) der EMU-Systemkonfiguration hinzugefügt wird. Die Implementierung ist im *open source* des EMU-Systems (EMU Developers, 2011a) einsehbar.

```
1
2 {aetobi.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/aetobi.zip"}
3 {ae.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/ae.zip"}
4 {andosl.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/andosl.zip"}
5 {ema.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/ema.zip"}
6 {epgassim.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/epgassim.zip"}
7 {epgcoutts.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/epgcoutts.zip"}
8 {epgdorsal.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/epgdorsal.zip"}
9 {epgpolish.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/epgpolish.zip"}
10 {first.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/first.zip"}
11 {gerplosives.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/gerplosives.zip"}
12 {gt.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/gt.zip"}
13 {isolated.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/isolated.zip"}
14 {kielread.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/kielread.zip"}
15 {mora.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/mora.zip"}
16 {second.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/second.zip"}
17 {stops.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/stops.zip"}
18 {timetable.zip "http://www.phonetik.uni-muenchen.de/forschung/EMU/timetable.zip"}
```

Abbildung 5.9.: Inhalt der Datenbanklistendatei mit den im EMU-System frei verfügbaren Datenbanken.

Als Verfahren zum Bereitstellen der Beispieldatenbanken wurde folgendes Konzept entwickelt und implementiert[11]: Das Programm ermöglicht den Zugriff zu auf Servern gespeicherten EMU-Datenbanken. Die URL-Adressen der Datenbanken werden dabei in Form einer Liste in einer Textdatei auf dem Server gespeichert. Die für die leichte Weiterverarbeitung entwickelte Syntax innerhalb der Datei ist in FB 4 angegeben.

FB 4.

```
(* ZIP - Dateiname des zip-Archivs *)
(* URL - URL des zip-Archivs *)
(* LF - Zeilenwechsel *)
LE = '{',ZIP,' ','"',URL,'"','}';
LISTE = {LE,LF};
```

Für den Austausch von Datenbanken können die Datenbankarchive also auf Servern gesichert werden und es ist ausreichend, allein die URL zur Auflistungsdatei für den Austausch z. B. auf einer Internetseite oder per e-mail zu Verfügung zu stellen. Im EMU-System wird beim Aufruf der Applikation automatisch zunächst die Datenbanklistendatei `http://www.phonetik.uni-muenchen.de/forschung/EMU/-databaselist.txt` geladen, die die URL-Adressen zu den frei verfügbaren Beispieldatenbanken aus Harrington (2010a) enthält. Abbildung 5.9 zeigt den Inhalt der Datei.

Wie aus Abbildung 5.10 hervorgehen sollte, wurden folgende Aspekte für die benutzerfreundliche grafische Umsetzung berücksichtigt und integriert:

- Verfügbarkeit der Onlineverbindung
- Verfügbarkeit der Online-Datenbanklistendatei
- Eingabemöglichkeit einer alternativen Datenbanklistendatei
- lokaler Speicherort der Datenbank
- die Auswahl der Installationsschritte
- die Auswahlliste verfügbarer Datenbanken in der Datenbanklistendatei
- die Auswahlliste verfügbarer Datenbanken im lokalen Speicherort
- die Fortschritt- und Fehlerinformation für den Anwender
- die Dokumentation der Bedienung der Oberfläche selbst

Die Verwendung des autoDBinstaller wird im Videotutorial für Kapitel 2.1 in Harrington (2010a) visualisiert (John, 2010i).

[11] vgl. EMU Developers (2011a)

5.4.2. Ergebnis und Fazit

Abbildung 5.10.: Die grafische Oberfläche der EMU-Applikation autoDBinstaller mit den verfügbaren Datenbanken aus der Datenbankliste.

Mit der EMU-Applikation autoDBinstaller besteht nun also die Möglichkeit, EMU-Sprachdatenbanken sehr einfach zwischen Systemen auszutauschen. Die Umsetzung bietet für den Anwender die Freiheit, auch nur einzelne Schritte automatisch ausführen zu lassen. So ist es zum Beispiel möglich, nur die Verzeichnispfade in der Datenbanktemplate ändern zu lassen, was nötig wird, wenn man die Datenbank manuell lokal in ein anderes Verzeichnis verschoben hat.

Die durch die Entwicklung dieses Tools vorgeschlagene und für das Tool notwendige Organisation von Datenbankdaten ist ein Schritt in die Richtung, die EMU-Datenbankdatenorganisation generell zu vereinheitlichen. Für die Verwendung relativer Pfade ist die vorgeschlagene Organisation ebenfalls mehr als vorteilhaft.

Sollte sich die Organisation durchsetzen, könnte in Zukunft jede EMU-Sprachdatenbank mithilfe des autoDBinstallers, der in jedem EMU-System verfügbar ist, ohne Mehraufwand in anderen EMU-Systemen aufgespielt werden. Die Funktion des Datentransfers und somit die benutzerfreundliche Interoperabilität zwischen EMU-Systemen wurde mit der Entwicklung des autoDBinstallers im Rahmen dieser Dissertation erfolgreich umgesetzt.

6. Interoperabilität und Datentransfer

> "Our goal is to enable Emu to be used with any file format which fits the constraints of the internal representations used in the system." (Cassidy, 1999a, S. 64)

Dem Zitat zur Folge erhebt das EMU Speech Database System den Anspruch, einen Anschluss an möglichst viele existierende andere Sprachverarbeitungssysteme und zu vorhandenen Sprachkorpora zu haben (vgl. Kapitel 1.3).

Wie aus Kapitel 3 hervorgeht, ist das EMU-System ein Sprachverbeitungsprogramm, das Sprachdaten wie Sprachsignale, abgeleitete Signale und Etikettierungen datenbankbasiert verarbeitet. Dafür wird für jede Sprachdatenbank ein EMU-Datenbanktemplate mit einem enthaltenen Etikettierungsschema benötigt.

Nach Kapitel 1.1 erfolgt die Bearbeitung von Sprachdaten in Sprachverarbeitungsprogrammen meist nicht auf der Basis von Datenbanken im engeren Sinne. Mithilfe von dateibasierten Programmen wird aus Datenbanksicht eine reine Datenbasis erstellt. Somit gibt es für diese Datenbasis kein Datenbankschema und auch kein Datenbankmanagementsystem, über das die Daten aus der Datenbasis unter Verwendung des Datenbankschemas und einer DML gezielt angefragt und extrahiert werden können (vgl. Kapitel 2.1). Aus diesem Grund kann der Zugriff auf vorhandene Etikettierungen, die mithilfe dieser Programme erzeugt wurden, durch das EMU-System nur über die Dateien erfolgen. Die Datenbasis für die Sprachverarbeitung besteht aus Etikettierungen und ggf. Signalen. Gleiches gilt für mit Sprachverarbeitungsprogrammen aufgebaute ganzheitliche Sprachkorpora.

Im Rahmen einer Interoperabilität muss geklärt werden, ob die Etikettierungs- und Sprachsignaldateien aus anderen Systemen in EMU überhaupt bzw. in welcher Form verarbeitbar sind. Es muss berücksichtigt werden, dass EMU-Etikettierungen in den meisten Systemen nicht verarbeitbar sind, da sich die Eigenschaften der Etikettie-

rungsmodelle zu stark unterscheiden. Weiterhin kann für einen Austausch von Daten geprüft werden, ob die einzelnen Projekte vielleicht bereits Routinen zum Einlesen von EMU-Etikettierungen anbieten.

Vor der Weiterentwicklung des Systems gab es bereits die Möglichkeit des direkten Lesezugriff auf die Datenbasis, die durch verschiedene Programme erstellt wurde, wie zum Beispiel für die Programme ESPS/Waves+ (Schalkwyk, de Villiers, van Vuuren und Vermeulen, 1997) sowie SpeechStation (Sensimetrics Corporation, 2011) und auf die Sprachkorpora TIMIT (Garofolo, Lamel, Fisher, Fiscus, Pallett und Dahlgren, 1993) und ACCOR (European consortium, 1993). Das in diesen Programmen verwendete Etikettierungsmodell bietet in Analogie zur EMU-Terminologie nur zeitgebundene Ebenen an und die Daten werden in externen Etikettierungsdateien gespeichert, die entweder Start- oder Endzeiten aber auch Start- und Endzeiten zusammen mit den Tokens tabellarisch enthalten. Über die Definitionsmöglichkeiten auch der physischen Struktur der Datenbasis[1] in der Datenbanktemplate und angepassten Routinen im EMU-DBMS ist diese Art von Zugriff realisierbar gewesen.

Diese Voraussetzungen sind nicht für alle Sprachverarbeitungsprogramme gegeben, so sind z. B. für Praat (Boersma, 2002) die Etikettierungen auf allen Ebenen zeitgebunden aber für mehrere Ebenen in einer und nicht in separaten Etikettierungsdateien pro Ebene gespeichert. Der direkte Lesezugriff ist nicht anwendbar, da in den Dateien die Daten nicht vom Etikettierungsschema getrennt sind.

Aus diesem Grund sollten weitere Schnittstellen zur Erweiterung der Interoperabilität bereitgestellt werden. Im Rahmen der Dissertation wurde ein indirekter Anschluss an die Sprachverarbeitungsprogramme und Korpora über den indirekten Zugriff, der Konvertierung der Etikettierungsdateien bzw. die Integration von Etikettierungen in eine EMU-Sprachdatenbank, entwickelt. Diese Art von Zugriff ist für das Kiel Corpus und die Software Articulate Assistant (Wrench, 2003; Articulate Instruments Ltd., 2010), die die Verarbeitung von elektropalatografischen Daten unterstützt, sowie Praat umgesetzt worden.

Ebenfalls im Rahmen der Dissertation aber in Zusammenarbeit mehrerer EMU-Entwickler wurde auch ein direkter Zugriff auf die Sprachverarbeitungsprogramme Praat und WaveSurfer (Sjölander und Beskow, 2000; Sjölander und Beskow, 2010) realisiert. Im Folgenden werden diese Punkte für die Interoperabilität zu Praat und

[1] vgl. die Datenbankschemadeklaration für **labfile** mit den Attributen **mark** und **format** in Kapitel 3.2

Articulate Assistant sowohl für den indirekten Zugriff als auch für Praat und WaveSurfer für den direkten Zugriff erläutert.

6.1. Allgemeine Methode

Für die Umsetzung der Interoperabilität des EMU-Systems zu den anderen Sprachverarbeitungsprogrammen wurden folgende Punkte für alle Tools eruiert.

1. Zugriffsmöglichkeit auf die Daten
2. Speicherung der Daten
3. Etikettierungsmodell
4. Datenformat der Etikettierungsdateien

Für das EMU-System wurden diese Informationen im Rahmen der Vorarbeiten ermittelt und durch geeignete konzeptuelle Darstellungen als handhabbares Orientierungswerkzeug aufbereitet. Die Konzepte sind in Kapitel 3 zusammengestellt. Für die anderen Programme wurde auf die Dokumentationen aber auch auf die eigene Erfahrung im Umgang mit den Tools zurückgegriffen. Da, wie aus Kapitel 1.1 hervorgeht, die meisten Programme ohne zentrale Verwaltung auf benutzerzugänglichen Dateien arbeiten, waren Punkt eins und zwei bereits beantwortet. Die anderen beiden Punkte waren leicht anhand der Dateien und der Benutzung der Tools zu eruieren und sind in den folgenden Kapiteln jeweils separat dargestellt.

Mit den gewonnenen Informationen und unter Zuhilfenahme der entwickelten Orientierungswerkzeuge wurde nun ein geeignetes *Mapping* für die verschiedenen Etikettierungsmodelle konzipiert und implementiert. Weiterhin wurden Routinen entwickelt und implementiert, die die Etikettierungen eines Tools extrahieren, diese dann auf die Etikettierungsstrukturen des Zieltools abbilden und im zieltoolgeeigneten Format wieder ablegen.

Das Extrahieren der Etikettierungen musste abhängig vom verwendeten Programm umgesetzt werden. Für Tools ohne zentrale Datenverwaltung mussten hierfür die Speicherorte der Etikettierungsdateien erfasst werden, was nur durch eine Benutzereingabe gewährleistet werden konnte, anschließend wurden die Dateien eingelesen und für die *Mapping*routine in geeignete interne Datenstrukturen aufbereitet. Für das EMU-System mit zentraler Verwaltung der Daten wurden die Daten der durch den Benutzer spezifizierten Datenbank über die EMU-DML beim DBMS angefragt.

Gleiches gilt für die Speicherung der Etikettierungen nach dem *Mapping*. Während für Tools ohne DBMS Etikettierungsdateien am benutzerdefinierten Speicherort gespeichert werden mussten, wurde beim Import von Etikettierungen in eine EMU-Sprachdatenbank der Datenbankzustand über die DML direkt durch das DBMS erzeugt. Das DBMS ist dann für die Speicherung der Dateien zuständig. Das hierfür benötigte Datenbankschema wurde jeweils automatisch unter Berücksichtigung interpretierter Etikettierungsstrukturen erzeugt. Die Implementierung der automatischen Erstellung des Datenbankschemas orientiert sich für die Erzeugung von Deklarationen an der eigens dafür erstellten formalen Beschreibung der DDL in Kapitel 3.2.3.

Der benutzerfreundliche Zugriff auf die Interoperabilität musste schlussendlich konzipiert, implementiert und im EMU-System integriert werden. Für die Implementierung jeglicher Interoperabilität wurde die in EMU vorhandene Tcl/Tk-Schnittstelle (vgl. Kapitel 1.3) verwendet. Sämtliche Implementierungen sind in der derzeitigen Version des EMU-Systems enthalten und der Quellcode jeweils im EMU-Quellcode (EMU Developers, 2011a) einsehbar.

6.2. Indirekter Zugriff

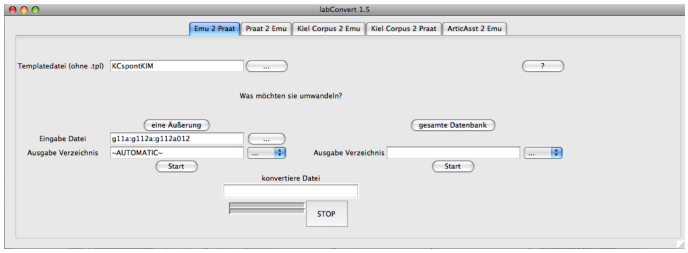
6.2.1. Zugang für den Benutzer

Für den indirekten Zugriff auf die Datenbasis anderer Programme wurde im Rahmen der Dissertation die Applikation labConvert entwickelt. Sie ist als grafische Oberfläche mit verschiedenen Registerkarten realisiert, die als Eingabemaske verschiedener Aufbereitungen genutzt werden kann und die die Stapelverarbeitung für Konvertierungen bereitstellt. Die Applikation ist als Tcl/Tk Applikation über das Dateimenü der primären EMU-Benutzerschnittstelle (`dbemu`) im Frontend des EMU-Systems eingebunden.

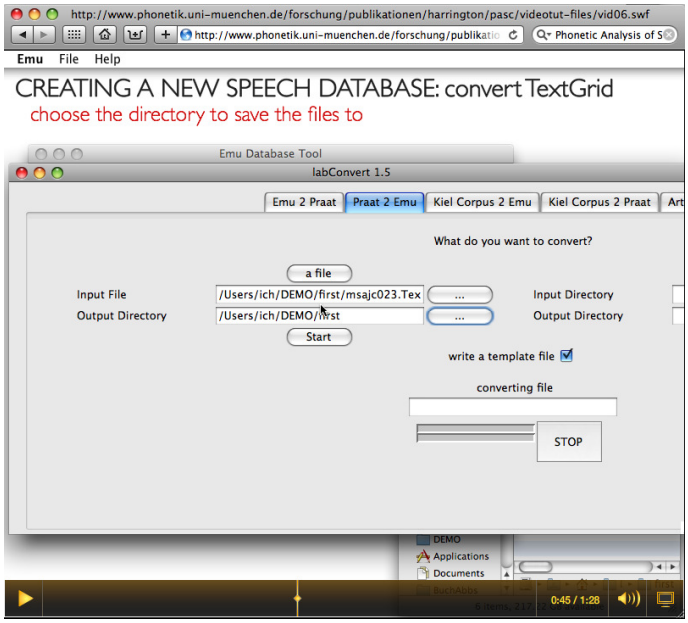
Abbildung 6.1(a) zeigt die Benutzeroberfläche. Wie aus der Abbildung hervorgeht, ist das Aufbereiten von EMU-Etikettierungsdateien in Praat-TextGrids, die umgekehrte Richtung sowie die Aufbereitung des Kiel Corpus für EMU- und Praat-Etikettierungen (vgl. Kapitel 7) und der Etikettierungsdateien aus dem Articulate Assistant Programm bereits in der labConvert EMU-Applikation integriert. Weitere Aufbereitungsarten können ohne weiteres ebenfalls integriert werden.

Das Programmierinterface wurde so gestaltet, dass für jede neue Konvertierungsart nur die auszuführende Funktion mit den Parametern aufgerufen werden soll, wobei die Parameter durch Eingabefelder auf einer neuen Registerkarte vom Benutzer eingeholt werden können.

Zur Dokumentation der Applikation wurde passend zu Kapitel 2.4 in Harrington (2010a) ein Videotutorial (John, 2010d) erzeugt. Das Videotutorial ist beispielhaft in Abbildung 6.1(b) dargestellt ist. Außerdem wurde die Programmierschnittstelle und die Oberfläche textlich dokumentiert (John, 2003c).



(a) labConvert



(b) labConvert Videotutorial

Abbildung 6.1.: EMU-Applikation labConvert (a) mit Benutzereingaben zum Konvertieren einer Äußerung aus der Datenbank KCspontKIM (Kiel Corpus of Spontaneous Speech und unveröffentlichtes Material) und ein Ausschnitt aus dem Videotutorial (b).

6.2.2. EMU und Praat

Etikettierungsmodell und Dateiformat

Das Praat-Etikettierungsmodell sieht nicht assoziierte zeitgebundene Ebenen in unbeschränkter Anzahl vor. Hierbei hat jede Ebene einen Ebenennamen und einen Typ. Unterschieden werden 'Interval Tiers' und 'Point Tiers'. Auf den Ebenen sind eine beliebige Anzahl von Tokens mit Start- und Endzeiten auf 'Interval Tiers' oder nur mit einer Zeitmarke auf 'Point Tiers' sequentiell angeordnet.

Die Etikettierungen liegen in *TextGrid*-Dateien vor, in denen alle Tokens mit Start- und Endzeit für alle Ebenen getrennt strukturiert gespeichert sind. Einen Einblick in das Praat-TextGrid-Format gibt Abbildung 6.2.

Im Header der Datei (vor `item []:`) ist zunächst der Typ der Datei und des Ursprungobjektes in Praat notiert, gefolgt von den zwei Zeitmarken, die den Anfang und das Ende des etikettierten Zeitsignals markieren. Zeile 6 ist in der Form obligatorisch und in der darauf folgenden Zeile ist die Anzahl der Ebenen notiert. Die Daten selbst sind nach Ebenen getrennt in verschiedenen untereinander angeordneten `item`-Blöcken mit dem entsprechenden Index der Ebene organisiert. Jeder dieser Blöcke enthält zunächst den Typ, die `class` der Ebene, gefolgt vom Namen und den Zeitmarken der Ebene. Je nach Typ sind dann die entsprechenden Tokens wiederum untereinander angeordnet in einem `intervals` oder `points` Block angegeben. Nach der Anzahl der Tokens ist für jedes die Start- und Endzeit bzw. nur die Zeitmarke sowie das Etikett notiert. Auf eine formale Beschreibung dieser Speicherstruktur konnte verzichtet werden, da für die automatische Erstellung eine Art Muster-TextGrid verwendet wird, in denen die entsprechenden Angaben nur ergänzt werden.

Mapping EMU zu Praat

Für das *Mapping* von EMU-Etikettierungen in Praat-TextGrids (Harrington, Cassidy, John und Scheffers, 2003) musste die Einschränkung auf zeitgebundene Ebenen und das Nichtvorhandensein von Assoziationsmöglichkeiten in Praat berücksichtigt werden.

Für jede zeitgebundene Ebene in der EMU-Datenbank kann eine äquivalente zeitgebundene Ebene in Praat erzeugt werden, d. h. jede Ebene des Etikettierungssche-

```
File type = "ooTextFile"
Object class = "TextGrid"

xmin = 0
xmax = 95.909375
tiers? <exists>
size = 4
item []:
    item [1]:
        class = "IntervalTier"
        name = "Mary"
        xmin = 0
        xmax = 95.909375
        intervals: size = 3
        intervals [1]:
            xmin = 0
            xmax = 2.2920701033158433
            text = ""
        intervals [2]:
            xmin = 2.2920701033158433
            xmax = 8.550630172553687
            text = "ein Segment"
        intervals [3]:
            xmin = 8.550630172553687
            xmax = 95.909375
            text = ""
    item [2]:
        class = "IntervalTier"
        name = "John"
        xmin = 0
        xmax = 95.909375
        intervals: size = 1
        intervals [1]:
            xmin = 0
            xmax = 95.909375
            text = ""
    item [3]:
        class = "TextTier"
        name = "bell"
        xmin = 0
        xmax = 95.909375
        points: size = 2
        points [1]:
            time = 15
            mark = "ein Event"
        points [2]:
            time = 20.838778601179204
            mark = "zweites Event"
    item [4]:
        class = "TextTier"
        name = "Daniel"
        xmin = 0
        xmax = 95.909375
        points: size = 0
```

Abbildung 6.2.: Ausschnitt einer Praat-Etikettierungsdatei.

mas in EMU wird auch im Praat-TextGrid mit demselben Ebenennamen und Art der Zeitgebundenheit angelegt. Intervall-zeitgebundene Ebenen werden als Praat-Intervall-Tiers und event-zeitgebundene Ebenen als Praat-Point-Tiers umgesetzt. Für die zeitlosen Ebenen in der Datenbank gibt es keine Alternativen im Etikettierungsmodell von Praat.

Zum Darstellen dieser Information auch in Praat wurde für das *Mapping* eine Alternative entwickelt. Für alle Segmente auf zeitlosen Ebenen wird versucht, eine Zeitmarkierung über assoziierte Segmente auf den zeitgebundenen Ebenen zu erhalten. Eine Anfrage an das EMU-DBMS nach den Zeitmarken eines Segments gibt automatisch abgeleitete Zeitmarken zurück, sofern sie verfügbar sind. Jede dieser Ebenen wird aufgrund der Vorgaben des Etikettierungsmodells, in jedem Fall in ein Intervall-Tier, mit demselben Ebenennamen in Praat umgewandelt. Die eigentlich zeitlosen Segmente werden basierend auf ihren abgeleiteten Zeitmarken auf dem Tier abgebildet.

Während in Harrington et al. (2003) noch eine komplette Ebene aus der Konvertierung ausgeschlossen wurde, sobald es ein Segment ohne ableitbare Zeitmarken auf dieser Ebene gab, bezieht sich der Ausschluss aus der Konvertierung in der derzeitigen Version nur auf das zeitlose Segment der Ebene. Das tritt beispielsweise bei getilgten Phonen auf, wenn die Etikettierung eine phonematische und phonetische Transkription enthält, wie in Abbildung A.3. Dann ist das Phonem etikettiert, aber nicht mit einem etikettierten Phon, aus dem die Zeitinformation ableitbar wäre, assoziiert. An dieser Stelle kommt es in der Konvertierung zum Verlust von Informationen, aber während 2003 noch die Etikettierungen auf der gesamten nicht konvertierten Ebene verloren gingen, ist der Verlust heute auf einzelne Segmente beschränkt. Ein getilgter Glottalverschluss führte in Harrington et al. (2003) zum Ausschluss der gesamten `Phoneme`-Ebene. Die Konvertierung der Äußerung g365a007 der Kiel Corpus EMU-Datenbank, die in Harrington et al. (2003, S. 357) als Beispiel dargestellt wird, führt heute zu einem anderen Ergebnis. Dieses Ergebnis ist in Abbildung 6.3 dargestellt.

Ein für das Deutsche typischer initialer Glottalverschluss im ersten Wort `<Ihnen>` taucht in der Praat-Etikettierung in der Abbildung nicht auf, obwohl er auf der phonematischen `Kanonic`-Ebene in der EMU-Etikettierung vorhanden ist (vgl. Harrington et al. (2003) oder Abbildung 7.4 auf S. 204). Weiterhin werden gegenüber 2003 many-to-many Beziehungen während der Konvertierung analysiert und die Etiketten aller Eltern eines Segments verbunden und als ein Segment in Praat dargestellt. Ein

Beispiel zeigt sich in dem Praat-Segment `@_n` auf der `Kanonic`-Ebene in Abbildung 6.3 aus S. 177.

Auf das Löschen von Ebenen wurde verzichtet, um zum einen den Datenverlust zu reduzieren, aber zum anderen auch um die Etikettierungsdateien mehrerer aus einer Datenbank stammender Äußerungen in ihrem ‚Praat-Etikettierungsschema' für alle Äußerungen zu vereinheitlichen. Zuvor konnten diese verschieden sein. Dieser Umstand ist in der automatischen Weiterverarbeitung hinderlich und für eine wiederholte Konvertierung in EMU-Äußerungen äußerst unpraktisch, denn wie sich in Kapitel 6.2.3 zeigen wird, müsste für jedes Etikettierungsschema ein neues Datenbanktemplate entworfen oder geändert werden.

In der Implementierung werden bei diesem Vorgehen sämtliche Informationen über vorhandene Ebenen, die Zeitgebundenheit der Ebenen sowie die Segmente mit Etikett und Zeitmarken auf den Ebenen über die DML beim DBMS angefragt. Die zurückgegebenen Informationen werden zwischengespeichert. Im Anschluss daran müssen verschiedene Berechnungen der Anzahl der verwendeten Ebenen, Gesamtstart- und Endzeit der Ebenen und Etiketten durchgeführt werden. Das sind die Informationen, die in einem Praat-TextGrid enthalten sein müssen. Abschließend wird die gesammelte Information in eine Datei geschrieben, wobei die Struktur einer *TextGrid*-Datei zeilenweise erzeugt wird. Die *TextGrid*-Datei wird auf der lokalen Festplatte gespeichert und kann aus dem Praat-Programm heraus geöffnet werden. Die Signaldateien müssen aus dem Speicherort gelesen werden, der im EMU-Datenbanktemplate (vgl. Kapitel 3.2.2.2) angegeben ist. Hierbei ist anzumerken, dass Praat keine SSFF-Dateien lesen kann. Abhilfe schafft hier eine Signalkonvertierung von SSFF nach wav (EMU Developers, 2011b).

Mapping Praat zu EMU

Für den Import von Praat-Etikettierungen (John, 2003d) wird die Praat-TextGrid Datei eingelesen und hinsichtlich der Anzahl, Art und Namen der Ebenen analysiert. Aus diesen Informationen wird das notwendige EMU-Etikettierungsschema und das Datenbanktemplate ohne die Deklaration der *Tracks* erstellt. Diese müssen vom Benutzer selbst eingefügt werden.

Da jede Ebene in Praat eine zeitgebundene Ebene darstellt, wird für alle Ebenen eine externe Etikettierungsdatei vorgesehen, deren Speicherort durch eine Benutzereingabe vordefiniert ist. Dieser Speicherort wird in der Deklaration der zeitge-

bundenen Ebenen im Datenbanktemplate verwendet. Die Ebenennamen in Praat geben sowohl die Ebenennamen im EMU-Etikettierungsschema als auch die Dateiextension der externen Etikettierungsdateien vor. Bei der Konvertierung mehrerer Praat-TextGrids mit identischen analysierbaren Etikettierungsschema wird nicht für jede Datei ein neues Datenbanktemplate und somit eine neue Datenbank erstellt, sondern es werden neue Äußerungen der vorhandenen Datenbank hinzugefügt.

Die durch die Deklaration erstellte leere Datenbank wird über das DBMS mit Ausprägungen gefüllt, indem die aus der TextGrid analysierten Etiketten mit ihren Zeitmarken auf die jeweilige Ebene eingepflegt werden. In Abbildung 6.4 ist das Ergebnis eines Datenimports dargestellt.

Fazit

Für den Export von EMU-Etikettierungen in das Praat-TextGrid-Format bleibt festzustellen, dass es möglich ist, aber es zu Verlusten an Etiketten und somit Information kommt. Aber die Verluste sind heute geringer als in der ersten Implementierung der Konvertierung in Harrington et al. (2003).

Der Weg des Imports führt zu keinerlei Datenverlust. Im Gegenteil, es können Informationen gewonnen werden, wenn in der Weiterverarbeitung Segmente miteinander assoziiert werden, um die Abfragemöglichkeiten zu erhöhen. Hierfür bietet das EMU-System skriptbasierte Lösungen in Form von AutoBuild Skripten, wie das automatische Assoziieren der Segmente auf verschiedenen Ebenen aufgrund ihrer zeitlichen Alignierung. Mit der Verwendung der ad-hoc Skriptmethode (vgl. Kapitel 5.3.3) können u. a. auch Inkonsistenzen, die aufgrund der dateibasierten Verarbeitung entstanden sein könnten, semi-automatisch behoben werden.

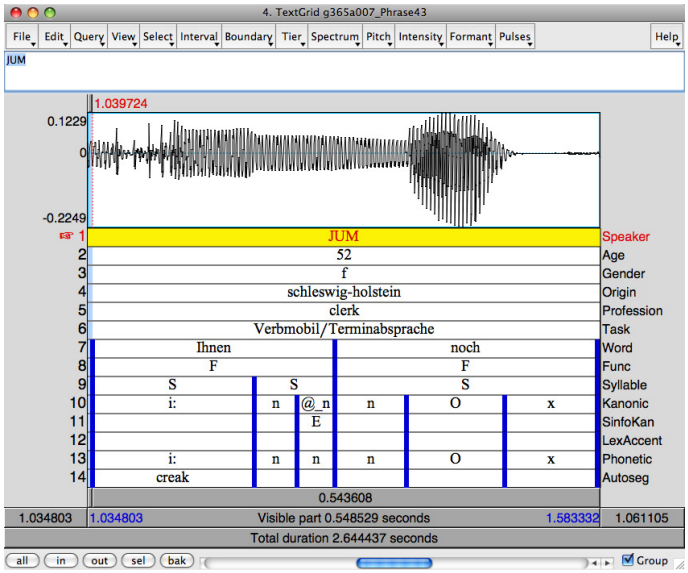


Abbildung 6.3.: Kiel Corpus EMU-Datenbank Äußerung g365a007 nach der Konvertierung nach Praat.

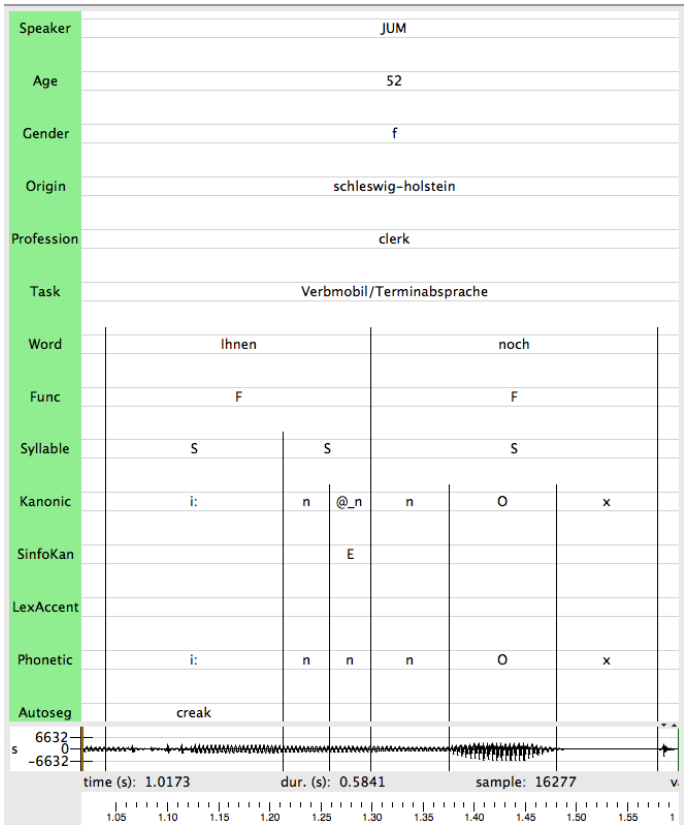


Abbildung 6.4.: Kiel Corpus Praat Äußerung g365a007 nach der Konvertierung nach EMU.

6.2.3. EMU und Articulate Assistant

Der Articulate Assistant (Articulate Instruments Ltd., 2010) ist primär ein Programm zur Aufnahme und umfangreichen Analyse artikulatorischer Daten (z. B. in McLeod, Roberts und Sita (2003)). Die Etikettierung ist nicht das Hauptaugenmerk und auch im Detailreichtum ärmer als es in der Linguistik oder in der Phonetik zu erwarten wäre. Die Abtastraten von artikulatorischen Daten wie EPG, wofür Articulate Assitant genutzt wird, sind sehr viel geringer als die der Sprachsignale, die in der Phonetik verwendet werden.

Etikettierungsmodell und Dateiformat

Abbildung 6.5 zeigt den Inhalt einer Etikettierungsdatei. Jeder Etikettierungsblock in der Etikettierungsdatei enthält beginnend mit einer Startzeit eine Endzeit gefolgt von mehreren Etiketten und endet mit dem Schlüsselwort `[endnote]`. Die Start-

```
1  [starttime:11798]
2  [endtime:13411]
3  /@/ segment vowel central voiced
4  [endnote]
5  [starttime:13447]
6  [endtime:17689]
7  /s/ segment fricative alveolar unvoiced
8  [endnote]
9  [starttime:11820]
10 [endtime:26255]
11 a_soup  phrase
12 [endnote]
13 [starttime:11802]
14 [endtime:13438]
15 a word article
16 [endnote]
17 [starttime:13441]
18 [endtime:26258]
19 soup word noun
20 [endnote]
21 [...]
```

Abbildung 6.5.: Ausschnitt einer Articulate Assistant Etikettierungsdatei.

und Endzeiten sind in Abtastpunkten angegeben aber die Reihenfolge der Etikettierungsblöcke ist weder nach Startzeit, noch nach Endzeit, noch nach Anzahl der Etiketten und auch nicht alphabetisch geordnet. Weitere Anordnungskriterien wurden nicht analysiert. Es wurde eine ungeordnete Aneinanderreihung angenommen, die eventuell auf die Reihenfolge der Eingabe zurückzuführen ist.

Bei näherer Betrachtung der Etikettierungsdatei lässt sich mit fachlich geschultem Auge ein Zusammenhang der Etiketten erkennen. Es scheint, es gäbe eine Phrase `<a_soup>`, die aus den Wörtern `<a>` und `<soup>` besteht, wobei `<a>` als `/@/` und `<soup>` initial mit `/s/` produziert werden sollte. Aus den Etikettierungzseiten ist bereits in den ersten drei Etiketten zu erkennen, dass die Start- und Endzeiten überlappen können. `/s/` hat gegenüber `/@/` zwar sich ausschließende Zeitmarken aber beide Segmente liegen innerhalb des Zeitintervalls von `<a_soup>`. Soweit stimmen die Zeitmarken mit den vermuteten Zusammenhängen überein. Das Wort `<a>` hat jedoch eine leicht frühere Anfangszeit und das Wort `<soup>` eine leicht spätere Endzeit gegenüber der gesamten Phrase `<a_soup>`.

Weiterhin ist aus den Etiketten ersichtlich, dass die Anzahl der Etiketten pro Zeitintervall variieren. Inhaltlich scheinen die Etikettierungen über zusätzliche Etiketten einen Zusammenhang zu bilden, wie bereits angenommen. Diese Tatsachen können von Menschen festgestellt werden jedoch nicht von Algorithmen, denn die Beobachtung unterliegt formal gesehen keiner Strukturierung.

Mapping

Für den ganz ohne Ebenen auskommenden Articulate Assistant mit Etikettierungen, deren Start- und Endzeiten beliebig scheinen, und ein oder mehrere Etiketten haben, ist die Aufgabe des *Mappings* zu EMU besonders. Zunächst müssen alle Segmente auf Ebenen untergebracht werden, so dass sie sich nicht überlappen und so viele Label-Links angeboten werden, wie es zusätzliche Informationen pro Etikett bzw. Zeitintervall gibt.

Dafür wurde ein Algorithmus entwickelt, der die Zeitmarken der Etiketten so analysiert, dass so viele Segmente wie möglich auf eine Ebene gebracht werden können, ohne, dass es Überlappungen gibt. Dafür wurden die Segmente anhand ihrer Startzeiten sortiert und in dieser Reihenfolge kontrolliert, ob sie innerhalb des vorangehenden Segments beginnen. Ist dies der Fall, wird dieses Segment auf eine zusätzliche Ebene gesetzt. Dieses Verfahren des Hinzufügens neuer Ebenen wird solange durchgeführt bis alle Segmente untergebracht sind. Weiterhin wird das Segment mit den meisten zusätzlichen Informationen gesucht. Die Anzahl der Informationen dieses Segments legt die Anzahl der Label-Links zu jeder Ebene fest.

Mithilfe der Information über die Anzahl der Ebenen und Label-Links wird ein Datenbanktemplate geschrieben. Bei der Namensgebung der Ebenen wird sich an

FB 5 orientiert, wobei `INT0` pro neue Ebene und `INT` pro neuem Label-Link um eins erhöht wird.

FB 5 (AA-EBENE).

```
DIGIT = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';
INTP = DIGIT-'0',{DIGIT};
INTPN = '0' | INTP;
(* INT0 jeweils um eins erhöht beginnend mit 0 *)
EBENENNAME = 'ann',{'SE'|'EV'},INTPN;
(* INT jeweils um eins erhöht beginnend mit 1 *)
LABELLINK = EBENENNAME,'_',INTP;
```

Das fertige Datenbanktemplate wird im System über die Konfiguration der möglichen Templatepfade mithilfe der `emuconf`-API (vgl. S. 144) im EMU-System integriert. Über das DBMS werden dann die Segmente auf die Ebenen verteilt.

Werden mehrere Dateien konvertiert, wird für jede Datei ein eigenes Datenbanktemplate erstellt, da das enthaltene Etikettierungsschema dateiindividuell ist. Dennoch unterliegen sie einem gemeinsamem Schema[2], denn unabhängig von der Anzahl der Ebenen haben alle Ebenen pro Schema die gleiche Anzahl an Label-Links. Die Benennung ist hierbei nach FB 5 strukturiert und somit vorhersagbar. Erst nach der Stapelverarbeitung wird ein auf alle Dateien zutreffendes Template ausgewählt oder ggf. erzeugt. Der Algorithmus nutzt die genannte Strukturvorgabe und vergleicht alle vorhandenen Etikettierungsschemata. Dem komplexesten Schema wird jeweils die maximale Anzahl an Label-Links hinzugefügt, die in anderen Etikettierungsschemata gefunden worden sind.

Der Pfad zu den Signaldaten muss anschließend noch vom Benutzer im Datenbanktemplate definiert werden. Der Articulate Assistant wird mit EPG Daten verwendet. Um auch diese der Datenbank hinzuzufügen und das EmulabelModule (vgl. Kapitel 5.3.1) `epgdisplay` zu nutzen, können die EPG-Daten mit einer EMU-Applikation EPG2SSFF umgewandelt und ebenfalls zur Datenbasis ergänzt werden.

Das Ergebnis des Verfahrens anhand der Etikettierungsdatei in Abbildung 6.5 ist in Abbildung 6.6 auf S. 181 dargestellt. Im EMU Labeller erkennt man die leicht unterschiedlichen Startzeiten eigentlich theoretisch assoziierter Segmente. Die Verteilung auf den Ebenen entspricht nicht der impliziten Hierarchie der Etiketten, die aus linguistischer Sicht anzunehmen ist, da der Algorithmus derartige Zusammen-

[2] im Sinne von Struktur

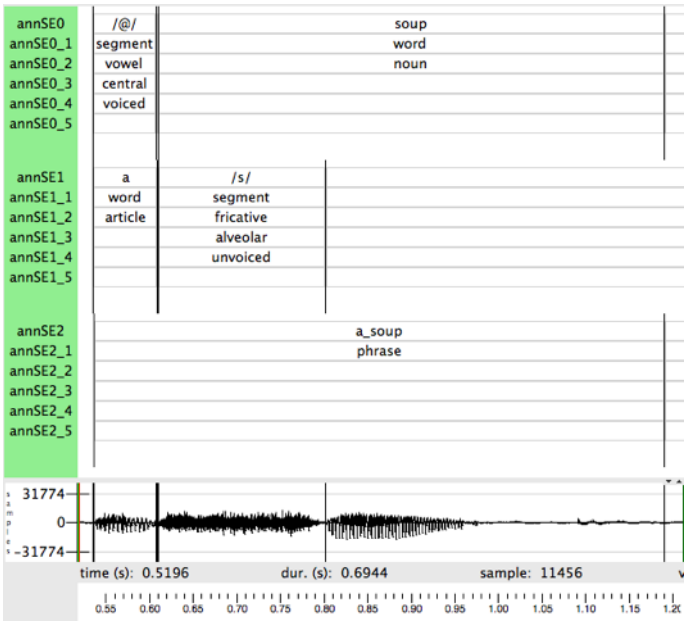


Abbildung 6.6.: Ausschnitt einer Articulate Assistant Beispieläußerungs-Etikettierung J_Smith_1:a soup the chalk the dolls the sugar_1 im EMU Labeller.

hänge nicht interpretieren kann. Etiketten und Zeiten sind in Articulate Assistant frei setzbar, somit könnte alles enthalten sein.

Fazit

Es konnte gezeigt werden, dass das EMU-System, trotz der Voraussetzung, strukturierte Daten zu verarbeiten, mit unstrukturierten Daten umgehen kann, wenn sie durch ein geeignetes *Mapping* aufbereitet werden. Das geeignete *Mapping* wurde im Rahmen dieser Dissertation erfolgreich entwickelt und umgesetzt.

Der Ansatz zeigt aber noch eine andere Seite. Er zeigt, dass das EMU-Etikettierungsmodell Vorteile für die Konsistenz der Daten bringt. Unter Verwendung zeitloser und assoziierter Ebenen in einem geeigneten Etikettierungsschema für eine derartige Etikettierung wären die Zeitmarken für inhaltlich zusammenhängende Segmente identisch und es kommt nicht zu ungewollten Überlappungen, die auf verschiedenen Ebenen dargestellt werden müssten.

6.3. Direkter Zugriff über das EMU-System

Ein direkter Zugriff auf Sprachverarbeitungsprogramme wurde im Rahmen der Weiterentwicklung für Praat (Boersma, 2002) und WaveSurfer (Sjölander und Beskow, 2010) realisiert. Beide Programme teilen die Gemeinsamkeit, eine Kommunikationsschnittstelle für andere Programme anzubieten. Außerdem teilen beide Programme Darstellungs- und Analysemethoden für Teile der in EMU möglichen Etikettierungsstrukturen, die für den direkten Zugriff ausgenutzt werden.

Der direkte Zugriff wurde in der primären grafischen Schnittstelle des EMU-Systems integriert, in der Datenbanken geladen und Äußerungen für die Weiterverarbeitung wie der Darstellung, Etikettierung, Signalanalyse etc. ausgewählt werden können. Für ausgewählte Äußerungen wird durch die Integration als Weiterverarbeitungsmöglichkeit u. a. auch das Öffnen der Äußerung in Praat oder WaveSurfer vom EMU-System bereitgestellt.

Ein Videotutorial für die Etikettierung einer EMU-Datenbank Äußerung in Praat wurde zu Dokumentationszwecken aufgezeichnet (John, 2010k) und steht für Kapitel 2.2 aus Harrington (2010a) auf John (2010m) bereit.

6.3.1. Praat

Praat bietet als Schnittstelle für andere Programme eine Bibliothek `sendpraat` (Boersma und Weenink, 2006) an. Sie stellt den Befehl `sendpraat` zur Verfügung, über den beispielsweise aus dem EMU-System Praat-Befehle an Praat gesendet werden können. Wie aus Kapitel 1.1 hervorgeht, werden in Praat verschiedene Arbeitsschritte durchgeführt, um ein Signal mit einer Etikettierung zu betrachten: Die Signal- und Etikettierungsdateien werden zunächst geladen, die daraus entstehenden Objekte markiert und der Nutzer fordert zum Öffnen des TextGrid-Editors auf. Genau diese Schritte werden für die Umsetzung des direkten Zugriffs von EMU auf Praat mittels `sendpraat` an Praat geschickt. Die Information der zu öffnenden Signaldatei wird mithilfe der EMU-DML beim EMU-DBMS angefragt, das den gesamten Dateipfad zurückgibt.

EMU-Etikettierungsdateien sind von Praat nicht lesbar, daher muss eine Konvertierung der EMU-Etikettierung im Praat-Etikettierungsformat erfolgen. Hierfür werden nur die zeitgebundenen Ebenen wie in Kapitel 6.2.2 konvertiert. Jedem Etikett wird dabei zusätzlich die Segmentnummer, die im logischen Datenbankschema der

Etikettierung Segmente eindeutig identifiziert (vgl. Kapitel 3.2.2.2), mit einem Separator vorangestellt, wie in FB 6 formal und in FB-B 2 als Beispiel dargestellt. An diesem Teil des Etiketts wird sich später beim Finden der Änderungen gegenüber dem Datenbankzustand und beim Einpflegen von Änderungen orientiert.

FB 6 (Praat-Etikett).

```
(* SNR - Segmentnummer *)
(* ETIKETT - das Etikett des Tokens *)
PRAATETIKETT = SNR,'__',ETIKETT;
```

FB-B 2 (Praat-Etikett Bsp.). `348__aI`

Die *TextGrid*-Datei kann wie in Praat üblich verwendet werden. Es können Segmente hinzugefügt, gelöscht oder Segmentgrenzen verschoben werden. Für das Einpflegen der Änderungen musste eine Strategie überlegt werden, die diese Funktion direkt von Praat aus verfügbar macht. Umgesetzt wurde folgende Methode: Es wird vom EMU-System aus ein *Button* zum Speichern von EMU-Etikettierungen in das Dateimenü des TextGrid-Editors hinzugefügt. Dieses Verfahren wird bereits bei der Konfiguration des EMU-Systems bei der Erstbenutzung der EMU–Praat-Schnittstelle durchgeführt, indem der Praat-Konfigurationsdatei der Eintrag bezüglich des Menüpunktes angehangen wird. Zum Speichern der Änderungen wird über den Menüpunkt ein Praat-Skript aufgerufen, das wiederum ein Tcl-Skript aufruft, welches wiederum ein Kommando an das EMU-System zum Analysieren der Änderungen der TextGrid gegenüber dem Datenbankzustand und zum Einpflegen dieser Änderungen schickt. Der hierfür implementierte Algorithmus extrahiert die Segmentnummern und das Etikett aus den Praat-Etikettierungen. Im Anschluss daran werden die Etiketten im Datenbankzustand mit denen aus Praat pro Token verglichen und ggf. über das DBMS verändert. Weiterhin werden hinzugefügte Etikettierungen detektiert, dadurch dass sie keine Segmentnummer haben und als neue Segmente eingepflegt. Auch nicht mehr vorhandene Segmente werden über das DBMS aus dem Datenbankzustand entfernt.

Dieses Verfahren ist verlustfrei, da zeitgebundene Ebenen in beiden Programmen auf die gleiche Weise verarbeitet werden. Durch die explizite Angabe der Segmentnummern, kann auch die EMU-Etikettierungshierarchie nach dem Verändern der zeitgebundenen Etikettierung beibehalten werden. Über die verschiedenen Methoden des Datenaustauschs und zum Senden von Kommandos ist die Schnittstelle jedoch sehr fehleranfällig gegenüber Veränderungen in der Praat-Implementierung. Gerade bei der Installation des Speicherpunktes im TextGrid Menü tauchen mit jeder neuen

Praat-Version und jedem neuen Betriebssystem Probleme auf, da diese Konfiguration nicht über `sendpraat` und somit durch Praat selbst organisiert wird, sondern es wird eine auf dem lokalen System befindliche Konfigurationsdatei angepasst. Der Pfad zur Datei, aber auch der Dateiname der Konfigurationsdatei selbst war in den letzten Jahren ständig im Wandel. Besonders bei derartigen Veränderungen sind die EMU-Entwickler auf das Melden von aufgetretenen Fehlern angewiesen. Daher sei auch an dieser Stelle das vorhandene *Bug Tracking System* (EMU Developers, 2000) der EMU-Software erwähnt.

6.3.2. WaveSurfer

Die direkte Schnittstelle zu WaveSurfer ist über ein umfangreiches WaveSurfer-Plug-In (Erweiterungsmodul) realisiert. Genau wie EMU auf Tcl/Tk und `snack` (vgl. Kapitel 3) basierend bietet das Programm eine Programmierschnittstelle (API) für die grafische Oberfläche an und stellt die Erweiterung durch Plug-Ins in den Fokus des Programms (Sjölander und Beskow, 2000). Die Gemeinsamkeiten dieser beiden Sprachverarbeitungsprogramme machen eine Interoperabilität einfacher als zum Beispiel mit Praat. Dennoch sind Unterschiede gegeben. WaveSurfer ist kein datenbankbasiertes System und stellt für die Etikettierung und auch für die Darstellung keine Möglichkeit für zeitlose Ebenen zur Verfügung. Im Unterschied zu Articulate Assistant und in Gemeinsamkeit mit Praat und EMU wird die Etikettierung auf Ebenen vorausgesetzt. Unter diesen Umständen ist es offensichtlich, dass auch in WaveSurfer nur die zeitgebundenen Ebenen dargestellt werden, hier jedoch keine Segmentnummern in den Etiketten notwendig sind, da über das Plug-In direkt Anweisungen an das EMU-DBMS gesendet werden können. WaveSurfer selbst stellt auch das Einlesen vom EMU-Etikettierungsformat zur Verfügung. Unter Benutzung dieser Schnittstelle werden jedoch aus den gegebenen Gründen keinerlei hierarchische Strukturen verarbeitet.

In den letzten Jahren wurde in Betracht gezogen, das EMU-System allein als WaveSurfer Plug-In weiterzuentwickeln. Diese Idee musste im Rahmen der zu entwickelnden Plattformunabhängigkeit des Systems in Betracht gezogen werden, da diese im EMU-System aufgrund der unmöglich plattformunabhängigen Implementierung für die bis dahin verwendete Signaldarstellung nicht gegeben war. Es zeigte sich jedoch sehr schnell, dass die Darstellung, die WaveSurfer übernehmen sollte, nur eine kleine Funktion des Systems darstellt und dass der Umfang an Funktionen des EMU-Systems, die durch diese Arbeit sehr leicht zugänglich und benutzerfreundlich

gestaltet sind, den Rahmen eines einfachen Plug-Ins sprengt. Als der EMU Labeller noch die primäre Schnittstelle des EMU-Systems darstellte, wäre eine solche Umsetzung noch sinnvoll gewesen. Mit der zentralen Verwaltung der in großer Zahl entwickelten EMU-Applikationen, durch das EMU Database Tool und der gelungenen eigenen Implementierung einer Signal- und Etikettierungsdarstellungsmöglichkeit mit dem neuen EMU Labeller, der nunmehr auf Tcl/Tk und Snack Toolkit basiert, ist diese Idee jedoch nicht mehr relevant.

6.4. Vergleich und Fazit

Für alle Programme, für die Schnittstellen entwickelt worden sind, gilt: Sie stellen Etikettierungen nur zeitgebunden dar.

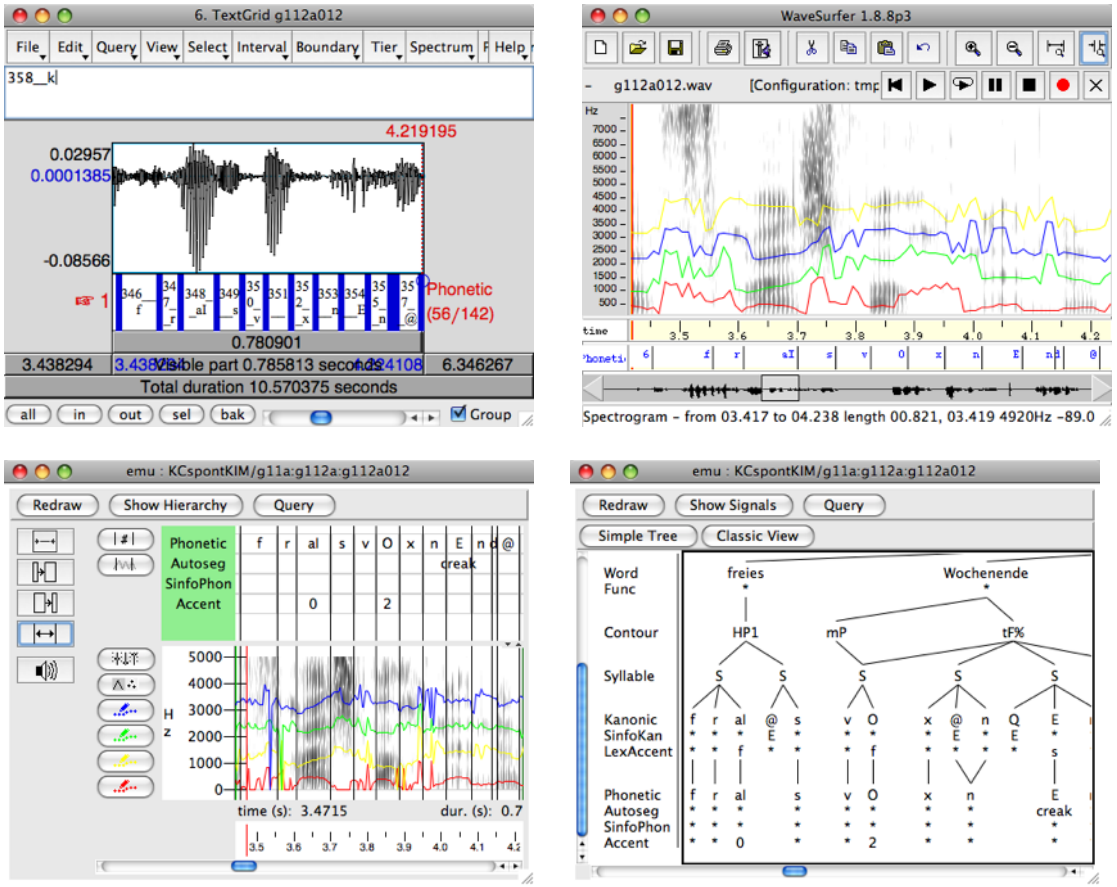


Abbildung 6.7.: Kiel Corpus EMU-Datenbank Äußerung g112a012 nach dem Öffnen mit Praat (oben links), WaveSurfer (oben rechts) und im EMU Labeller (unten), mit der Signaldarstellung (links) und die hierarchische Etikettierung (rechts).

Während Praat und WaveSurfer Ebenen definieren auf denen Tokens sequentiell angeordnet sind, kommt die Etikettierung in der Articulate Assistant Software ganz ohne Ebenen aus und erlaubt dennoch Überlappungen der Segmente.

Im Vergleich der in drei verschiedenen Sprachverarbeitungsprogrammen geöffneten EMU-Äußerung mithilfe der Schnittstellen in der Abbildung 6.7 sind keine größeren Unterschiede auszumachen, soweit es sich um die zeitgebundenen Ebenen handelt. Die Hierarchie der Etikettierungen, wie sie für EMU unten rechts dargestellt wird, kann jedoch keines der anderen beiden Programme in der Darstellungen bieten. Hier zeigt sich deutlich im Vergleich zu Abbildung 6.3 in Praat und Abbildung 6.6 aus der unstrukturierten Articulate Assistant Etikettierung und selbst die Abbildung 6.4 aus der unstrukturierten Praat-Etikettierung, wie viel Information in strukturierter Form unter der Verwendung des EMU-Etikettierungsmodell über eine indirekte Assoziation zum Zeitsignal leicht übersichtlich dargestellt werden kann. Einhergehend mit der Strukturiertheit durch Assoziationen ist auch die damit verbundene vereinfachte und gezielte Abfragemöglichkeit (vgl. Kapitel 3.3).

Insgesamt bleibt festzustellen, dass verschiedene Sprachverarbeitungsprogramme sowohl über das Erstellen von Etikettierungsdateien oder das Importieren in das Datenbankformat als auch über direkte Schnittstellen miteinander kommunizieren können. Über die vorhandenen Schnittstellen kann eine Datenbasis indirekt in mehr als nur einem weiteren Programm benutzt werden. So könnten EMU-Etikettierungen in ELAN, einem multimodalen Signaletikettierungsprogramm, das nicht für phonetische segmentelle Etikettierungen geeignet ist, gelesen werden, wenn sie vorher in Praat-TextGrids konvertiert wurden, denn ELAN bietet den Import des TextGrid Formates an (Brugman und Russel, 2004).

Dateikonvertierungen haben den Vorteil, zwischen Forschern leichter austauschbar zu sein. Das ist notwendig, wenn unterschiedliche Forscher mit unterschiedlicher Software am gleichen Projekt arbeiten. Über die hier dargestellte Interoperabilität der Programme ist aber auch die Arbeit eines Forschers an einem Projekt mit verschiedener Software möglich. Durch die Schnittstellen können die Vorteile und Funktionen jedes Sprachverarbeitungsprogramms nach den jeweiligen Bedürfnissen und Anforderungen der Methode zur Beantwortung von gegebenen Forschungsfragen genutzt werden. So könnte das Schneiden von sehr langen Signalen in Praat oder dem EMU Segmenter durchgeführt werden, die phonetische Transkription der Sprachsignale in WaveSurfer, zusätzliche Etikettierungen von Videodaten in ELAN, weitere zusätzliche Etikettierungen auf Phonem-, Silben-, Morphem-, Wort-, Phra-

senebene durch automatisches Aufbauen einer Etikettierungshierarchie in EMU erfolgen, um schließlich all diese Daten mit einem Programm zur statistischen Analyse wie beispielsweise R (R Development Core Team, 2011) zu analysieren. Hier liegt der entscheidende Unterschied zwischen den Programmen und hier zeigt sich auch die Notwendigkeit der Vielfalt an Programmen: Jedes von ihnen hat seine spezielle Ausrichtung. Die im Rahmen dieser Dissertation entwickelte und vorgestellte Interoperabilität des EMU-Systems trägt einen großen Teil dazu bei, die Spezialisierung der einzelnen Tools geeignet ausnutzen zu können.

7. Kiel Corpus Aufbereitung

Wie aus Kapitel 1.1 hervorgeht, erfolgt die Bearbeitung von Sprachdaten in Sprachverarbeitungsprogrammen wie Praat und WaveSurfer nicht auf der Basis von Datenbanken im engeren Sinne. Mithilfe dieser Programme wird, wie bereits aus Kapitel 6 hervorgeht, aus Datenbanksicht eine reine Datenbasis erstellt, die einen Sprachdatenkorpus bilden. Für diese Datenbasis gibt es kein Datenbankschema und auch kein Datenbankmanagementsystem, über das die Daten aus der Datenbasis unter Verwendung des Datenbankschemas gezielt abgefragt und extrahiert werden können. Aus diesem Grund kann der Zugriff auf vorhandene Korpora nur über die Dateien, die den Korpus bilden, erfolgen.

Ein in der phonetischen Forschung verwendetes Korpus ist das Kiel Corpus (IPDS, 1994; IPDS, 1995; IPDS, 1996; IPDS, 1997a), das unter der Verwendung von Texteditoren und dem Sprachverarbeitungsprogramm xassp (IPDS, 1997b) aufgebaut wurde. Das Korpus enthält, wie jedes Korpus in der phonetischen Forschung, aufgenommene Sprachsignale und die Etikettierungen der Sprachsignale (Kohler, 1992b; Kohler, Pätzold und Simpson, 1995; Kohler, Pätzold und Simpson, 1997). Der Umfang des Korpus an etikettierten Äußerungen ist in Tabelle 7.1 (S. 190) aufgelistet. In den spontansprachlichen Daten enthält jede Äußerung den Sprechakt eines Sprechers während eines längeren Dialogs mit einem weiteren Sprecher (IPDS, 2002a; IPDS, 2002b).

Das Kiel Corpus ist ein rein dateibasiertes Korpus. Um Abfragemöglichkeiten zu gewährleisten, gibt es für das Kiel Corpus eine Überführung in eine netzwerkartige Datenbank KielDat (Pätzold, 1997), die für diese Dissertation jedoch nicht verwendet wurde. Grund dafür ist, dass das KielDat-Datenbanksystem nicht plattformübergreifend verwendbar ist und im Allgemeinen nicht verwendet wird, da es für heutige Verhältnisse nicht benutzerfreundlich ist. Abfragen funktionieren nicht über Abfrageausdrücke, wie sie in dieser Arbeit vorgestellt wurden (vgl. Kapitel 3.3), sondern über die Programmierung kleiner Abfrageprogramme. Die nicht plattformübergreifende Verwendbarkeit war jedoch maßgebend für die Entscheidung, KielDat nicht

Tabelle 7.1.: Anzahl der zur Verfügung stehenden Kiel Corpus Etikettierungsdateien (s1h) ohne prosodische Etikettierung (Read Speech) und mit prosodischer Etikettierung.

Korpusteil	Anzahl	Gesamt
Read Speech	3876	
		3876
Spontaneous Speech Vol I	525	
Spontaneous Speech Vol II	862	
Spontaneous Speech Vol III	597	
		1984
unveröffentlichtes	539	
		539
Gesamt		6399

zu verwenden und auch maßgebend für die Entscheidung, das Kiel Corpus in ein anderes plattformübergreifendes, mit einer Vielzahl an Applikationen ausgestattetes Datenbanksystem, nämlich EMU, zu überführen.

Das große Korpus wird in der Forschung im Rahmen von korpusbasierten Analysen verwendet (Barry, Andreeva, Russo, Dimitrova und Kostadinova, 2003; John und Harrington, 2007; Russo und Barry, 2008). Der Zugang zu den Daten ist nicht über eine Sprachverarbeitungssoftware gewährleistet. Verschiedene Sprachverarbeitungsprogramme können die segmentellen Etikettierungen verarbeiten und darstellen. Das Korpus enthält jedoch Informationen, die nicht mit der Darstellung einer segmentellen Etikettierung abgedeckt sind. EMU bietet aufgrund seines Etikettierungsmodells explizit die Möglichkeit, nicht segmentelle Etikettierungen darzustellen. Aus diesem Grund und für eine leichtere Verwendung des Korpus in der zukünftigen phonetischen Forschung durch eine geeignetere Darstellung der Etikettierung und die daran angepasste Abfragemöglichkeit, die das EMU-System anbieten kann (vgl. Kapitel 3.3), wurde das Korpus im Rahmen dieser Dissertation als EMU-Sprachdatenbank aufbereitet. Diese Aufbereitung wird im Folgenden vorgestellt.

7.1. Material

Grundlage für die Konvertierung des Kiel Corpus in das EMU-Format waren die originalen Signal- und Etikettierungsdateien. Diese Dateien lagen auf den veröffent-

lichten Kiel Corpus CDs (IPDS, 1994; IPDS, 1995; IPDS, 1996; IPDS, 1997a) vor. Weiteres bisher unveröffentlichtes Material wurde vom Institut für Phonetik und digitale Sprachverarbeitung der Christian-Albrechts-Universität zu Kiel bereitgestellt. Der Umfang ist in Tabelle 7.1 zusammengefasst. Zuzüglich zum zu konvertierenden Material auf den CDs lagen die Dokumentationen zum Aufbau des Kiel Corpus vor (Kohler 1992b; Kohler et al. 1995; Kohler et al. 1997) und für die prosodische Etikettierung (Kohler, 1991; Peters, 1999; Peters und Kohler, 2004) sowie die unveröffentlichten Arbeitsprotokolle aus dem Archiv des Instituts für Phonetik und digitale Sprachverarbeitung.

Im Folgenden werden die Etikettierungen konzeptuell und in der Speicherung in den Etikettierungsdateien näher betrachtet. Diese Informationen mussten vor der Entwicklung der Überführung des Kiel Corpus in eine EMU-Sprachdatenbank gesammelt werden. Die Beschreibung diente als Grundlage für das Design des EMU-Etikettierungsschemas und der Zeichenverwendung in der EMU-Datenbank. Die Organisation der Daten innerhalb der Datei musste bekannt sein, um den Inhalt der Dateien für das *Mapping* richtig interpretieren und implementieren zu können.

7.1.1. Etikettierungsschema des Kiel Corpus

Das Kiel Corpus enthält kein explizites Etikettierungsschema, denn den Daten liegt kein explizites Modell zugrunde. Nur die prosodischen Etikettierungen sind modellbasiert (Kieler Intonationsmodell). Die Etikettierungen im Korpus sind nicht als Tokens oder Segmente auf Etikettierungsebenen organisiert. Abbildung 7.1 zeigt eine Etikettierung der KC Äußerung g365a007 in xassp (IPDS, 1997b). Alle Etiketten, die explizit an die Zeit gebunden sind, werden auf einer Ebene linear nebeneinander dargestellt. Die unterschiedlichen Stufen der Etiketten sind reine Darstellungsparameter, um die Etikettsymbole nicht überlappen zu lassen. Trotz fehlendem Etikettierungsmodell und -schema lässt sich aufgrund der Information aus der Literatur und unter Betrachtung der Etikettierungsdateien erläutern, welche Etikettierungen im Kiel Corpus enthalten sind. Diese Erläuterung ist im Folgenden enthalten.

Die Kiel Corpus Etikettierung enthält zunächst die Transliteration des Signalinhaltes. Hierzu gehören die lexikalischen Einheiten, aber auch Pausen, Atmen, starke segmentelle Längungen sowie nicht verbale Einheiten. Weiterhin sind die hochdeutschen Aussprachevarianten der in der Transliteration verwendeten lexikalischen Einheiten enthalten. Diese kanonische Transkription wurde dem Kiel Corpus auto-

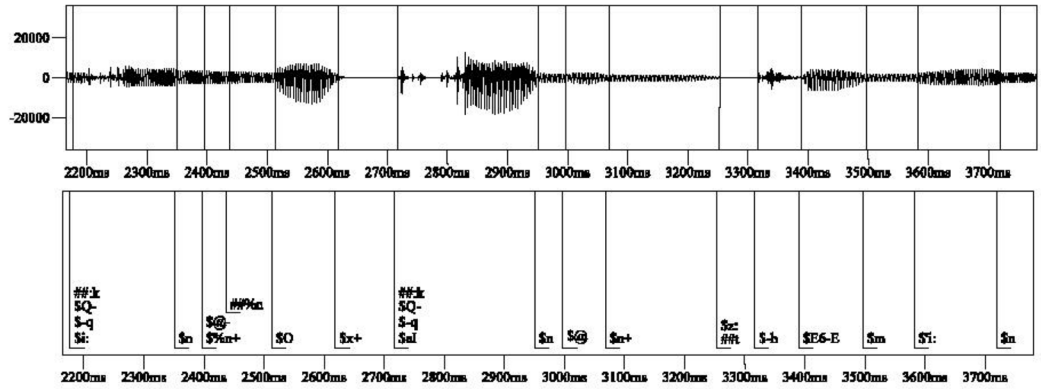


Abbildung 7.1.: Ausschnitt der Kiel Corpus Äußerung g365a007 in der linearen Darstellung der Etikettierungen in xassp.

matisch über den Graphem-Phonem Konverter aus dem RULSYS Text-to-Speech System (Kohler, 1992a) hinzugefügt. Die phonematische Transkription enthält weiterhin die Markierung des Wortakzentes für Wörter, die keine Funktionswörter im Deutschen darstellen. Hierbei wird Haupt- und Nebenakzent unterschieden. Nur Komposita tragen einen Haupt- und einen Nebenakzent.

Veränderungen der im Signal vorhandenen Aussprache gegenüber der automatisch erzeugten Kanonik sind ebenfalls in der Etikettierung enthalten. Hierbei werden Einfügungen, Veränderungen und Löschungen von Segmenten gegenüber der Kanonik unterschieden. Einfügungen können Phone und autosegmentelle Eigenschaften (Knarrstimme und Nasalierung oder beides) sein, während Löschungen und Veränderungen auf Phone beschränkt sind.

Außerdem ist die Segmentierung des Zeitsignals im Kiel Corpus enthalten. Die Segmentierung umfasst intervall-basierend die Phone bzw. das entsprechende Phonem des realisierten Phons und Wörter sowie die lexikalischen Einheiten in Komposita. Konnten in der Segmentierung Segmentgrenzen nicht sicher gesetzt werden, ist diese unsichere Grenze als solche markiert. Unsichere Worterkennung und damit verbundene unsichere Kanonikableitungen sind im Korpus ebenfalls markiert, tauchen jedoch sehr selten auf. Weiterhin sind die Sprechaktanfänge markiert. Es können mehrere solcher Äußerungsanfänge in einer Etikettierung auftreten. Das ist für die Segmentierung der Etiketten, bzw. Segmente in Dialogen notwendig, wenn ein Zeitbereich von einem anderen Sprecher gefüllt ist, denn Dialoge sind für jeden Sprecher

im Dialog separat etikettiert und liegen somit in unterschiedlichen Etikettierungsdateien vor.

Die prosodischen Etikettierungen im Kiel Corpus of Spontaneous Speech und den Lindenstraßendialogen basieren auf dem Kieler Intonationsmodell (Kohler, 1991; Peters, 1999; Peters und Kohler, 2004). Das Modell besteht wie jedes Modell wiederum aus Objekten und Beziehungen und beschreibt die Intonation einer Äußerung über die Objekte Phrasen- und Akzentkonturen sowie Akzentstufen und Sprechgeschwindigkeit. Sämtliche Konturen stehen dabei in einer sequentiellen Beziehung, während die Akzentstufen speziell mit den Akzentkonturen in einer Beziehung stehen, so dass jede Akzentkontur eine Akzentstufe enthält. Die Sprechgeschwindigkeit ist ein Attribut der Phrase. Für das Modell wird eine natürliche Deklination (Absinken) der Grundfrequenz während des Sprechaktes vorausgesetzt. Stetig abfallendes F0 (Grundfrequenz) wird im Modell somit nicht markiert.

Das Sprachmaterial wird in prosodische Phrasen aufgeteilt. Das Modell sieht dabei verschiedene Phrasentypen vor. Innerhalb einer jeden Phrase werden phraseninitiale Konturen (*'pre-heads'*), Konkatenationskonturen und phrasenfinale Konturen neben den Akzentkonturen unterschieden. Das Modell unterscheidet verschiedene Akzentkonturen, die Intonationsverläufe über satzakzentuierten Silben beschreiben, d. h. Silben, die eine stärkere perzeptive Prominenz gegenüber anderen Silben in der Äußerung haben. Dafür werden als Objekte auch verschiedene Akzentstufen unterschieden. Der Intonationsverlauf zwischen Akzentkonturen wird über die Konkatenationskonturen beschrieben, während Verläufe bis zur ersten Akzentkontur durch die phraseninitialen und nach der letzten Akzentkontur bis zum Phrasenende durch die phrasenfinalen Konturen beschrieben werden. Neben Konturen und Akzentstufen enthält das Kieler Intonationsmodell die Angabe der Sprechgeschwindigkeit, die an der Phrase markiert ist.

7.1.2. Speicherung

Die Sprachsignale liegen als Dateien auf den Kiel Corpus CDs in einem RAW-Datenformat (Ohne Definition der Struktur am Anfang der Datei) vor. Auch die Etikettierungen des Kiel Corpus liegen in Dateien vor. Abbildung 7.2 (S. 194) zeigt Ausschnitte von zwei Etikettierungsdateien, zum einen der Kiel Corpus Äußerung g365a007 ohne prosodische Etikettierung und zum anderen die Äußerung g071a003 mit prosodischen Etikettierungen. In der *s1h*-Datei für Äußerung g365a007 (links)

```
g365a007.s1h
JUM007: [...] <:<#Rascheln> Ihnen:> noch
        <:<#Rascheln> einen<Z>:> Termin
        [...] Kalender ?
oend
  [...]  :k Q i: n @ n+  n O x+
  :k Q aI n @ n+ z:  t E 6 m 'i: n  [...]
  k a l 'E n d 6  ?
kend
  c: [...]
  :k Q- -q i: n @- %n+  %n O x+
  :k Q- -q aI n @ n+ z:
  t -h E6-E m 'i: n
  [...]  k -h a l 'E n d 6  ?
hend
    10037 #c:            0.6272500
 [...]
    34834 ##:k           2.1770625
    34834 $Q-            2.1770625
    34834 $-q            2.1770625
    34834 $i:            2.1770625
    37611 $n             2.3506250
    38342 $@-            2.3963125
    38342 $%n+           2.3963125
    38991 ##%n           2.4368750
    40212 $O             2.5131875
    41895 $x+            2.6183750
    43476 ##:k           2.7171875
    43476 $Q-            2.7171875
    43476 $-q            2.7171875
    43476 $aI            2.7171875
    47214 $n             2.9508125
    47957 $@             2.9972500
    49095 $n+            3.0683750
    52052 $z:            3.2531875
    52052 ##t            3.2531875
    53067 $-h            3.3166250
    54239 $E6-E          3.3898750
    55952 $m             3.4969375
    57321 $'i:           3.5825000
    59516 $n             3.7196875
[...]
   346018 ##k           21.6260625
   346826 $-h           21.6765625
   347681 $a            21.7300000
   348767 $l            21.7978750
   349672 $'E           21.8544375
   351237 $n            21.9522500
   352678 $d            22.0423125
   352808 $6            22.0504375
   355656 #?            22.2284375
```

```
g071a003.s1h
HAH003: [...] , Dienstag , dem zweiten , [...]
oend
  [...]  , d 'i: n s t a: k  , d e: m+
  t s v 'aI t @ n  , [...]
kend
  [...]  &2^ d -h 'i: n s t- a: k-x  , &0
  d -h e: m+ &1. &2^
  t s v 'aI t-Q @- n  , &2. &PGn [...]
hend
[...]
    78659 #&2^
    78659 ##d
    79617 $-h
    79913 $'i:
    80839 $n
    81686 $s
    83108 $t-
    83108 $a:
    83843 $k-x
    84181 #,
    84181 #&0
    84181 ##d
    84789 $-h
    85242 $e:
    85776 $m+
    86364 #&1.
    86364 #&2^
    86364 ##t
    87061 $s
    88262 $v
    88948 $'aI
    91230 $t-Q
    91635 $@-
    91635 $n
    92972 #,
    92972 #&2.
    92972 #&PGn
[...]
```

Abbildung 7.2.: Etikettierungsdatei g365a007.s1h der Kiel Corpus Äußerung g365a007 im originalen Format (links) und g071a003.s1h der Äußerung g071a003 (rechts).

lassen sich vier unterschiedliche Blöcke an Informationen ausmachen. Die ersten Blöcke, die den Dateiheader darstellen, werden mit den Schlüsselwörtern *oend*, *kend* und schließlich *hend* (Ende des Headers) abgeschlossen.

Der mit *oend* abgeschlossene Block enthält die Transliteration des Signalinhaltes in deutscher Orthographie und Interpunktion (vgl. Abbildung 7.2). Die Transliteration der Signale auf diese Weise unterliegt gewissen Konventionen (Kohler et al. 1994; Kohler et al. 1995), aufgrund derer eine rechnergestützte automatische Erfassung der Information möglich sein sollte. Nicht verbale Ereignisse werden wie in `<:#Rascheln>` in spitzen Klammern mit der Angabe des Ereignisses in deutscher Orthographie markiert.

Im zweiten Block, der sich in den *s1h*-Dateien zwischen den Schlüsselwörtern *oend* und *kend* befindet, sind wie in Abbildung 7.2 im Vergleich zur Transliteration zu sehen ist, die kanonischen Formen notiert. Ein Beispiel zeigt das Wort `<Ihnen>` aus dem ersten Block. Es ist mit `/Q i: n @ n/` in Zitierform notiert. Den verwendeten Zeichen liegt ein modifiziertes SAMPA (Wells, 1987) zugrunde. Mit den Modifikationen (Kohler et al. 1995) entspricht die gegebene Transkription `/Q i: n @ n/` in IPA /ʔiːnən/ (IPA, 1999).

Die Transkription ist weitgehend phonematisch, wobei dem Glottalverschluss, entgegen den meisten phonologischen Beschreibungen (z. B. Hakkarainen (1995), Hirschfeld (2003)) des Deutschen, ein Phonemstatus zugewiesen wird.[1] Ähnlich verhält es sich mit [ɐ], das im Wort `<Kalender>` als `/6/` transkribiert wurde, obwohl es als /ʁ/-Realisierung im Silbenreim vorhersagbar ist (Wiese, 2000; Hirschfeld, 2003). Ein weiteres Beispiel einer nicht phonematischen Transkription findet sich im Wort `<noch>` `/n O x/` ≙ /nɔx/ in Abbildung 7.2 im Vergleich zu dem Wortsegment `<sich>` in Abbildung A.3, das mit den Kanonicsegmenten `/z I C/` ≙ /zɪç/ assoziiert ist. Aus diesen Beispielen geht hervor, dass die beiden allophonischen Varianten [ç, x] von /x/ anstatt des Phonems selbst verwendet werden. Weitere allophonische Varianten wie [χ] werden jedoch nicht berücksichtigt.

Die phonematische Transkription der Wörter `<Termin>` und `<Kalender>` sind Beispiele in Abbildung 7.2, in denen der Wortakzent markiert ist. Die Markierung geschieht mit einem Apostroph vor dem Vokal der wortbetonten Silbe. Das Wort `<Wochenende>` aus Abbildung A.3 im Anhang ist über die Zeichenkette `/v 'O x @ n #Q "E n d @/` transkribiert. Als Kompositum trägt es einen Hauptakzent `'` und einen Nebenakzent `"`. Die Grenze zwischen beiden lexikalischen Einheiten im Kompositum ist durch `#` markiert.

Funktionswörter, die ohne Akzentmarkierung stehen, werden durch das angehängte `+`-Zeichen markiert. Auch die nicht verbalen Ereignisse, die in der Transliteration bereits auftauchen, werden in der Kanonik markiert, wobei sie hier anders markiert sind als in der Orthographie. So entspricht die Zeichenkette `<:#Rascheln>` in der Orthographie über dem Schlüsselwort *oend* `<:k` zwischen den Schlüsselwörtern *oend* und *kend*. Die verschiedenen verwendeten Etiketten und deren Bedeutung sind im Anhang B.2 aufgelistet.

[1] Der Glottalverschluss ist am Morphem- und Wortanfang und im Onset betonter Silben vor Vokal vorhersagbar oder ist optional. Es gibt im Deutschen jedoch auch wenige Minimalpaare wie /fɛʁaɪzən/ , /fɛʔaɪzən/. In der SAMPA des Deutschen wird er aber in Gibbon (1997) explizit als Phonem angenommen.

Bevor mit *hend* der Dateiheader geschlossen wird, steht ein Block, der mit dem Kanonik-Block zunächst übereinstimmt. Die gleich zu Beginn des Blocks in Abbildung 7.2 auftretende `c:` Zeichenkette markiert hierbei den Sprechaktanfang. Die Einfügungen, Veränderungen und Löschungen von Segmenten gegenüber der Kanonik werden in diesem Block hinzugefügt und mit `-`-Symbol markiert. Die Transkription des Wortes `<Termin> /t -h E6-E m 'i : n/` in diesem Block in Abbildung 7.2 zeigt eine eingefügte Aspiration `-h` und eine Veränderung des Diphthongs (`E6` zu `E`). Der Glottalverschluss in `<Ihnen>` ist als gelöscht markiert `Q-`, während eine Glottalisierung des Vokals hinzugefügt ist `-q`. Autosegmentelle Eigenschaften, wie die Knarrstimme oder die Nasalierung oder beides, sind jeweils auf der Zeitmarke des betreffenden Segments in der Reihenfolge vor dem Segment markiert. Das `%`-Zeichen in `<Ihnen> /Q- -q i: n @- %n+/` markiert eine unsichere Grenze.

Dem Dateiheader folgt die Auflistung der Zeitmarken der durch Freizeichen getrennten Zeichenketten aus der modifizierten Kanonik. Jede Zeitmarke ist in Abtastpunkten angegeben und wird nach einem Freizeichen gefolgt von der Zeichenkette, die dem Zeitbereich zwischen der vorangegangenen und der aktuellen Zeitmarke im Sprachsignal zugeordnet ist. Die Zeitangaben in Sekunden in der dritten Spalte nach dem *hend*-Schlüsselwort in Abbildung 7.2 links sind erst später in die Etikettierungsdateien hinzugefügt worden.

In der Auflistung der Zeitpunkte werden wortinitiale Phoneme mit einem voranstehenden `##` und alle übrigen mit `$` markiert. In diesen Segmentierungen und Etikettierungen werden also die Wortgrenzen explizit durch `##` angegeben. Für Etiketten, die derselben Zeitmarke zuzuordnen sind, werden Zeitmarken mehrfach mit unterschiedlichen Etiketten angegeben. Für Abtastpunkt 34834 sind in Abbildung 7.2 `##:k` sowie `$Q-` als auch `$-q` und `$i` angegeben. Diese Etikettierung weist auf ein mit einem Geräusch überlagertes Wort ohne wortinitialen Glottalverschluss, aber einer Glottalisierung des vorderen geschlossenen ungerundeten Vokals hin.

In der rechten *s1h*-Datei in Abbildung 7.2 sind prosodische Etikettierungen enthalten. Die Etiketten basieren auf dem prosodischen Etikettiersystem PROLAB (Kohler, 1995). Die Markierungen sind wie alle anderen Etikettierungen sowohl im Dateiheader vor dem Schlüsselwort *hend* als auch nach dem Schlüsselwort außerhalb des Dateiheaders enthalten. Für alle prosodischen Etiketten gilt, dass sie mit einem `&`-Zeichen vor dem eigentlichen Symbol markiert sind. So beginnt der Ausschnitt in Abbildung 7.2 rechts nach *hend* mit einem prosodischen Etikett. Dieses Etikett markiert eine Akzentstärke `2` und eine Akzentkontur `^`.

Akzentkonturen werden auf der selben Zeitmarke vor dem Wortanfangsetikett `##` gesetzt und beziehen sich auf den nächsten mit Akzent markierten Vokal. Alle verwendeten Akzentkonturetiketten sind mit ihrer Bedeutung im Anhang B.5 zu entnehmen. Hier im Beispiel trägt `'i :` die Akzentkontur, da es den nächsten Vokal nach dem Akzentkonturenetikett (`2^`) repräsentiert und dieser mit der Hauptakzentmarkierung `'` versehen ist. Innerhalb einer lexikalischen Einheit werden keine prosodischen Etiketten eingestreut.

Phraseninitiale Konturen- sowie Konkatenationskonturenetiketten (Tabelle B.6 und B.7) werden grundsätzlich auf die Zeitmarke von Wortanfängen gesetzt und gehen diesen in der Datei in der angegebenen Reihenfolge voraus. Um welche Art von Kontur es sich handelt (initial, final, Konkatenation) geht aus dem Kontext der Reihenfolge und auch dem Etikett hervor, da nicht alle Konturetiketten für alle Konturen gleichermaßen verfügbar sind. Ein Beispiel ist in der rechten *s1h*-Datei in Abbildung 7.2 Zeile 25 und 26 zu sehen. Zeile 25 enthält ein Etikett für eine Konkatenationskontur, die sich auf die Silben bezieht, die zwischen der letzten Akzentsilbe und diesem Wortende vorkommen, also `t a : k-x d -h e : m+` aus `<Dienstag, dem>`, denn in Zeile 26 gibt es ein Etikett für die nächste Akzentkontur, die die erste Silbe des nächsten Wortes betrifft. Denn die erste Silbe des nächsten Wortes enthält den hauptakzentuierten Vokal `'aI`.

Phrasenfinale Konturen (vgl. Tabelle B.8 und B.9) und das Phrasenetikett (vgl. Tabelle B.3) selbst, welches die Zusammengehörigkeit von Wörtern zu einer Phrase markiert, stehen in dieser Reihenfolge nach dem letzten segmentellen Etikett eines Wortes auf der selben Zeitmarke. Die letzten beiden Zeilen in der *s1h*-Datei zeigen die phrasenfinale Kontur `2.`, die den Intonationsverlauf ab der letzten Akzentsilbe `-t s v ' a I t -Q` somit über `@- n` beschreibt. Das Etikett in der letzten Zeile markiert, dass alle Wörter im Ausschnitt zu einer prosodischen Phrase gehören.

7.2. Methode und Ergebnisse

Für die ersten Konvertierungsversuche wurde nur die nicht prosodisch etikettierte Lesesprache im Kiel Corpus of Read Speech (IPDS, 1994) verwendet. Das Material wurde später um die spontansprachlichen Aufnahmen des Kiel Corpus of Spontaneous Speech Vol I bis III (IPDS, 1995; IPDS, 1996; IPDS, 1997a) sowie unveröffentlichtes Material erweitert. Die prosodischen Etikettierungen wurden erst in einem weiteren Schritt berücksichtigt, da die Lesesprache nicht prosodisch etiket-

tiert ist. Mit dieser Erweiterung wurden auch die noch nicht offiziell im Korpus veröffentlichten Lindenstraßendialoge (Peters, 2006) konvertiert. In jedem Schritt wurde auf Grundlage der Informationen über den Aufbau des Korpus und der Beispieldateien (vgl. Kapitel 7.1) ein geeignetes EMU-Etikettierungsschema diskutiert, entworfen und ggf. erweitert und verändert. Weiterhin wurde das konzeptuelle EMU-Datenbankschema in einem Datenbanktemplate formuliert. Das EMU-Etikettierungsmodell, Etikettierungsschemata und Datenbanktemplates wurden u. a. in Kapitel 3 vorgestellt.

Das gesamte Korpus wurde zunächst auf die lokale Festplatte eines Rechners aufgespielt. Die Verzeichnisstruktur ist auf den verschiedenen CDs und für die Lindenstraßendialoge unterschiedlich. Um sie gemeinsam in eine Datenbank integrieren zu können, wurde die Verzeichnisstruktur so angepasst, dass Äußerungen noch in unterschiedlichen Verzeichnissen vorliegen, aber jede Äußerung in der gleichen Tiefe des Verzeichnisbaumes zu finden ist. Das Anpassen der Verzeichnisstruktur wurde über Tcl/Tk Konsolen-Befehle durchgeführt.

Der erste Schritt, in dem direkt mit der Datenbasis des Kiel Corpus gearbeitet wurde, war die Konvertierung der RAW-Signaldateien in ein für das EMU-System verarbeitbares Format. Formate wurden an dieser Stelle nicht diskutiert, sondern die Entscheidung fiel direkt auf das sehr weit verbreitete WAV-Format (IBM Corp. and Microsoft Corp., 1991; Microsoft, 1994; Microsoft, 2001). Die Konvertierung wurde über eine Stapelverarbeitung mit dem Programm GoldWave (AtlanticSX LLC und Goldwave, Inc., 2007) durchgeführt. Die Signale konnten nun über die Deklaration eines *Track* Objektes mit den Attributen der Dateiextension und dem Verzeichnispfad im Datenbanktemplate in die Datenbank eingepflegt werden. Im ersten nicht leeren Datenbankzustand enthielt die erste Kiel Corpus EMU-Datenbank nur die Sprachsignale.

Die für EMU-Sprachdatenbanken geeignete Verzeichnisstruktur, die Informationen aus der Literatur und das jeweils gegebene Etikettierungsschema boten wiederum die Grundlage für die Implementierung der Konvertierung. Die Implementierung erfolgte in der Programmiersprache Tcl/Tk. Sie ist im `xassp2emu` und `xassp2any` Paket im Quellcode des EMU-Systems (EMU Developers, 2011a) frei zugänglich. Es wurden Tcl Prozeduren geschrieben, die jede einzelne *s1h*-Datei öffnen und den Block vor dem Schlüsselwort *oend* sowie den gesamten Inhalt der Datei nach dem Schlüsselwort *hend* einlesen.

Aus der Orthographie wurden sämtliche Wörter extrahiert und von jeder zusätzlichen Markierung gefiltert. Der letzte *s1h* Block wurde für sämtliche weitere Bearbeitungen verwendet. Nach ersten Tests stellten sich Inkonsistenzen zwischen den verschiedenen Blöcken in der *s1h*-Datei heraus. Daher fiel die Entscheidung, sämtliche Informationen aus dem vierten Block, der eigentlichen Etikettierung, zu interpretieren. Die Orthographie ist in diesem Block nicht enthalten, so dass hierfür auf den Dateiheader zurückgegriffen werden musste.

Aufgrund der bekannten Regeln und Strukturen (vgl. Kapitel 7.1) wurden die Phone, die Kanonik, die prosodischen Etiketten und die Zeiten der Wortgrenzen sowie der Phone extrahiert. Die Zeiten wurden aus den Abtastpunkten und der bekannten Abtastrate der Signale in Sekunden umgerechnet. Über die Zeitinformation wurden die Kanonik und die Phone untereinander, sowie die Kanonik und die Phone den Wörtern zugeordnet. An dieser Stelle traten häufiger Probleme auf, wenn die Transliteration nicht exakt mit den Konventionen für die Transliteration übereinstimmten, beispielsweise bei Jahreszahlen, Zahlwörtern und Abkürzungen bzw. Akronymen. Diese wurden in der Transliteration zum Teil als einzelne Wörter über doppelte Freizeichen markiert, im segmentellen Block jedoch als ein Wort behandelt. Auch der umgekehrte Fall trat auf. Die Prozeduren wurden für diese möglichen Fehler entsprechend angepasst. Aus den Kanonikeinheiten und den Phonen wurden zusätzliche Informationen für die Verwendung als Etiketten in der EMU-Datenbank gefiltert. Die Informationen selbst wurden dabei entsprechend zwischengespeichert, um sie später auf die Ebenen verteilen zu können. Die extrahierten Etiketten wurden gemäß der Tabellen B.3, B.5, B.6, B.7 und B.8 in andere Etiketten übersetzt.

Für die prosodische Etikettierung wurden Prozeduren implementiert, die die Konturen den Arten zuweisen und entsprechend eine zeitliche Ausdehnung unter Berücksichtigung der Phone und deren zusätzlicher Informationen bestimmen. Unter Verwendung der EMU-DML (vgl. Kapitel 3.2.3.2) wurden die aufbereiteten Etikettierungen in die Datenbank eingepflegt.

Nach der Implementierung der Konvertierungsfunktion wurde eine Schnittstelle für die EMU-Applikation labConvert implementiert. Somit konnte die Funktionalität in die Applikation eingebunden werden. Über die bereitgestellte Stapelverarbeitung wurden schließlich die Kiel Corpus *s1h*-Dateien in das EMU-Etikettierungsformat konvertiert. Durch das bereits vorhandene Datenbanktemplate, in dem bereits die Pfade zu den Signalen und Etikettierungsdateien deklariert waren, wurde durch

die Konvertierung der Datenbankzustand der Kiel Corpus EMU-Datenbank mit der Etikettierung vervollständigt.

Neben den in den *s1h*-Etikettierungsdateien vorliegenden Informationen wurde das Kiel Corpus in der nun vorhandenen EMU-Sprachdatenbank um die Silbifizierung auf Phonembasis und um Metadaten über den Sprecher und die Aufnahme über EMU-Datenbankfunktionen und der EMU-*Scripting* Möglichkeit AutoBuild (Harrington, 2010a, Appendix B) erweitert.

Für die Silbifizierung wurde die AutoBuild-Funktion `Syllabify` genutzt. Diese Funktion setzt `S` Segmente auf einer angegebenen Ebene und silbifiziert die Segmente auf einer ebenfalls angegebenen Kind-Ebene nach dem Maximalen Onset Prinzip (Hooper, 1972). Die maximalen Konsonantenverbindungen, die in der Sprache am Anfang einer Silbe vor dem Silbenkernvokal stehen dürfen, werden im AutoBuild-Skript definiert, die möglichen Silbenkerne im Datenbanktemplate als *legal label* Klasse `vowel`. Im Rahmen der Kiel Corpus Aufbereitung wurde die Funktion um das Berücksichtigen bereits vorhandener Wortgrenzen (vorhandene Assoziationen von Segmenten auf der Kind-Ebene der Silben-Ebene mit Segmenten einer dominierenden Ebene) erweitert. Die Verwendung von AutoBuild Skripten und die Verwendung der AutoBuild-Funktion `Syllabify` ist in Harrington (2010a, S. B-16) für die deutsche Sprache und der Verwendung der SAMPA in der Etikettierung, wie in der Kiel Corpus EMU-Datenbank, beschrieben. Daher wird an dieser Stelle auf eine weitere Beschreibung verzichtet.

In Kapitel 5.3.2 wurde die AutoBuild Funktion `AddLabelsFromFile`, die in dieser Arbeit entstanden ist, erklärt. Diese Funktion wurde verwendet, um die Sprecherinformationen in die Datenbank einzupflegen. Hierfür wird für jede Äußerung der Datenbank eine Textdatei mit den hinzuzufügenden Informationen erwartet. Dafür wurde die Information über die Sprecher aus dem Archiv des IPDS herausgesucht und in Tabellen digitalisiert. Aus diesen Tabellen wurde für jede Äußerung skriptbasiert jeweils eine Informationsdatei erzeugt. Beide AutoBuild Skripte wurden auf alle Äußerungen der Kiel Corpus Datenbank mithilfe der Stapelverarbeitung der EMU-Applikation AutoBuild Wizard ausgeführt. AutoBuild Wizard ist eine grafische Oberfläche, die es ermöglicht, AutoBuild Skripte auf die gesamte Datenbank oder nur auf Teile der Datenbank anzuwenden (vgl. Kapitel 5.3.2).

Aufgrund der Konvertierung der prosodischen Etikettierung wurden, nach weiblichen und männlichen Sprechern getrennt, Grundfrequenzdaten mit der EMU-Applikation tkassp aus den Sprachsignalen analysiert und der Datenbank hinzugefügt, indem

dem Datenbanktemplate eine weitere *Track*-Deklaration für die Grundfrequenzdaten hinzugefügt wurde. Die EMU-Applikation tkassp ist eine grafische Benutzeroberfläche zu den Signalanalyseprogrammen in `libassp` (vgl. Kapitel 3, S.53). Die Trennung der Sprecher nach Geschlecht wurde über EMU-Abfragen erreicht. Aus der Datenbank wurden alle Äußerungen weiblicher und nicht weiblicher Sprecher über die Abfrage ermittelt. Die dabei entstandenen Äußerungslisten konnten über eine Schnittstelle des EMU-Systems zum tkassp Signalanalyseprogramm weitergeleitet werden. Hier wurden nach Geschlechtern getrennt Parameter angepasst und schließlich die Grundfrequenzdaten durch die F0ana Analyse berechnet. Die aus der Berechnung resultierenden Dateien im SSF-Format (vgl. Kapitel 3.2.2.2) werden durch tkassp im selben Verzeichnis gespeichert wie die für die Berechnung als Grundlage dienenden Sprachsignale.

Die Überführung des Kiel Corpus in eine EMU-Datenbank ist nur mit einem EMU-Datenbankschema (Datenbanktemplate) und einem Konvertierungsprogramm zu bewerkstelligen gewesen. Neben dem erzeugten Kiel Corpus EMU-Datenbankzustand sind auch Template und Programm als Ergebnis dieser Arbeit anzusehen. Die Kiel Corpus EMU-Datenbank liegt unveröffentlicht im Archiv des Instituts für Phonetik und Sprachverarbeitung an der Ludwig-Maximilians-Universität in München auf CDs vor. Das Datenbanktemplate, das das Etikettierungsschema enthält, ist auf den CDs enthalten, damit der Datenbankzustand im EMU-System verarbeitet werden kann. Etikettierungsschema, Datenbanktemplate und das Konvertierungsprogramm werden im Folgenden näher erläutert.

7.2.1. EMU-Etikettierungsschema und Ausprägung

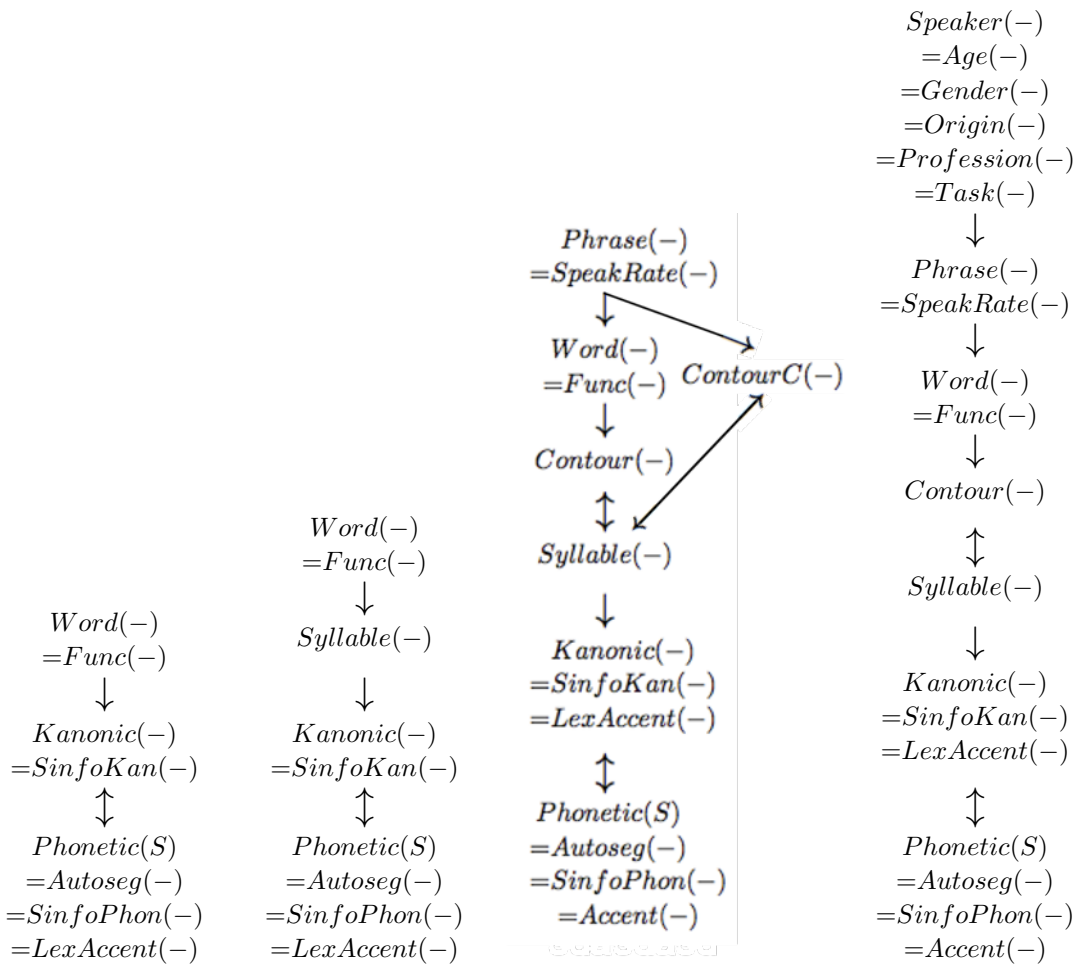
Wie aus Kapitel 7.1.1 hervorgeht, ist die Etikettierung im Kiel Corpus sehr umfangreich und deckt sehr verschiedene linguistische und phonetische Ebenen ab. Für die Modellierung des EMU-Etikettierungsschemas wurden, soweit möglich, alle Arten der Etikettierungen im Kiel Corpus diskutiert und ggf. in das Etikettierungsschema in Form von Ebenen eingefügt. In der Modellierung wurden auch die möglichen Abfragen berücksichtigt. Da Kiel Corpus Analysen in der Fachliteratur vorgestellt werden, sind die Ebenennamen in englischer Sprache gewählt worden. Die während des Aufbaus der Kiel Corpus EMU-Datenbank entworfenen EMU-Etikettierungsschemata sind in Abbildung 7.3 in den Schemata 12–15 abgebildet. Neben den Ebenen, die das Etikettierungsmodell vorgibt, wurden für die Datenbankausprägung die möglichen Zeichenmengen festgelegt und Kiel Corpus Etiketten durch intuitivere

Zeichenketten ersetzt. Abbildung 7.4 (S. 204) zeigt ein Beispiel einer konvertierten Äußerung des Kiel Corpus ohne prosodische Etikettierungen. Das Etikettierungsschema entspricht Schema 13 in Abbildung 7.3

Durch die Transliteration des Sprachsignals in Wörter ist eine `Word`-Ebene ohne Diskussion bereitzustellen gewesen. Im Kiel Corpus werden Funktionswörter von Inhaltswörtern unterschieden. Für das Etikettierungsschema wurde diese Tatsache als Eigenschaft der Wörter interpretiert und somit auf einem Label-Link zur `Word`-Ebene mit einer `Func`-Ebene modelliert (vgl. Schemata 12–15 in Abbildung 7.3). In der Datenbankausprägung werden die Wörter der Äußerung auf der `Word`-Ebene orthographisch abgebildet. Jedes Segment entspricht hierbei einem Wort. Die Orthographie stammt aus dem ersten Teil der *s1h*-Datei (siehe Abbildung 7.2) oberhalb des Schlüsselwortes *oend*. In der Orthographie wurden aufgrund der Zeichenkodierungsprobleme zu damaliger Zeit (2003) die deutschen Umlaute umgeschrieben. Die Interpunktion wurde nicht übernommen sondern entfällt gänzlich. Auf dieser Ebene werden auch die Häsitationspartikel abgebildet. Wenn in der *s1h*-Datei die Häsitation ausgeschrieben wurde, wurde diese Orthographie übernommen. Sonst findet man `hesitat` oder `haes` auf der `Word`-Ebene. Funktionswörter werden auf `Func`-Ebene mit dem Etikett `F` markiert, wie das Wort `<noch>` in Abbildung 7.4.

Die Wörter aus der orthographischen Transliteration wurden im Kiel Corpus in kanonische Formen übersetzt, so dass sie eine explizite Assoziation zu den Wörtern haben müssen. Diese Assoziation wurde im Schema über eine Dominanzbeziehung zwischen der `Word`-Ebene und einer `Kanonic`-Ebene umgesetzt. Da die Wörter aus den Phonemen der Kanonik bestehen, gilt die `Word`-Ebene als die Eltern-Ebene, während die `Kanonik`-Ebene die direkte Kind-Ebene darstellt. Als Eltern-Kind Beziehung wurde, wie aus den Schemata 12–15 in Abbildung 7.3 hervorgeht, die einfache one-to-many Beziehung gewählt, da auf Phonemebene jedes Phonem genau einem Wort zugeordnet werden kann und einem Wort mehrere Phoneme. `Kanonic` ist im Vergleich zu anderen Korpora eine schlechte Namensgebung, aber aufgrund der Ausnahmen der nicht phonematischen Transkriptionen für die Phone [ç, x, χ] und [ʔ] wurde auf die explizite Benennung der Ebene als `Phoneme`-Ebene verzichtet.

Es ist festzustellen, dass das Kiel Corpus eine segmentelle phonetisch/phonologische Etikettierung aufweist, die mit dem Zeitsignal aligniert ist. Aus Kapitel 3.1.1 geht hervor, dass auch das EMU-System Etikettierungsebenen anbietet, die für die segmentelle Etikettierung genutzt werden können. EMU unterscheidet intervall- und eventzeitgebundene Ebenen. Da die Etikettierung im Kiel Corpus nur Intervalle



Schema 12. **Schema 13.** **Schema 14.** **Schema 15.**

Abbildung 7.3.: Etikettierungsschemata in der Entwicklung der Kiel Corpus Aufbereitung für das EMU-System mit Schemata 12 und 13 für die Konvertierung des Kiel Corpus of Read Speech und den Erweiterungen in Schema 14 für den Kiel Corpus of Spontaneous Speech und unveröffentlichte Teile mit prosodischer Etikettierung, sowie Schema 15, das Schema 14 um eine Ebene erweitert, sonst jedoch identisch ist – die zweite Kette aus Schema 14 wurde in der Darstellung nicht berücksichtigt, ist aber ebenfalls Teil von Schema 15.

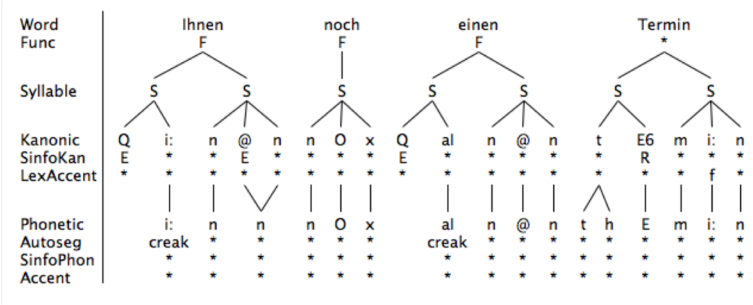


Abbildung 7.4.: Ausschnitt der Kiel Corpus Äußerung g365a007 in der aufbereiteten hierarchischen EMU-Etikettierungsstruktur ohne prosodische Etikettierungen und Metadaten.

enthält, wurde sich im EMU-Etikettierungsschema für eine intervall-zeitgebundene Ebene mit dem Ebenennamen `Phonetic` entschieden. Sie stellt die alleinige zeitgebundene Ebene im Schema dar, da alle anderen Ebenen mit der `Phonetic`-Ebene direkt oder indirekt assoziiert sind. Somit können die Zeiten für jedes Segment auf anderen Ebenen aus den Segmenten auf der `Phonetic`-Ebene abgeleitet werden. Über diese Modellierung ist gewährleistet, dass alle Zeitinformationen, die in den Kiel Corpus Etikettierungsdateien enthalten sind, in der Datenbank ebenfalls enthalten sind und somit abgefragt werden können. Die Entscheidung der Namensgebung der `Phonetic`-Ebene ist über linguistische und phonetische Theorien zu erklären und analog zu bereits vorhandenen EMU-Datenbanken. In den meisten Datenbanken sind die zeitalignierten Phone mit einer Ebene dieses Namens assoziiert. Dass es sich in der zeitalignierten Etikettierung um Phone handeln muss, ist aus der Theorie insofern erklärbar, da es sprachliche Einheiten sind, die eine physische Realität haben. In der klassischen Linguistik nennt man diese sprachlichen Einheiten Phone. Zudem sind diese Einheiten im Kiel Corpus mit Symbolen der IPA[2] etikettiert.

Sowohl die Datenbankausprägung für die `Kanonic`-Ebene als auch für die `Phonetic`-Ebene wurde aus den Etiketten in der *s1h*-Datei (unterhalb Schlüsselwortes *hend*) interpretiert. Für die `Kanonic` wurden die Etiketten verwendet, die auch oberhalb des *kend* Schlüsselwortes in der *s1h*-Datei notiert sind. Für die `Phonetic` entsprechen die Etiketten denen der Kanonik, außer für Segmente, die gegenüber der Kanonik als verändert markiert sind. In diesen Fällen entspricht das Etikett auf der `Phonetic`-Ebene dem interpretierten Ergebnis der markierten Veränderung. Im Wort `<Termin>` ist der Vokal `E6` in der *s1h*-Datei mit dem Etikett `E6-E` transkribiert und stellt damit

[2] aber den Entsprechungen in SAMPA-Notation

die Ersetzung des Diphthongs mit dem Monophthong dar. Das Etikett `E6` ist für die EMU-Datenbank eine Ausprägung[3] von `Kanonic`, während `E` eine Ausprägung von `Phonetic` ist. Gegenüber `Kanonic` enthält die `Phonetic`-Ebene Segmente für nicht verbale Ereignisse, die nicht mit den Phonen überlappen. Diese Ereignisse sind im Kiel Corpus mit Zeitmarken und Etikett notiert. Während die Zeitmarke übernommen wurde, sind die Etiketten mit intuitiveren Etiketten ersetzt worden. Im Kiel Corpus wird z. B. für Geräusche das Etikett `:k` verwendet, in der EMU-Datenbank ist dieses als `ASext.noise` markiert. Tabelle B.2 listet alle möglichen Kiel Corpus Etiketten mit den Entsprechungen in der EMU-Datenbank zusammen mit der Bedeutung der Etiketten auf.

Phone sind im Kiel Corpus aus der Kanonik abgeleitet, so dass auch hier eine direkte Beziehung besteht, die wiederum durch eine Dominanzbeziehung der `Kanonic`-Ebene zur `Phonetic`-Ebene im Schema (vgl. Schemata 12–15) realisiert wurde. Die Eltern-Kind Assoziation wurde als many-to-many modelliert, da aus theoretischer Sicht durch Elisionen oder Assimilationen auf Phonebene ein einzelnes Phon mit mehreren Phonemen der Kanonik assoziiert sein kann. Diese phonologischen Prozesse sind indirekt über das Markieren von Phonemlöschungen, -veränderungen und -hinzufügungen im Kiel Corpus enthalten. Für diese Informationen wurden sowohl der `Kanonic`- als auch der `Phonetic`-Ebene Informationsebenen als Label-Link modelliert. Beide Informationsebenen sind nötig, da Phoneinfügungen nicht in der Kanonik berücksichtigt werden können und Phonemtilgungen nicht in der Phonetik, da auf den entsprechenden Ebenen die jeweiligen Segmente nicht vorhanden sind. Auf dem `SinfoKan` Label-Link zur `Kanonic`-Ebene werden getilgte Segmente (Löschungen) mit dem Etikett `E` $\widehat{=}$ Elision markiert, wie in Abbildung 7.4 für den Glottalverschluss `Q` und das Schwa `@` im Wort `<Ihnen>`. Assimilationen oder Ersetzungen sind mit `R` $\widehat{=}$ Replaced versehen (vgl. `E6` im Wort `<Termin>`), während eingefügte Segmente mit dem Etikett `Ep` $\widehat{=}$ Epenthese auf dem `Phonetic` Label-Link `SinfoPhon` markiert werden.

In der Aufbereitung des Kiel Corpus als EMU-Datenbank wurden Konventionen für die Datenbankausprägung auf diesen Ebenen aufgestellt und umgesetzt, die nicht direkt in den Etikettierungsdateien enthalten sind. So wird Aspiration `-h` auf der `Phonetic`-Ebene als Segment eingefügt, aber nicht als gegenüber der Kanonik eingefügt markiert (kein `Ep`-Etikett auf der `SinfoPhon`-Ebene). Stattdessen ist in der

[3] Ausprägung wird hier im Rahmen eines konzeptionellen Schemas verwendet, in dem Segmente auf den Ebenen als Ausprägungen des Objekttyps angesehen werden, obwohl sich diese Sichtweise im logischen Datenbankschema ändern kann.

Datenbankausprägung das Plosivsegment in der Kanonik mit dem Plosivsegment und der Aspiration assoziiert (vgl. `t` in `Termin` in Abbildung 7.4). Weiterhin werden bekannte phonologische Prozesse des Deutschen bereits berücksichtigt. Wird auf der `Kanonic`-Ebene eine Folge /ən/ abgebildet, wobei eine Schwa-Tilgung über eine Markierung auf der `SinfoKan`-Ebene markiert ist, werden die Segmente mit den Etiketten `@` und `n` dem Nasalsymbol auf der `Phonetic`-Ebene hierarchisch zugeordnet. Dies begründet sich in der Annahme, dass die Folge /ən/ ein Morphem abbildet und das Allomorph zum Nasal reduziert wird.

Nach den Informationen in Kapitel 7.1.1 sind autosegmentelle Ereignisse, sowie der lexikalische Akzent eines jeden Wortes direkt am Phon markiert. Beide Informationen konnten im Schema somit als Eigenschaft einzelner Phone angesehen werden. Daher wurden hierfür die Ebenen `Autoseg` und `LexAccent` als Label-Links der `Phonetic`-Ebene dem Schema (vgl. Schemata 12 und 13 in Abbildung 7.3) hinzugefügt. Autosegmentelles wie Nasalierung wird mit `nas`, Knarrstimme mit `creak` oder beides mit `nascreak` auf dem `Autoseg` Label-Link zur `Phonetic`-Ebene in der Datenbankausprägung angegeben. In den Wörtern `<Ihnen>` und `<einen>` in Abbildung 7.4 sind Vokale mit Knarrstimme markiert.

Auf der `LexAccent`-Ebene gibt es in der Datenbankausprägung als Markierung das Etikett `f`, wenn der Vokal, der diese Markierung trägt, den ersten lexikalischen Akzent hat, wie in Abbildung 7.4 das Wort `<Termin>` und `s` wenn es der sekundäre Akzent der Silbe vom nicht initialen Element eines Kompositums ist. Ein Beispiel hierfür ist in Abbildung A.3 im Anhang im Wort `<Wochenende>` zu finden. Funktionswörter erhalten genauso wie im dateibasierten Kiel Corpus keine solche Markierung.

Die `Syllable`-Ebene in den Schemata 12–15 wurde ohne Referenz zu den Kiel Corpus Etikettierungsdateien hinzugefügt und enthält ausschließlich `S` Segmente, die die Phoneme auf `Kanonic`-Ebene dominieren und selbst durch die `Word`-Segmente auf der `Word`-Ebene dominiert werden. Das Einfügen der Silben in der Hierarchie zwischen Wort und Kanonik ist linguistischen Theorien zur Folge die einzige Lösung, da Wörter aus Silben bestehen und ein oder mehrere Phoneme bzw. Phone aufgrund der Sonoritätshierarchie oder anderen Theorien wie dem Maximalen Onset Prinzip Silben bilden (Hooper, 1972).

Mit dem Hinzufügen der prosodischen Etikettierungen in der Konvertierung (vgl. Abbildung 7.5) zur Aufbereitung des Kiel Corpus als EMU-Datenbank kamen Wortakzente mit Akzentstufen für das Vokalphon hinzu. Die Akzentstufe wurde als Rea-

lisierung des kanonisch vorgesehenen lexikalischen Akzents für die Modellierung des EMU-Etikettierungsschema interpretiert. Aus diesem Grund wurde die `LexAccent`-Ebene der `Kanonic` als Label-Link zugeordnet, während die `Accent`-Ebene den Label-Link der `Phonetic`-Ebene für die Akzentstufen ersetzt. Die Datenbankschemata in 12 und 13 wurden somit durch das Schema in 14 abgelöst. In der Datenbankausprägung finden sich auf dieser Ebene die Etiketten `0`, `1`, `2` und `3` äquivalent zu den Kiel Corpus Etikettierungen in den *s1h*-Dateien. Die Bedeutung der Etiketten ist Tabelle B.4 zu entnehmen.

Phrasen, Konturen und Akzente sind explizit im Kieler Intonationsmodell als Objekte vorhanden und wurden daher als Ebenen `Phrase`, `Contour` und `Accent` im Schema modelliert (vgl. Schema 14 in Abbildung 7.3). In der Datenbankausprägung in Abbildung 7.5 sind keinerlei Etiketten für die prosodische Etikettierung zu sehen, die mit den Etiketten in der *s1h*-Datei übereinstimmen. Grund dafür ist die Übersetzung der Zeichenketten in leichter verarbeitbare, d. h. abfragbare und intuitivere Zeichenketten. Im Kiel Corpus werden z. B. in den Etiketten der Akzentkonturen eine Vielzahl an Zeichen aus dem EQL-Alphabet verwendet (vgl. Tabelle B.5 im Anhang und Kapitel 3.3.1). Abfrageausdrücke in EQL werden damit unübersichtlich. Zudem müssen derartige Zeichen als Etikett in der einfachen Abfrage als Sonderzeichen behandelt werden, was einen Mehraufwand in der Formulierung mit sich bringt, aber auch zur weiteren Unübersichtlichkeit führt. Die Übersetzungen und Bedeutungen der im Kiel Corpus verwendeten Etiketten sind in den Tabellen B.3 für die `Phrase`-Ebene und in den Tabellen B.5, B.6, B.7 sowie B.8 getrennt nach Konturentypen für die `Contour`- bzw. `ContourC`-Ebene aufgelistet.

Da das gesamte Sprachmaterial im Kiel Corpus mit prosodischer Etikettierung in Phrasen eingeteilt ist, wurde die `Phrase`-Ebene als Eltern-Ebene aller bisher modellierten Ebenen in das Schema eingefügt. Für `Contour` stellt sich die Sache schwieriger dar. Im Schema 14 sind die Konturen in der Hauptkette mit den Silben assoziiert. Diese Assoziation beruht auf der Information, dass alle Konturetiketten im Kiel Corpus auf Silben bezogen sind und das wiederum in Bezug auf die Silben mit hauptakzentuiertem Vokal. Die many-to-many Eltern-Kind Beziehung bewirkt, dass theoriekonform ein diskretisierter Intonationsverlauf auf einer Silbe enden kann und ein weiterer anfängt, sie sich jedoch nicht überlappen. Weiterhin sind die Konturen laut Schema von Wörtern dominiert. Dieser Umstand ist nicht theoriekonform, denn Intonationsverläufe bewegen sich über Wortgrenzen hinweg und sind von der Einheit ‚orthographisches Wort' weitgehend unabhängig. In der Datenbankausprägung wird dieses Problem mit der Wiederholung des selben Konturetiketts in angrenzenden

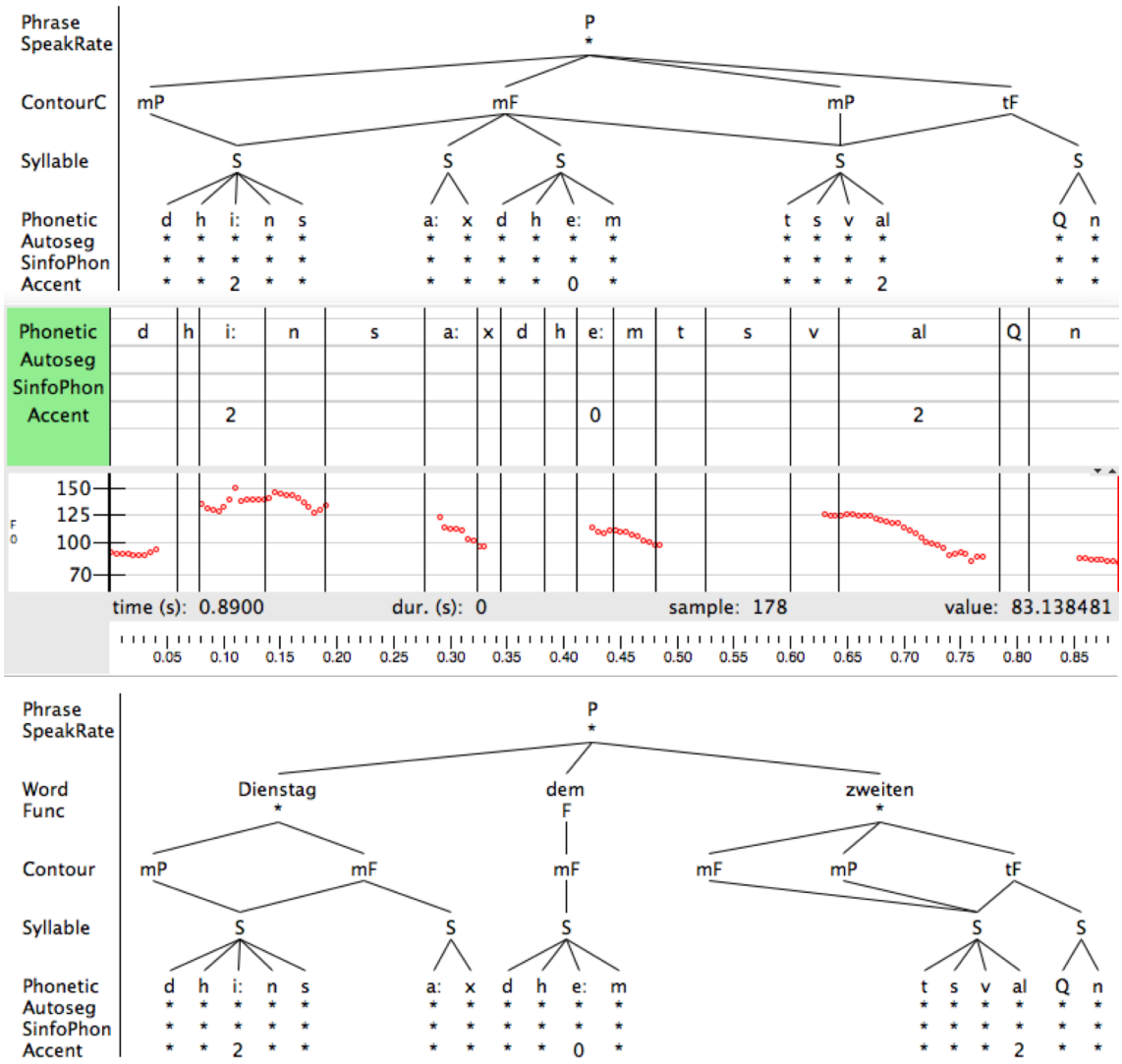
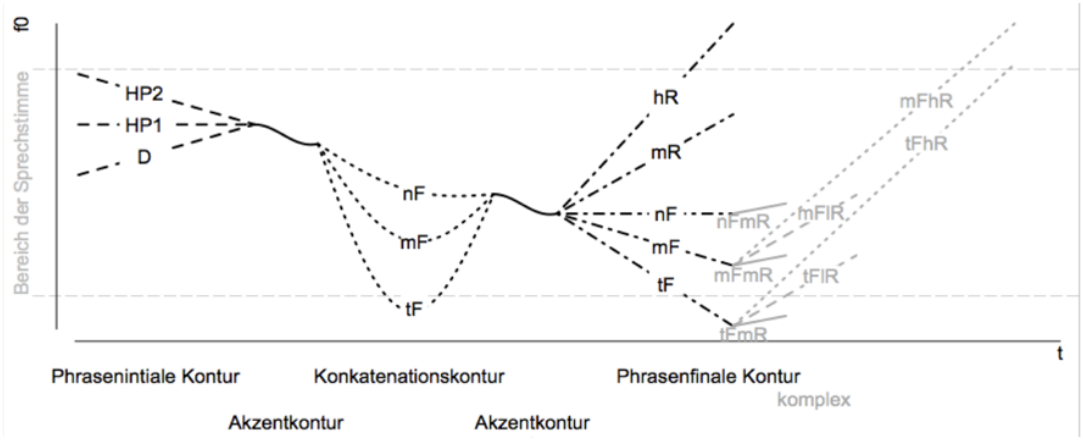


Abbildung 7.5.: Ausschnitt der Kiel Corpus Äußerung g071a003 mit prosodischer Etikettierung in der hierarchischen Etikettierung mit ConturC und ohne Wort-Ebene (oben), Grundfrequenzsignal mit alignierten Phonen der `Phonetic`-Ebene (mittig) und in der hierarchischen Etikettierung mit Contour und Wort-Ebene (unten).

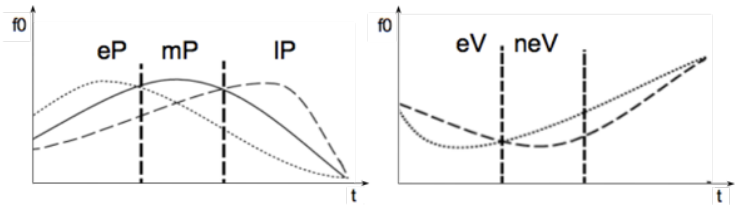
Wörtern gelindert. Eine theoriekonformere Modellierung wäre, `Word` aus der Hierarchie der Ebenen `Phrase` → `Word` → `Contour` → `Silbe` komplett herauszulassen, so dass die Hierarchie `Phrase` → `Contour` → `Silbe` besteht. Dieser Ansatz wäre als Sicht (vgl. S. 29) auf die Daten denkbar, ohne das Schema ändern zu müssen. Aufgrund des Lösungsansatzes, für die Datenbankausprägung Etiketten zu wiederholen, kann eine derartige Sicht jedoch nicht zur Verfügung gestellt werden. Auf die theoriekonforme Modellierung sollte allerdings nicht verzichtet werden. Es wurde eine zweite Kette entworfen, die über eine `ContourC`-Ebene die `Word`-Ebene überspringt (vgl. Schema 14 in Abbildung 7.3). Auf diese Weise sind beide genannten Alternativen im Etikettierungsschema enthalten. Die zweite Kette hätte die Modellierung der prosodischen Etikettierungen vollkommen abgedeckt. Für die Darstellung der Etikettierungsbäume, die als Sicht für die Nutzer aufgefasst werden kann, ist eine solche Modellierung aber ungeeignet. Denn Phrasen, Akzente und Wörter würden gleichzeitig dargestellt, wobei die Konturen fehlten oder es würden mit der zweiten Kette Phrasen, Konturen und Akzente, diesmal jedoch ohne die Wörter, dargestellt werden, die jedoch die Lesbarkeit von Etikettierungen verbessern (Schultze-Berndt, 2006). Beide Ketten sind zusammen mit der Datenbankausprägung in Abbildung 7.5 zum Vergleich dargestellt.

Die Datenbankausprägung der Ebenen in Bezug auf die Silben und die Phrasen lässt sich unter Berücksichtigung von Abbildung 7.6, einer Umarbeitung der Übersichtsdiagramme aus Peters und Kohler (2004, S. 6, 14) nachvollziehen. Wie aus dieser Abbildung und Kapitel 7.1.1 hervorgeht, werden Konturen innerhalb einer Phrase sequentiell nebeneinander etikettiert. Dabei sind sie jeweils mit Silben assoziiert. Die Phrase in Abbildung 7.6 (S. 210) besteht aus vier Konturen, der Akzentkontur `mP`, der Konkatenationskontur `mF` gefolgt von einer weiteren Akzentkontur `mP` und schließlich der phrasenfinalen Kontur `tF`. Die beiden Akzentkonturen sind dabei ausschließlich mit der Silbe assoziiert, die die Phoneme und die Phone der akzentuierten Silbe dominiert, während die Konkatenationskontur die dazwischenliegenden Silben, aber auch die beiden akzentuierten Silben dominiert. Gleiches gilt für die phrasenfinale Kontur.

Die Zusammenhänge der Konkatenationskonturen sowie der finalen Konturen mit der Akzentsilbe wurden in der Konvertierung so umgesetzt, da weder aus dem Kieler Intonationsmodell noch aus den Etikettierungen in den *s1h*-Dateien hervorgeht, wann eine Kontur explizit, in Zeiteinheiten ausgedrückt, beginnt und endet. Nur Akzentkonturen beziehen sich laut Modell auf eine spezielle Silbe. Aus Abbildung 7.6 geht jedoch hervor, dass die Konturen nahtlos ineinander übergehen. Diese Fakten



(a)



(b)

Abbildung 7.6.: Konturen im Kieler Intonationsmodell nach Peters und Kohler (2004, S. 6, 14) mit denen in der Konvertierung übersetzten Etiketten mit den möglichen Akzentkonturen in (b), wobei die vertikalen Striche den akzentuierten Vokal kennzeichnen, weiterhin mit phraseninitialen Konturen, Konkatenationskonturen und phrasenfinalen Konturen mit Platzhalter für die Akzentkonturen in (a). Grau markierte Konturen sind als komplexe Konturen aus einfachen zusammengesetzt.

sollen durch die Umsetzung in der Datenbankausprägung (Assoziation der Akzentkonturen mit nur einer Silbe und die Assoziation der anderen Konturen mit mehreren Silben inklusive der Akzentsilbe) zum Ausdruck gebracht werden. Unter Berücksichtigung der Bedeutung der Etiketten (vgl. Tabellen B.5, B.6, B.7, B.8) lässt sich die Intonationskontur in der Phrase, die in Abbildung 7.5 dargestellt ist, durch einen mittleren Grundfrequenzgipfel `mP` mit anschließendem Abfall der Grundfrequenz `mF`, einem weiteren mittleren Gipfel und anschließender starker Verringerung der Grundfrequenz `tF` beschreiben. Im Vergleich zum dargestellten Grundfrequenzverlauf (F0) in Abbildung 7.5 ist diese Beschreibung nachvollziehbar. Der F0-Verlauf zeigt jedoch, dass die beiden Grundfrequenzabsenkungen (`mF` und `tF`) bereits in den jeweils akzentuierten Silben beginnen, bzw. nicht von den Gipfelkonturen, wie sie in Abbildung 7.6 schematisch dargestellt sind, explizit trennbar sind. Diese Beobachtung

zeigt, dass die Umsetzung in der Datenbankausprägung der Etiketten nicht von der Hand zu weisen ist. Ein Beispiel der Etikettierung einer phraseninitialen Kontur ist in Abbildung A.3 im Anhang als erstes Phrasenetikett zu sehen.

Für das Hinzufügen der Metadaten über Sprecher und Aufnahme wurde die gesamte Hierarchie durch eine alles dominierende `Speaker`-Ebene mit Label-Links ein letztes Mal bis heute erweitert (vgl. Schema 15 in Abbildung 7.3). Die Datenbankausprägungen sind den Aufnahmeprotokollen entnommen, wobei für die Lindenstraßendialoge sämtliche Metadaten aus Peters (2006) entnommen werden konnten. Im Teil der Datenbank, die den Kiel Corpus of Read Speech enthält, gibt es keine genauen Angaben zum Alter, so dass nur Alterskategorien in der Datenbankausprägung zu finden sind.

Neben den bereits erwähnten Etikettierungsarten werden nach Kapitel 7.1.1 nicht verbale Ereignisse ebenfalls im Kiel Corpus notiert. Diese Ereignisse stellen zeitliche Intervalle dar, die mit den Phonen überlappen und demnach als autosegmentell betrachtet werden können. Eine Modellierung im Schema über eine weitere zeitgebundene Ebene in autosegmenteller Beziehung zur `Phonetic`-Ebene wäre denkbar. Bisher wurde jedoch darauf verzichtet.

Unsicherheitsmarkierungen, die im Kiel Corpus enthalten sind, wurden nur für die `Phrase`-Ebene modelliert (vgl. Abbildung A.3). Eine `uncertain`-Ebene wäre als Label-Link realisierbar. Jedoch müssten alle Ebenen im Schema einen solchen Label-Link mit sich führen, da auf jeder Ebene Unsicherheitsmarkierungen auftauchen können. Alternativ könnte auf einer einzelnen Ebene, die wiederum als Label-Link der `Phonetic`-Ebene modelliert wäre, für jedes Segment eine Unsicherheitsmarkierung erfolgen, die von einem unsicheren Wort oder Kanonik dominiert sind. Weiterhin wurde die Überlegung angestellt, eine Unsicherheitsmarkierung direkt in jedem Etikett zu behalten, wie es im Kiel Corpus der Fall ist. Davon wurde abgesehen, da diese Phone etc. nicht gleichermaßen abfragbar wären wie andere. Für die `Phrase`-Etiketten wurde anders entschieden, da die Richtigkeit der Entscheidung des Etikettes sehr wichtig ist, denn es ist im Prinzip die alleinige Information, die das Etikett ausdrückt, nämlich den Typ der Phrase vor, zwischen oder nach anderen Phrasen. Unter Verwendung regulärer Ausdrücke wären Abfragen formulierbar, die unsichere Segmente enthalten dürfen. Die EMU Query Language (vgl. Kapitel 3.3) unterstützt jedoch reguläre Ausdrücke nicht. Außerdem müsste jede Abfrage über diese Modellierung die Unsicherheitsmarkierung, insofern irrelevant für das Ergebnis, in allen einfachen Abfragen im Abfrageausdruck formuliert sein. Keine der diskutierten Mo-

dellierungen wurden im Bezug auf die geringe Häufigkeit als brauchbar eingestuft. Somit wurde auf diese Information in der Konvertierung gänzlich verzichtet.

7.2.2. Datenbanktemplate

Für die Kiel Corpus Aufbereitung wurden schließlich zwei Datenbanktemplatedateien erstellt. Zum einen liegt ein Datenbanktemplate zusammen mit den Etikettierungen und Signalen des Kiel Corpus in einem Verzeichnis gespeichert vor. Zum anderen ist im EMU-System ein Datenbanktemplatemuster bereitgestellt, das von EMU-Nutzern, die Teile des Kiel Corpus selbst konvertieren wollen, genutzt werden kann. Die Unterschiede in beiden liegen jedoch nur in den Pfadangaben. Das Datenbanktemplate, das zusammen mit der Datenbank vorliegt, ist in Abbildung 7.7 dargestellt.

```
1  ! this file was generated by tpled
2
3  level Speaker
4  level Phrase Speaker
5  level Word Phrase
6  level Contour Word
7  level Syllable Contour many-to-many
8  level Kanonic Syllable
9  level Phonetic Kanonic many-to-many
10 level ContourC Phrase
11 level Syllable ContourC many-to-many
12
13 label Speaker Age
14 label Speaker Gender
15 label Speaker Origin
16 label Speaker Profession
17 label Speaker Task
18
19 label Phrase SpeakRate
20
21 label Word Func
22
23 label Kanonic SinfoKan
24 label Kanonic LexAccent
25
26 label Phonetic Autoseg
27 label Phonetic SinfoPhon
28 label Phonetic Accent
29
30 labfile Phonetic :type SEGMENT :extension ph :time-factor 1000
31
32 legal Phonetic vowel 2: 2:6 a: a:6 e: E: E:6 e:6 i: i:6 o: o:6 u:
33           u:6 y: y:6 aI aU OY U 9 a a6 E E6 I I6 O O6 U6 Y Y6 6 @
34
35 path ph ./KIMlabs/*/*
36 path hlb ./KIMlabs/*/*
37 path wav ./sigslabs/*/*
38 path sf0 ./sigslabs/*/*
39
40 track samples wav
41 track F0 sf0
42
43 set PrimaryExtension hlb
44 set HierarchyViewLevels Speaker Phrase Word Contour Syllable Kanonic Phonetic
```

Abbildung 7.7.: Datenbanktemplate der Kiel Corpus EMU-Datenbank.

Die Datenbanktemplatedatei deklariert zunächst in den Zeilen 2–28 die Ebenen aus dem Schema 15 und die `ContourC`-Ebene aus Schema 14 in Abbildung 7.3. Die erste

Kette ist hierfür in den Zeilen 2-8 deklariert, die zweite in 10 und 11. Die Intervall-Zeitgebundenheit der `Phonetic`-Ebene ist in Zeile 30 deklariert. Als Dateiextension wurde für die externen Etikettierungsdateien *ph* gewählt. Das für das skriptbasierte Einfügen der Silben benötigte Zeicheninventar wird als *legal label* Klasse `vowel` in den Zeilen 32–34 realisiert. Die darauf folgenden Pfaddeklarationen für die externen Etikettierungsdateien, der *hlb*-Dateien und für die Signale sind als relative Pfade angegeben, so dass die Datenbank bei einer Weitergabe auf jedem Rechner mithilfe der im Rahmen dieser Arbeit entstandenen EMU-Applikation autoDBinstaller installiert werden kann. Da im Aufbau des Kiel Corpus als EMU-Datenbank Grundfrequenzdaten analysiert wurden, sind diese zusammen mit den Sprachsignalen, die als *wav*-Dateien in der aufbereiteten Datenbank vorliegen, als *Tracks* in den Zeilen 40 und 41 deklariert.

Den genannten Deklarationen folgt das Setzen verschiedener Variablen. Die *PrimaryExtension*, die ausschlaggebend für die Äußerungsinstanzen ist, wurde so gesetzt, dass nur das Vorkommen der *hlb*-Dateien die Instanzen der Äußerungen bestimmt. Diese Entscheidung beruht auf der Tatsache, dass aufgrund von Inkonsistenzen nicht alle Kiel Corpus *s1h*-Dateien konvertiert werden konnten. Die Signale sind jedoch in der Datenbank enthalten. In Abfragen über alle Äußerungen würde das Fehlen einer *hlb*-Datei zu einer vorhandenen Äußerung zu Fehlern bzw. Analysefehlern gerade in der Statistik über das Vorkommen von Phänomenen in der Datenbank führen. Die Variable *HierarchyViewLevels* setzt eine Sicht auf die Daten, so dass alle in den Etikettierungen enthaltenen Informationen innerhalb einer Baumstruktur der Etikettierung dargestellt werden.

7.2.3. Konvertierungsprogramm

Das Konvertierungsprogramm besteht aus zwei Tcl/Tk Paketen `xassp2emu` und `xassp2any`, die in der labConvert Applikation in das EMU-System eingebunden sind, so dass die Applikation eine grafische Benutzerschnittstelle zum Tcl/Tk Paket zur Verfügung stellen kann. Das Paket selbst enthält die Tcl-Implementierung der Routinen zum Öffnen und Einlesen der *s1h*-Datei, zum Selektieren der Etiketten für die verschiedenen Ebenen im EMU-Etikettierungsschema, das Übersetzen der Etiketten und das Erstellen der Datenbankausprägungen für den Datenbankzustand. Weiterhin bietet das Paket neben der Schnittstelle zur labConvert Applikation, eine API für die skriptbasierte Konvertierung an, die in Quellcode 7.4 gezeigt ist.

Quellcode 7.4: Schnittstelle des Konvertierungsprogramms `xassp2emu` zur Konvertierung von Kiel Corpus *s1h*-Dateien in das EMU-Format zur Verwendung in einer EMU-Sprachdatenbank.

```
1 package require xassp2emuPack
2 package require xassp2anyPack
3 emu::x2e::x2econvert path/to/utterance/utterance
```

Das Konvertierungsprogramm selbst besteht jedoch nicht nur aus dem Tcl/Tk-Paket, sondern stellt ein Datenbanktemplate zur Verfügung, das von Nutzern selbst angepasst werden kann und muss.

7.3. Diskussion und Fazit

Für das Kiel Corpus konnte eine Methode zur Verfügung gestellt werden, die das Korpus in eine Kiel Corpus EMU-Datenbank überführt. Hierfür wurde ein geeignetes Etikettierungsschema modelliert, ein Datenbanktemplate erzeugt und eine Konvertierungsfunktion implementiert. In diesem Zusammenspiel konnten für das Kiel Corpus of Read Speech, das keine prosodische Etikettierung enthält, 3872 Datenbankäußerungen aus 3876 *s1h*-Dateien erzeugt werden. Die Fehlerquote der Methode liegt somit bei 0.1%. Die Anzahl der nicht konvertierbaren Äußerungen mit prosodischer Etikettierung sind nach Dialogen kategorisiert in Abbildung 7.8 dargestellt. Von 2523 *s1h*-Dateien konnten 2345 konvertiert werden. Für diesen Teil liegt die Fehlerquote bei 7%. Mit dem Blick auf die Abbildung fehlen sehr viele Konvertierungen (in Zahlen ca. die Hälfte aller) in den mit l0 bezeichneten Dialogen. Bei diesen Dialogen handelt es sich um die unveröffentlichten Lindenstraßendialoge. 22% des verwendeten Materials mit prosodischer Etikettierung ist unveröffentlicht. Unter Berücksichtigung dieser Fakten ist die Fehlerquote akzeptabel, denn dieses Material ist u. a. wegen mangelnder Konsistenz in den Dateien noch nicht veröffentlicht. Diese Inkonsistenzen werden zur Zeit bearbeitet, so dass in naher Zukunft das Kiel Corpus in einer überarbeiteten Version veröffentlicht werden wird[4].

Die möglichen auftretenden Fehler in den veröffentlichten Teilen des Corpus wurden detaillierter betrachtet und dokumentiert (John, 2003a). So sind Differenzen in der Wortanzahl zwischen angegebener Orthographie und der Phonetik innerhalb der *s1h*-Datei als Problem zu nennen. Zahlen sind inkonsistent in der Orthographie in

[4]Diese Information stammt aus der persönlichen Kommunikation mit Dr. Michael Scheffers am IPDS, der maßgeblich an der Arbeit des Kiel Corpus beteiligt ist.

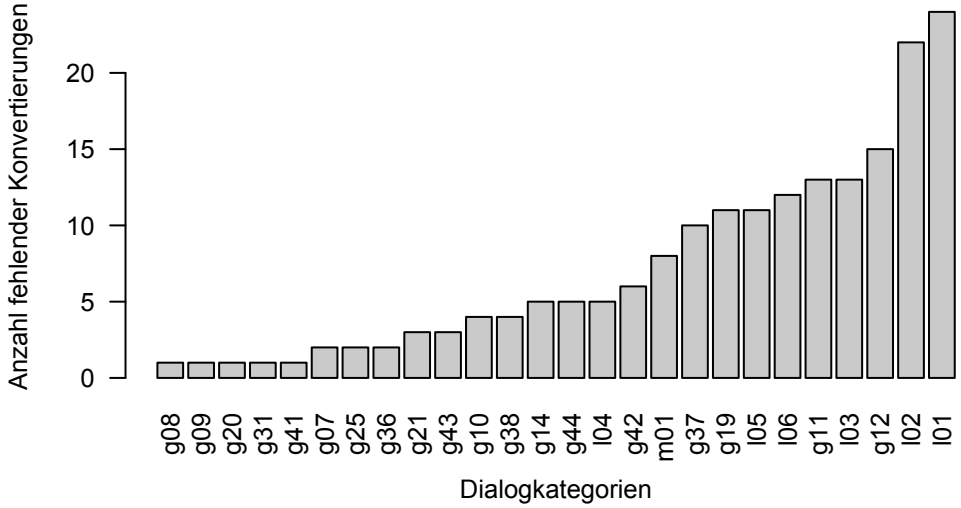


Abbildung 7.8.: Anzahl der nicht konvertierbaren Äußerungen des Kiel Corpus mit prosodischer Etikettierung in Dialogkategorien zusammengefasst. Die Grafik wurde in R auf Grundlage der Dateinamen der vorhandenen *s1h*- und *hlb*-Dateien erzeugt.

manchen Dateien als Ziffernfolge und somit als ein Wort zu finden, während in der segmentellen Etikettierung mehrere Wörter (z. B. pro Ziffer) zu finden sind.

Ein weiteres Problem ist der Umgang mit außersprachlichen Markierungen besonders für überlappende Geräusche. Hierfür gibt es keine konsistente Umsetzung in den *s1h*-Dateien, sodass falsches Setzen dieser Markierungen automatisch mit anderen Wortzuordnungsregeln kollidiert. Mit dem letzten Problem sind die häufigsten Fehlerquellen abgedeckt. In manchen *s1h*-Dateien fehlt die Wortende- bzw. Äußerungsendemarkierung, zumindest in der Form, wie sie vordefiniert sein sollte. Es gibt aber auch einzigartige Fehler, die sich nicht strukturiert häufen können, da es sich um Tippfehler und ähnliches handelt.

Das gehäufte Vorkommen von Fehlern innerhalb einer Dialogkategorie könnte auf den individuellen menschlichen Etikettierer zurückzuführen sein, der als Arbeitsumfang jeweils eine Dialogkategorie zu etikettieren hatte. Möglich wäre auch, dass in den Dialogkategorien durch den Kontext besonders viele Zahlen oder auch technische Abbrüche aufgrund zeitlich begrenzter technischer Probleme vorkamen.

Die bekannten Fehlerquellen hätten in der Implementierung berücksichtigt werden können. Jedoch können Korrekturen von möglichen Fehlern ad absurdum geführt werden, da der Fehler selbst schwer zu detektieren ist. Wenn Fehler auftreten, könnte jede Fehlerquelle in Betracht gezogen und die Konvertierung noch einmal durchgeführt werden, bis der Fehler gefunden ist. Da Fehler jedoch auch kombiniert auftreten können und die Zahl der nicht konvertierbaren Dateien im Verhältnis akzeptabel ist, wurde darauf verzichtet.

Die Fehlerquellen zeigen sehr schön, dass eine dateibasierte Verarbeitung von Sprachdaten leicht zu Inkonsistenzen führen kann. Über eine zentrale Verwaltung der Daten und mit einem Etikettierungsschema, das die Vorgaben für die Etikettierungen gibt, die durch das DBMS kontrolliert werden, sind derartige Fehler nicht zu erwarten. Das EMU-System kann diese Funktionalität bieten und zudem eine angepasste Abfragemöglichkeit zur Verfügung stellen. Damit das Kiel Corpus im gesamten Umfang datenbankbasiert verarbeitbar ist, sollte das Korpus in seiner überarbeiteten Version in der Zukunft noch einmal konvertiert werden.

Für eine erneute Konvertierung könnte das Hinzufügen der bisher nicht berücksichtigten Informationen aus den Etikettierungsdateien diskutiert werden. Zu diskutieren wären außersprachliche Ereignisse, die mit den sprachlichen Ereignissen überlappen, weiterhin die Unsicherheitsmarkierungen und die Interpunktion. Lösungsansätze wurden für die ersten beiden Probleme in der Diskussion des Etikettierungsschemata in Kapitel 7.2.1 vorgestellt. Für Letzteres könnte eine `Text`-Ebene modelliert werden, die den gesamten Text enthält[5].

Die Konvertierung des Kiel Corpus hat außerdem gezeigt, dass das EMU-System eine weite Palette an Funktionen zur Verfügung stellt, mit der eine solche Aufgabe umsetzbar ist.

[5]Hierbei wäre darauf zu achten, dass Etiketten im EMU-System bisher bis zu einer Zeichenkettenlänge von 70 bit akzeptiert werden (vgl. Kapitel 3.1.1).

8. EMU im relationalen Datenbankmodell – ein Ausblick

In Kapitel 2.1 wurde unter anderem das Datenmodell relationaler Datenbanken vorgestellt. Es stellt nur ein Objekt zur Modellierung zur Verfügung, nämlich Relationstypen, die sich durch die Menge von Attributen definieren. Relationen stellen die Ausprägung eines Relationstyps dar, wobei jede Relation durch ihre Attributwertmenge (Tupel, vgl. S. 38) definiert ist. Operationen in diesem Modell unterliegen der Relationenalgebra. Desweiteren wurde als Standardsprache für relationale Datenbanksysteme für die beiden DBMS Sprachen DDL und DML die Sprache SQL vorgestellt und darauf hingewiesen, dass Relationstypen im relationalen Datenmodell in SQL als Tabellen umgesetzt sind. Dabei stellt jede Spalte ein Attribut und jede Reihe eine Relation dar, wobei wiederum jede Reihe als relationales Tupel, nämlich als Attributwertmenge aufgefasst werden kann.

Eine Modellierung und das Einpflegen von Etikettierungen in eine relationale Struktur ist aufgrund der Eigenschaften von Etiketten, rein zeichenbasierend zu sein, möglich. In Kapitel 3 wurde das Etikettierungsmodell in EMU auf konzeptueller Ebene vorgestellt und gezeigt, in welcher Struktur die Etikettierungen dateibasiert in der Datenbank vorliegen. Durch die Struktur der Dateien und die kommentierte ER-Modellierung der internen Strukturen wurde ein Einblick in das logische EMU-Etikettierungsmodell des Systems gegeben. Als Abfragesprache wurde die EQL vorgestellt, die Abfragen für alle Strukturen, die mit diesem Modell möglich sind, abfragen kann.

Cassidy (1999a) äußert Bedenken bezüglich der Effizienz der EQL auf wirklich große Datenbanken. Bedingt ist diese Aussage durch die verwendeten Speicherstrukturen im EMU-System, die für Anfragen eine Vielzahl an Dateiöffnungs- und Leseoperationen erfordert. Diese Operationen sind bei SQL-Abfragen in relationalen Datenbanken nicht zu erwarten, da die Datenbank zur Laufzeit des Datenbanksystems nicht in Dateien gespeichert ist, sondern in relationalen Tabellen, deren physische

Umsetzung implementierungsabhängig ist. Durch eine Überführung von Etikettierungen aus dem EMU-System in ein relationales System, das SQL-Abfragen auf die EMU-Datenbasis ermöglicht, sind somit geringere Abfragedauern auf die gleiche Datenbank zu erwarten.

Cassidy (1999a) nahm sich dieser Idee an. Er überführte eine EMU-Datenbank in eine relationale und verglich EQL-Abfragedauern mit SQL-Abfragedauern. Das von ihm verwendete relationale Datenbanksystem war die freie MySQL-Software. Hierfür wurde eine EMU-Testdatenbank im Umfang von 136749 Segmenten und 438559 Assoziationen zwischen den Segmenten aus 1000 Äußerungen gelesener Sätze verwendet. Das EMU-Etikettierungsschema enthielt sowohl intervall-zeitgebundene als auch zeitlose Ebenen in mehreren Ketten. Dabei standen fast alle Segmente in hierarchischer Beziehung zu anderen Segmenten. Bedingt durch das Schema konnten alle Arten von Abfragen getestet werden.

Als Ergebnis seiner Studie ist zunächst festzustellen, dass ein relationales Datenbankschema entworfen werden kann, das EMU-Etikettierungsstrukturen abbildet. An diesem können Abfragen mit MySQL formuliert werden, so dass sie zum gleichen Ergebnis führen, wie eine Abfrage mit EQL auf der EMU-Datenbank. Seine Ergebnisse zeigen weiterhin überzeugend die hohe Effizienz der relationalen Strukturierung der Daten für die Abfrage. Tabelle 8.1 zeigt Dauermessungen der durchschnittlichen Bearbeitungszeiten für die dreifache Ausführung von verschiedenen Abfragen, die untersucht wurden. Die Tabelle stellt die Rechenzeiten für die gleichen Abfragen mit EQL (vgl. Kapitel 3.3) im EMU-System und mit MySQL (vgl. Kapitel 2.2) in der relationalen Datenbank gegenüber. Für einfache Abfragen waren

Tabelle 8.1.: Durchschnittliche Abfragedauern in Sekunden für verschiedene Abfragen jeweils im EMU-System und in SQL (Pentium 266, 96 MB) aufbereitet aus Cassidy(1999).

Art der Abfrage	EMU in *s*	SQL in *s*
Einfache-Abfrage	34	5
Sequenz-Abfrage	34	22
Dominanz-Abfrage	31	24
Disjunktion	45	45
Komplexe Abfrage	40	22

die MySQL-Abfragen auf die relationale Datenbank sieben Mal schneller als mit EQL im EMU-System. Für komplexe Abfragen wurde die Rechenzeit halbiert. Für

Dominanz- und Sequenzabfragen wurden nur zwei Drittel der Zeit benötigt. Die Ergebnisse sind beeindruckend.

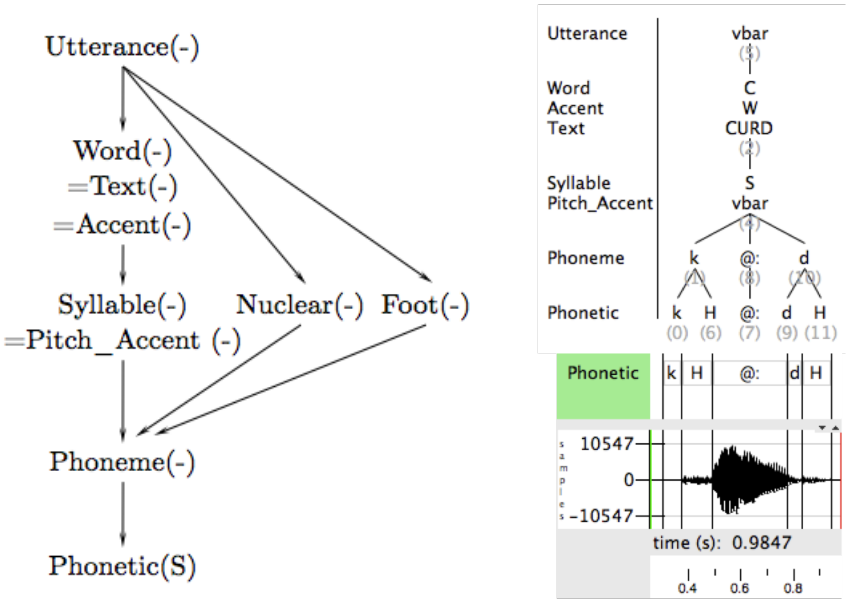
Da sich die Modellimplementierung in EMU seit 1999 nicht wesentlich geändert hat, sich relationale Datenbanken aber etabliert haben, wären in einer vollständigen Wiederholung des Experimentes ähnliche Ergebnisse in den Unterschieden der Abfragegeschwindigkeit vorhersagbar, obgleich die effektiven Geschwindigkeiten aufgrund der höheren Rechnerleistung höher sein sollten. Während Cassidy in 1999 die Effizienz der EQL auf sehr große Datenbanken anzweifelte, ist das für die Effizienz der SQL in 2011 auch für größere Datenbanken als die in Cassidy (1999a) verwendete, nicht begründbar. Aber es wäre zu prüfen, ob das Überführungsverfahren nach Cassidy (1999a) auch für größere Datenbanken anwendbar ist.

Das Überführen von EMU-Etikettierungen in ein relationales Datenbankmodell wird derzeit für die Weiterentwicklung des EMU-Systems diskutiert, um die durch die Weiterentwicklung im Rahmen dieser Arbeit sehr anwenderfreundlich nutzbare Software zusätzlich mit einer zeitlich effizienteren Abfragemöglichkeit anbieten zu können. Als Diskussionsgrundlage wurde in der vorliegenden Dissertation eine EMU-Datenbank nach dem Ansatz aus Cassidy (1999a) in eine relationale Datenbank überführt, die Funktionsweise der verwendeten Skripte eruiert und für die vorliegende EMU-Version aufbereitet, das relationale Datenbankschema mit dem aktuellen Etikettierungsschema verglichen, die Komplexität und Benutzerfreundlichkeit der Abfrageausdrücke überprüft, um schließlich notwendige Änderungen diskutieren zu können. Die einzelnen Punkte werden im Folgenden näher ausgeführt.

8.1. Methode

Für die Überführung wurde, anders als bei Cassidy (1999a), die `demo`-Datenbank verwendet, die als eine kleine Beispieldatenbank mit dem EMU-System ausgeliefert wird und Daten aus Millar, Vonwiller, Harrington und Dermody (1994) und Millar, Dermody, Harrington und Vonwiller (1997) enthält. Gewählt wurde dieses Beispiel, da die Etikettierungsstrukturen sehr ähnlich zum Beispiel in Cassidy (1999a) sind. Abbildung 8.1 zeigt das Etikettierungsschema und die Etikettierungen in der Hauptkette der Äußerung msdjc001 der Beispieldatenbank `demo`.

Um die vorhandene EMU `demo`-Datenbank in eine relationale Datenbank zu überführen, wurde das von Cassidy im Artikel erwähnte und auf der EMU-Internetseite



(a) (b)

Abbildung 8.1.: Etikettierungsschema der `demo`-Datenbank in (a) und die Ausprägung durch die Äußerung msdjc001 in (b). Die Etikettierung der Nuclear und Foot Ebene sind nicht enthalten.

(Cassidy, 1999b) erhältliche Tcl-Skript `emu2dbase` leicht modifiziert, um es für EMU 2.3 kompatibel zu machen. Das angepasste Skript ist im Anhang C.4 angefügt.

Das Skript wurde ausgeführt. Wie in der Dokumentation des Skripts beschrieben, entstanden die Dateien database, levels.db, links.db und tokens.db. Die für `demo` erfolgreich kreierte `database` Datei (vgl. Abbildung 8.2) enthält die MySQL-Befehle zum Erstellen der Datenbank `demo`, zur Erstellung des Datenbankschemas und schließlich zum Einpflegen der Daten in die Datenbank. Die Datenbankausprägung liegt tabellarisch (tabsepariert) in den für `demo` ebenfalls erzeugten *.db*-Dateien vor.

Mit dem MySQL-Befehl CREATE DATABASE (vgl. Abbildung 8.2) wird die `demo`-Datenbank erzeugt. In diesem Zustand ist die Datenbank ohne Datenbankschema und ohne Datenbankausprägungen. Die Befehle CREATE TABLE erzeugen die Relationstypen `levels`, `tokens` und `links`. Nachdem alle Relationstypen erstellt sind, ist die Datenbank vollständig definiert und in einem leeren Datenbankzustand.

```
1  DROP DATABASE IF EXISTS demo;
2  CREATE DATABASE demo;
3  use demo;
4  CREATE TABLE levels (
5           id       int NOT NULL PRIMARY KEY,
6           name     char(20) NOT NULL
7  );
8  CREATE TABLE tokens (
9        id     char(20) NOT NULL PRIMARY KEY,
10       level int NOT NULL,
11       stime real NOT NULL,
12       etime real NOT NULL,
13       seq    int NOT NULL,
14       label char(20) NOT NULL,
15       utt    char(50) NOT NULL,
16       INDEX labelindex (label)
17 );
18 CREATE TABLE links (
19          parent  char(20) NOT NULL,
20          child   char(20) NOT NULL,
21          INDEX parentindex (parent),
22          INDEX childindex (child)
23 );
24 LOAD DATA LOCAL INFILE 'tokens.db' INTO TABLE tokens;
25 LOAD DATA LOCAL INFILE 'levels.db' INTO TABLE levels;
26 LOAD DATA LOCAL INFILE 'links.db' INTO TABLE links;
```

Abbildung 8.2.: MySQL-Befehle zum Aufbau der Datenbank in der `database` Datei.

Über die MySQL-Befehle LOAD DATA... (Abbildung 8.2) wird der Datenbankzustand geändert, denn es werden die Daten aus den *db*-Dateien in die Datenbank eingepflegt. Die *.db* Dateien enthalten relationale Tabellen im ASCII Format.

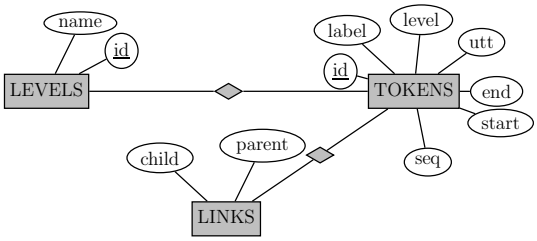
Aus den MySQL-Befehlen wurde das logische Datenbankschema automatisch durch die Verwendung einer Datenbanksoftware MySQL-Workbench (Oracle Corporation, 2011) abgeleitet. Zu Vergleichszwecken wurde aus diesem Schema und den Datenbankausprägungen das interne Schema auf konzeptueller Ebene in ER-Darstellung modelliert (vgl. Abbildung 8.3(a)). Das logische Schema ist in Abbildung 8.4(a) dargestellt und zeigt die drei Relationstypen mit ihren Attributen. Auch Abbildung 8.4(b) stellt das Schema weniger implementationsnah als SQL-Tabellen dar und enthält zusätzlich Datenbankausprägungen.

8.2. Diskussion der Schemata und Ausprägungen

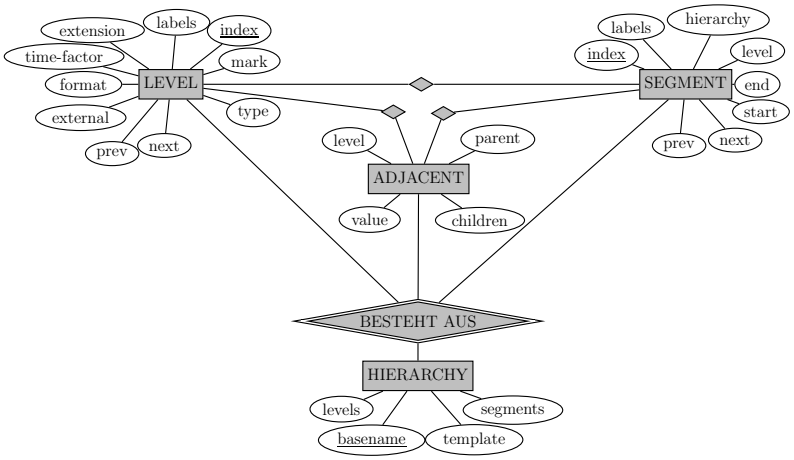
Zum Vergleich der konzeptuellen Etikettierungsschemata auf interner Ebene stehen in Abbildung 8.3 die Modellierungen im Entity-Relationship Modell für den relationalen Ansatz in (a), im Folgenden mit EMU v.rel bezeichnet, und für das in den Vorarbeiten erstellte derzeitige in (b), im Folgenden EMU v.2 genannt, zur Verfügung. Ein Vergleich der beiden internen Datenbankschemata für die Etikettierung zeigt deutliche Unterschiede aber auch Gemeinsamkeiten. Beide Schemata modellie-

ren die Objekte der realen Welt Ebene, Segmente bzw. Tokens und Assoziationen. Das Objekt Äußerung ist hingegen in EMU v.2 als Entitätstyp **HIERARCHY** realisiert und fehlt als solcher in EMU v.rel.

Ein Blick auf die Modellierung von Ebenen zeigt für **LEVELS** in EMU v.rel und **LEVEL** in EMU v.2 die gleichen Attribute für die Modellierung der Ebenennamen als **name**- bzw. **labels**-Attribut und eine Indizierung der Ebenen mit **id** bzw. **index**, jedoch weist **LEVEL** hier sehr viel mehr Attribute auf. Ähnliches gilt für



(a) EMU v.rel



(b) EMU v.2

Abbildung 8.3.: Ausschnitt des internen Etikettierungsschemas in ER-Darstellung im relationalen EMU (EMU v.rel) (a) und zum Vergleich im EMU in der aktuellen Version (EMU v.2) (b) – ein Ausschnitt aus Abbildung 3.8.

TOKENS und **SEGMENT**, die Tokens auf Ebenen modellieren. In beiden Entitätstypen sind die Attribute **label** bzw. **labels** gegeben. Unterschiede bestehen in der Modellierung der sequentiellen Reihenfolge der Segmente auf den Ebenen. Anstatt über **prev**- und **next**-Attribute wie es für EMU v.2 der Fall ist, wird dies in EMU v.rel mit dem Attribut **seq** modelliert. Weiterhin unterscheiden sich die Attribute in der Bezeichnung, die auf das Objekt Äußerung Bezug nehmen. In EMU v.2 wird es mit **hierarchy** und in v.rel mit **utt** modelliert.

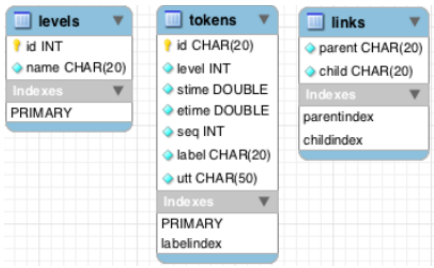
Die Modellierung der Assoziationen ist in beiden Modellen fast gleich, unterscheidet sich nur in der Namensgebung **LINKS** in v.rel und **ADJACENT** in v.2, außerdem in einem **value** und **level**-Attribut, das gegenüber dem v.rel in v.2 vorhanden ist. Der Unterschied in der Namensgebung der Attribute **child** und **children** gehen auf unterschiedliche implementierungsnahe Strukturen zurück, modellieren aber den gleichen Ausschnitt der realen Welt.

Der auffälligste Unterschied zwischen beiden Schemata ist die Art der Verknüpfung zwischen den genannten Entitätstypen, die Ebenen, Tokens und Assoziationen modellieren. Während in EMU v.rel alle Entitätstypen in einer Sequenz über zwei Beziehungstypen miteinander verknüpft sind, stehen sie in EMU v.2 alle miteinander in drei Beziehungstypen. Weiterhin sind in EMU v.2 alle Entitätstypen im Beziehungstyp **BESTEHT AUS** mit dem gegenüber EMU v.rel nicht vorhandenen **HIERARCHY**-Entitätstypen verknüpft.

Im Vergleich der Datenbankausprägungen (Kapitel 3.2.2 für EMU v.2 und Abbildung 8.4(b) auf Seite 224) sind ebenfalls Unterschiede, aber auch Gemeinsamkeiten zu erkennen. Die beiden Entitätstypen **LEVELS** in v.rel und **LEVEL** in v.2 enthalten Indexnummern in den Attributen **id** bzw. **index**. Unterschiedlich hingegen sind die Ausprägungen für **name** bzw. **labels**. Hier sind in v.2 Einträge für jeden Label-Link vorgesehen, während in v.rel nur ein Ebenenname pro Entität enthalten sein kann.

Auch ähnliche Ausprägungen gelten für das in beiden Schemata modellierte Objekt Assoziation (**LINKS** und **ADJACENT**). In beiden Schemata werden Segmente gepaart. Die Angabe, ob eine Assoziation zwischen den Segmenten besteht oder nicht, unterscheidet die beiden Modellierungen. In EMU v.2 sind alle Paarungen von möglichen Eltern- und Kindsegmenten enthalten und es ist über einen booleschen Wert angegeben, ob die Assoziation besteht. Dagegen sind in EMU v.rel nur Tupel eingetragen, für die es eine Assoziation zwischen den entsprechenden referierten Tokens gibt.

Wie aus Kapitel 3.2.2 hervorgeht, gibt es im internen Schema im derzeitigen EMU-Datenmodell Segmente, die mit einer Segmentnummer versehen sind (**index** von **SEGMENT**). Diese Segmentnummer kann jedoch nicht als **TOKENS id** im EMU v.rel verwendet werden, da diese für jede Äußerung (**HIERARCHY**) erneut von 0 bis n nummeriert sind. Im relationalen Datenbankschema sind alle Etikettierungen aller Äußerungen in einer Tabelle modelliert, so dass es zu gleichen Ausprägungen im **id**-Attribut kommen würde, was für Schlüsselattribute nicht erlaubt ist. Die in der



(a) logisches Datenbankschema.

(b) SQL-Tabellen

LEVELS

id	name
1	Utterance
2	Word
3	Syllable
4	Phoneme
5	Phonetic
6	Foot
7	Nuclear

LINKS

parent	child
msdjc001/5	msdjc001/2
msdjc001/5	msdjc001/4
msdjc001/5	msdjc001/1
msdjc001/5	msdjc001/8
msdjc001/5	msdjc001/10
msdjc001/5	msdjc001/0
msdjc001/5	msdjc001/6
msdjc001/5	msdjc001/7
msdjc001/5	msdjc001/9
msdjc001/5	msdjc001/11
msdjc001/2	msdjc001/4
msdjc001/2	msdjc001/1
...	...
msdjc006/3	msdjc006/11

TOKENS

id	level	stime	etime	seq	label	utt
msdjc001/5	1	2867.650000	3513.650000	1	vbar	msdjc001
msdjc001/2	2	2867.650000	3513.650000	1	C	msdjc001
msdjc001/4	3	2867.650000	3513.650000	1	S	msdjc001
msdjc001/1	4	2867.650000	3059.650000	1	k	msdjc001
msdjc001/8	4	3059.650000	3343.650000	2	@:	msdjc001
msdjc001/10	4	3343.650000	3513.650000	3	d	msdjc001
msdjc001/0	5	2867.650000	2939.650000	1	k	msdjc001
msdjc001/6	5	2939.650000	3059.650000	2	H	msdjc001
msdjc001/7	5	3059.650000	3343.650000	3	@:	msdjc001
msdjc001/9	5	3343.650000	3401.650000	4	d	msdjc001
msdjc001/11	5	3401.650000	3513.650000	5	H	msdjc001
msdjc001/3	6	2867.650000	3513.650000	1	S	msdjc001
...	...	...	...	...	...	...
msdjc006/3	6	20271.25000	20785.25000	1	vbar	msdjc006

Abbildung 8.4.: Logisches Datenbankschema einer EMU-Datenbank (a) nach der Konvertierung in ein relationales Datenmodell mit `emu2dbase` von Cassidy(1999) erstellt mit MySQL-Workbench (Oracle Corporation 2011) mit den relationalen Tabellen in (b), die aus den jeweiligen Dateien levels.db, links.db und tokens.db in die Datenbank eingepflegt sind. Die Tabellen LINKS und TOKENS zeigen aus Platzgründen nur die Ausprägungen der Äußerung msdjc001 vollständig.

Überführung für EMU v.rel verwendete Lösung für die Schlüsselattributwerte von **id** ist eine Zusammensetzung des Äußerungsnamens (**HIERARCHY basename** in v.2) mit der Segmentnummer (vgl. Zeile 44–46 und 78 in C.4).

Das **seq**-Attribut in **TOKENS** stellt für v.rel anhand einer Ziffer die Position eines Segments in einer sequentiellen Ordnung dar, während in EMU v.2 **prev** und **next** in **SEGMENT** Segmentnummern benachbarter Segmente enthält. Der Unterschied in den Ausprägungen von **label** in EMU v.rel und **labels** in EMU v.2 liegt, wie der Name vermuten lässt, in der Menge der Etiketten, die repräsentiert werden. So ist es in v.rel nur ein Etikett und in EMU v.2 alle Etiketten auf allen Label-Links.

Start- und Endzeiten von Segmenten sind in beiden Schemata obgleich Ausprägungen von Attributen mit unterschiedlichen Bezeichnungen mit gleicher Ausprägung versehen. Der **TOKENS**-Entitätstyp in v.rel zeigt Attribute **stime** und **etime**, **SEGMENT** in v.2 hingegen **start** und **end**, die die Start- und Endzeiten der Segmente modellieren. Das wird deutlich, wenn die Zeitmarken der Segmente in Abbildung 8.1(b) mit den Ausprägungen der **TOKENS**-Tabelle in 8.4(b) verglichen werden. Sie stimmen beim Hinzurechnen der Signalstartzeit[1] und der Umrechnung in ms überein.

Aus der Gesamtdarstellung in Abbildung 8.4 geht nicht hervor, wie zeitlose Segmente repräsentiert sind. Nach Abbildung 8.2 (S. 221) müssen die Zeitattribute (**stime**, **etime**) mit einem Zahlenwert belegt werden, sie dürfen nicht undefiniert (vgl. Zeile 11-12: NOT NULL) sein. Bei der Anfrage der Zeiten[2] gibt das EMU-DBMS für zeitlose Segmente ohne ableitbare Zeiten den Wert -1.0000000 zurück, der bei der Überführung in die relationale Datenbank übernommen wurde. Dieser Wert ist definiert und damit verschieden von NULL. Das birgt jedoch Gefahren in der Weiterverwendung durch andere Applikationen. Die derzeit bestehenden EMU-Applikationen und das EMU/R Paket berücksichtigen diese Umsetzung jedoch bereits, da es in der momentanen Implementierung des Modells genauso realisiert ist. Eine andere Modellierung wäre jedoch zu überlegen.

Die Beziehungen zwischen Ebenen und Segmenten sind in EMU v.2 in den Attributen der jeweiligen Entitätstypen modelliert, indem jedes Segment (Ausprägung von **SEGMENT**) eine Referenz zur zugehörigen Ebene im Attribut **level** enthält. Auch EMU v.rel enthält ein **level**-Attribut in **TOKENS**, dass jedoch keine eindeutig

[1] Die Sprachsignale liegen im SSF-Format (vgl. 3.2.2.2) vor. Die Startzeit der Äußerung in Abbildung 8.1 ist mit 2,563s im SSFF Header angegeben. Diese Zeitspanne muss der Zeitskala jeweils hinzugerechnet werden.

[2] emutemplate hierarchy ($hier) Kommando in Zeile 80 in C.4

modellierte Referenz zu Ebenen zeigt. Der Zusammenhang ist aus dem Datenbankschema in Abbildung 8.4 nicht ersichtlich, da diese Beziehung nicht, wie in Kapitel 2.1.2 vorgeschlagen, mithilfe gleicher Attributnamen in den unterschiedlichen Relationstypen umgesetzt wurde. Wie in Kapitel 3.1 jedoch erwähnt wurde, sind bei der Modellierung immer die Abfragemöglichkeiten zu berücksichtigen. Die Verbindung zwischen dem **level**-Attribut von **TOKENS** und dem **LEVELS**-Entitätstyp sind über Abfragen formulierbar. Daher ist diese Verbindung in Abbildung 8.3(a) über einen Beziehungstypen dargestellt worden.

Weiterhin unterscheiden sich die Ausprägungen für das Objekt der realen Welt Äußerung in beiden Schemata enorm, da sie in EMU v.2 als Attribut zu **SEGMENT** zur Laufzeit des Programms und der Datenbank erzeugt werden und eigentlich nicht explizit in der Datenbasis umgesetzt sind. **hierarchy** in v.2 enthält daher in der Ausprägung ausschließlich Verweise (Pointer) auf die bestehenden Entitäten in **HIERARCHY**[3]. In EMU v.rel sind sie hingegen durch den Basisnamen der Äußerung im **utt**-Attribut von **TOKENS** in der Datenbankausprägung enthalten, wie in der **TOKENS**-Tabelle in Abbildung 8.4(b) deutlich zu erkennen ist.

8.3. Abfrage im relationalen Modell mit SQL

Die Abfrage im relationalen Modell erfolgt über die Standarddatenbanksprache SQL. Das bedeutet, ein *select-from-where* Block wird verwendet, der das gesuchte Attribut in **SELECT** enthält, die Auswahl der Tabellen aus denen die relationalen Tupel gesucht werden sollen in **FROM** und die Einschränkungen auf die gesuchten Tupel in **WHERE**. In diesem Zusammenhang müssen die bisher verwendeten Entitätstypen mit den Relationstypen im logischen Schema in Abbildung 8.4(a) in der Argumentation herangezogen werden, da die Operationen nur auf Relationen nicht auf Entitäten ausführbar sind.

Eine Etikettabfrage als einfache Abfrage in EQL wie in EQ-B 2 wird aufgrund des Schemas und der Syntax sowie der Semantik der SQL in der Umsetzung in MySQL in dem *select-from-where* Block in SQ 5 formuliert. In der Abfrage sollen Tokens mit dem Etikett `vbar` gesucht werden.

EQ-B 2 (zu SQ 5). `Utterance = vbar`

[3] vgl. Kapitel 3.2.2

SQ 5 (EA).

```
SELECT
   tokens.label
FROM
   tokens, levels
WHERE
   tokens.label = 'vbar' and
   tokens.level = levels.id and
   levels.name = 'Utterance'
```

Die eigentliche Abfrage erfolgt intern über eine Operation der Relationenalgebra[4]. Zum besseren Verständnis der Abfrage und wie die Relationstypen miteinander verbunden werden können, kann die Abfrage von unten nach oben gelesen werden. In der letzten Zeile der Abfrage, gibt es ein **name**-Attribut in **levels**. Jede Relation in **levels** hat eine Ausprägung für **name** und das Attribut **id**. Das Tupel, das als Ausprägung von **name** `Utterance` besitzt, hat auch eine Ausprägung von **id**. Für diese Abfrage hat das Tupel die **id** Ausprägung `1`, wie aus der Tabelle **LEVELS** in Abbildung 8.4(b) hervorgeht. In der vorletzten Zeile der SQ 5 wird diese ID mit dem **level**-Attribut des **tokens** Relationstyps in Verbindung gebracht. Die Ausprägungen **level** in der **TOKENS** Tabelle und **id** in der **LEVELS** Tabelle müssen identisch sein, damit das Tupel als der Abfrage entsprechend ausgewählt wird. Die Tupel aus **tokens**, für die das der Fall ist, müssen laut der drittletzten Zeile außerdem im **label**-Attribut `vbar` besitzen, um der Abfrage zu entsprechen. Die **SELECT**-Anweisung in den ersten beiden Zeilen der Abfrage bedingt die Rückgabe der **label**-Ausprägungen[5] der gefundenen **tokens**-Tupel.

Eine vermeintlich komplexere, aber bei genauem Hinschauen doch leichte Abfrage, ergibt sich für Dominanzabfragen (vgl. EQ 15). Wie in Kapitel 3.3 beschrieben wurde, werden in einer Dominanzabfrage zwei einfache Abfragen miteinander verbunden.

EQ 15 (zu SQ C.5). [`Phoneme = d ^ Phonetic = H`]

Als SQL-Anweisung ausgedrückt, bedeutet es: die Anweisungen in SQ 5 zweimal jedoch mit zwei verschiedenen **tokens**-Tupel und zwei verschiedenen **levels**-Tupel in einer *select-from-where* Anweisung. Verbunden werden die beiden Segmente dem Schema entsprechend über die Assoziationen **links**. Eine Dominanzabfrage für die

[4]Relationenalgebraoperation Join:
$R = tokens \bowtie_{label='vbar' \wedge level=level.id \wedge levels.name='Utterance'} levels$

[5]durch Projektion von R aus dem Join auf das Attribut label: $\pi_{label}(R)$

Abfrage in EQ 15 ist in einem kommentierten Skript (#=Kommentar) im Anhang C.5 zu finden.

Durch die Verbindung der Tabellen ist auch erklärbar, warum der Relationstyp **levels** bzw. der Entitätstyp **LEVELS**, der die Ebenen repräsentiert keinerlei Attribute bereitstellt, die die Zeitgebundenheit und die möglichen Eltern-Kind Relationen zwischen den Ebenen angibt. Derartige Informationen sind nur an den Tokens modelliert und aus diesen ableitbar, sofern Tokens für die Ebene vorhanden sind.

8.4. Schemaüberarbeitung

Die Datenbankausprägung wurde im bisherigen Vergleich der beiden Schemata in Augenschein genommen und es wurde festgestellt, dass sämtliche Label-Links aus dem Etikettierungsschema fehlen. Unter der Betrachtung des Schemas und des Skripts in C.4 ist kein großer Aufwand zu erwarten, um auch Label-Links in die Datenbank einzupflegen. Diese Ebenen müssten bei der Anfrage der Daten aus der EMU-Datenbank berücksichtigt werden, was im Skript in C.4 nicht der Fall ist. Die Segmente auf Label-Links hätten als **tokens** Tupel eine identische Ausprägung wie die Segmente, zu denen sie in linearer Beziehung stehen, bis auf das Attribut **label**. Das heißt, sie haben auch die selbe Segmentnummer im EMU-Datenbankzustand. Die Tupel müssen sich jedoch in den **id**-Ausprägungen unterscheiden.

Unter Berücksichtigung dieser Aspekte wäre eine mögliche Lösung, das `emu2dbase` Skript um folgende Anweisungen zu erweitern: Alle Tupel[6], die Segmentnummern der Segmente auf der linear verknüpften Ebene zum Label-Link enthalten, werden kopiert. In der **TOKENS** Tabelle werden die kopierten Tupel mit neuen Segmentnummern versehen. Die neuen **id**-Ausprägungen könnten durch weiteres Durchnummerieren der Segmente realisiert werden. In der **LINKS** Tabelle werden die **id**-Ausprägungen in den kopierten Tupel entsprechend ersetzt.

Cassidy (1999a) macht explizit auf das Fehlen der Label-Links aufmerksam, sowie auf die fehlende Unterscheidung zwischen intervall- und eventzeitgebundenen Ebenen. Außerdem bietet diese Modellierung keine Möglichkeit der Positionsabfragen und der Anzahlabfragen, die EQL ermöglicht. Er weist jedoch darauf hin, dass derartige Ergänzungen leicht umsetzbar wären. Dieser Umstand scheint durch den

[6] genauer angehende Datenbanktupel

entworfenen Lösungsansatz für die fehlenden Label-Links nicht von der Hand zu weisen zu sein.

Weiterhin gibt er die Möglichkeit zu bedenken, im Datenbankschema eventuell für jede Ebene einen Relationstypen zur Verfügung zu stellen. Über diesen Ansatz könnten Abfragen eventuell noch effizienter gestaltet werden. Trotz dieses möglichen Vorteils bringt ein solcher Ansatz auch Nachteile mit sich. Die relationalen EMU-Datenbanken würden sich dann im logischen Datenbankschema unterscheiden, denn die Nutzer sind im EMU-System nicht auf ein Etikettierungsschema eingeschränkt. Die Vielfalt der Etikettierungsmöglichkeiten in ganz unterschiedlichen und selbst zusammenstellbaren Strukturen ist eine der Eigenschaften, die das EMU-System von mehr oder weniger vergleichbarer Software abhebt. Da die automatische Abbildung des Etikettierungsschemas in ein relationales Datenbankschema vom System selbst erfolgt, wäre das Schema systemweit bekannt und könnte für alle Operationen individuell berücksichtigt werden. Es besteht dann jedoch nicht die Möglichkeit, häufig verwendete Operationen im System vorzudefinieren, die auf jede Datenbank anwendbar wären. Das Datenbankschema sollte für alle Datenbanken identisch sein. Das konzeptuelle Etikettierungsschema ist davon ausgeschlossen, da hier die Unterschiede zwischen den EMU-Datenbanken bestehen. Als Schemata auf externer Ebene sind sie auch sehr weit vom internen Datenbankschema entfernt, wie das Überführungsbeispiel mehr als deutlich macht.

8.5. Fazit und Ausblick

Der Ansatz, EMU-Etikettierungen in relationalen Datenbanken abzufragen, wird als bewährt eingestuft. Jedoch müsste das Datenbankschema überdacht und überarbeitet werden. Hierfür sollten nochmals verschiedene Schemata getestet werden und zwar in Hinblick auf Effizienz und die Wahrung aller bestehenden funktionalen EMU-Eigenschaften in der Etikettierungsstruktur und den Abfragemöglichkeiten.

Datenbankschema nach Cassidy (1999a) Einige der im relationalen gegenüber dem aktuellen Schema fehlenden Attribute müssen eventuell wieder in die Modellierung einbezogen werden. Weiterhin sollte ein passendes Schema keine unnötige Komplexität in den SQL-Abfragen erfordern. Wie Cassidy (1999a) selbst erwähnt, könnte die Schnittstelle zum Benutzer weiterhin die Syntax und Semantik der EQL sein, so dass die Komplexität für den Nutzer nicht zum Nachteil wird. Die Aufga-

be der EMU-Entwickler wäre dann, EQL-Abfragen automatisch auf SQL-Abfragen abzubilden. Diese Aufgabe wäre zum Beispiel mit einem einheitlichen Datenbankschema, wie es hier vorgestellt wurde, leichter umzusetzen im Gegensatz zum Ansatz, die Ebenen als Relationstypen zu definieren. In den modernen EMU-Versionen ist die Benutzerschnittstelle zur Abfrage das im Rahmen dieser Arbeit entstandene queryGUI (vgl. Kapitel 4.2). Für das queryGUI ist keine Übersetzung von EQL zu SQL notwendig. Es könnten direkt aus den Nutzereingaben SQL-Abfragen automatisch erzeugt werden. Dennoch gilt es zu bedenken, dass auf diese Weise immer eine grafische Oberfläche und somit eine EMU-Applikation mehr für den Zugriff auf die Datenbank verwendet werden muss.

Das hier vorgestellte Datenbankschema ist ein geeignetes Schema, um vorhandene EMU-Datenbanken zu überführen.

Vollständiges Datenbankschema für EMU Nach eigenem Kenntnisstand gibt es dennoch keinen Ansatz, der weder alle Arten von EMU-Abfragen, wie sie in Kapitel 3.3 erläutert wurden, umsetzen kann. Noch gibt es einen Ansatz der die Daten wirklich dem EMU-Datenmodell wie in Abbildung 3.9 auf S. 78 gerechtwerdend modelliert. Abbildung 8.3(a) enthält nur die Etikettierungskomponente muss aber dennoch als vollständige Modellierung einer EMU-Datenbank angesehen werden. Demgegenüber, sind in Abbildung 3.9 wesentlich mehr Einschränkungen in der Etikettierung aber auch im ganzen mehr Komponenten vorhanden.

Im Gegensatz zu Cassidy (1999a) und anderen Ansätzen[7], müssen die Modellierungen außerdem nicht nur die Datenbankabfrage sondern auch die Datenbankerzeugung berücksichtigen, das heißt das Etikettieren von Signalen und das damit verbundene Einpflegen der Daten in die Datenbank auf diesem Weg.

Anders als bei Firmendatenbanken sind während der Etikettierung, d. h. beim Hinzufügen und Löschen von Segmenten, jeweils andere Tupel einer Äußerung von den Änderungen betroffen. Denn es ändert sich auf jedem Fall der Wert des **seq**-Attributes im **tokens** Relationstyp für alle Tupel der nachfolgenden Tokens auf gleicher Etikettierungsebene. Das bedeutet, der Datenbankzustand ändert sich enorm mit jeder dieser Operationen. Hierfür müssten die entsprechenden Funktionen effizient im Datenbankmanagementsystem implementiert werden.

[7] persönliche Kommunikation mit engagierten EMU-Anwendern

Bisher gibt es keinen Ansatz, der nur im Entferntesten das modellgerechte Einpflegen von Tokens, was beim Etikettieren unumgänglich ist, berücksichtigt. Hierbei kommen Integritätsbedingungen und dergleichen zum Tragen. Diese mussten in den vorliegenden Ansätzen noch keine Berücksichtigung finden, da die Integrität der Daten in vorhandenen EMU-Sprachdatenbanken vom momentan verwendeten DBMS bereits geprüft wurde.

Bisher unberücksichtigt sind auch die Signaldateien, die im EMU-System eindeutig zur Datenbank gehören. Die Signaldateien selbst können als Dateien, d. h. nicht als relationale Tabellen etwa in die Datenbank integriert und somit vom Datenbanksystem zentral verwaltet und vom Nutzer abgeschirmt werden. Der Speicherort der Dateien könnte für den schnellen Zugriff auf die Information in einem zusätzlichen Relationstyp in die Datenbank eingepflegt werden. Auf diese Weise würde die Datenbank Etikettierungen und Signale enthalten. Die Datenbank wäre vollständig, obgleich die eigentlichen Signale als Dateien separat vorliegen.

Für das Einbinden medialer Daten in Datenbanken werden derzeit objektorientierte Datenmodelle entwickelt. An dieser Stelle wäre also auch dieser Ansatz zu überlegen. EMU ist eine *open source* Software und alle verwendeten Module müssen *open source* sein. Objektorientierte Datenbanksysteme sind noch nicht als zuverlässige *open source* Systeme verfügbar und somit in EMU noch nicht verwendbar.

Verschiedene Ansätze zur Überführung in das relationale Modell zu entwickeln, ist wichtig für die Weiterentwicklung. Sie sind aber nur wirklich brauchbar, wenn sie wirklich die Funktionalität des EMU-Systems berücksichtigen, was als wesentlich komplexer eingeschätzt wird als die Überführung vorhandener Etikettierungen. Mit der Überführung vorhandener Datenbanken ist die Arbeit nicht getan. Es stellt in der EMU-Strukturierung lediglich eine Art von Applikation dar. Trotz erster Erfolge der vorliegenden Ansätze sollte die wirklich brauchbare Umsetzung der EMU-Datenbankfunktionalität im relationalen Datenmodell als Ziel für die Umstellung des ganzen Systems nicht aus den Augen verloren werden.

Organisation für den Austausch des Backends In Hinblick auf die Entwicklung des EMU-Systems für die Verwendung einer relationalen Datenbank muss entschieden werden, ob das Überführen als optionale Funktion bereitgestellt wird. Die Bereitstellung des Skripts in C.4 im Anhang, das im Rahmen dieser Arbeit für die aktuelle EMU-Version funktionstüchtig gemacht wurde, zusammen mit einer Dokumentation, wie sie durch diese Arbeit bereits vorhanden ist, wäre ein erster Schritt.

Alternativ wäre es zu überlegen, ob das gesamte System auf das neue Modell umgestellt wird. Im zweiten Fall müsste die gesamte `emu-core` Bibliothek ersetzt werden, aber auch in Hinblick auf die Kompatibilität zu älteren Datenbanken im System erhalten bleiben.

Obgleich die Umstellung des gesamten Systems oder nur das Anbieten der Überführung in das relationale Datenmodell auf Anfrage (optional) umgesetzt würde, müssen aus Kompatibilitätsgründen in beiden Fällen die Funktionen im DBMS für das alte und das neue Datenmodell zur Verfügung stehen bzw. entsprechende *Mappings* zur Verfügung gestellt werden. Somit ist eine schrittweise Annäherung von der optionalen Funktionalität hin zur vollständigen Umstellung ein möglicher Weg. Dieser Weg ist u. U. leichter und gefahrloser zu bewältigen, da die neue Funktion von den EMU-Nutzern getestet werden kann und würde. Auf diesem Weg besteht nicht die Gefahr, dass in dieser Testphase die Funktionalität des Systems zu sehr eingeschränkt ist, da sie vom Entwickler noch nicht berücksichtigt wurde.

Sehr gute Erfahrung mit diesem Weg wurde mit der Umstellung des EMU-Systems mit dem EMU Labeller als primäre Schnittstelle zu EMU mit der primären Schnittstelle EMU Database Tool gemacht. Es sei darauf hingewiesen, dass EMU eine *open source* Software ist und dadurch von den Erfahrungsberichten der Benutzer abhängig ist. Eine *open source* Software kann nicht von Entwicklerseite ausgiebig getestet werden, wie es bei kommerziellen Systemen eigentlich vorauszusetzen ist. Aber Fehler werden selbst in kommerzieller Software häufig erst durch die Benutzung des Endnutzers offenbart.

Für relationale Datenbanken gibt es die freie Softwarebibliothek SQLite (Newman, 2004), die als eine “self-contained, serverless, zero-configuration, transactional SQL database engine” in Hipp, Wyrick and Company, Inc. (2011) beschrieben wird. Diese Bibliothek stellt den brauchbarsten Kandidaten für die Verwendung im EMU-System dar, denn durch die serverlose und dadurch konfigurationsfreie Umsetzung werden vom EMU-Anwender keine neuen Anforderungen gestellt. Aufgrund des anders gearteten Ansatzes des EMU-Systems im Vergleich zu anderer Software auf dem Gebiet, der ein paar Konfigurationsaufgaben durch den Nutzer erfordert, wird die Benutzung des System im Gegensatz zu anderen Softwaresystemen in der Phonetik als schwieriger eingeschätzt (vgl. S. 8) und daher zugunsten leichter verwendbarer Software darauf verzichtet. Das Übertragen weiterer Konfigurations- und Wartungsarbeiten an den Nutzer würde die Einschätzung der hohen Schwierigkeit vermutlich verstärken.

Notwendige Anpassungen am Datenbanktemplate Neben internem Datenbankschema und Datenbankzustand geht mit der neuen Modellierung auch die Notwendigkeit leichter Modifikationen am externen Datenbankschema einher.

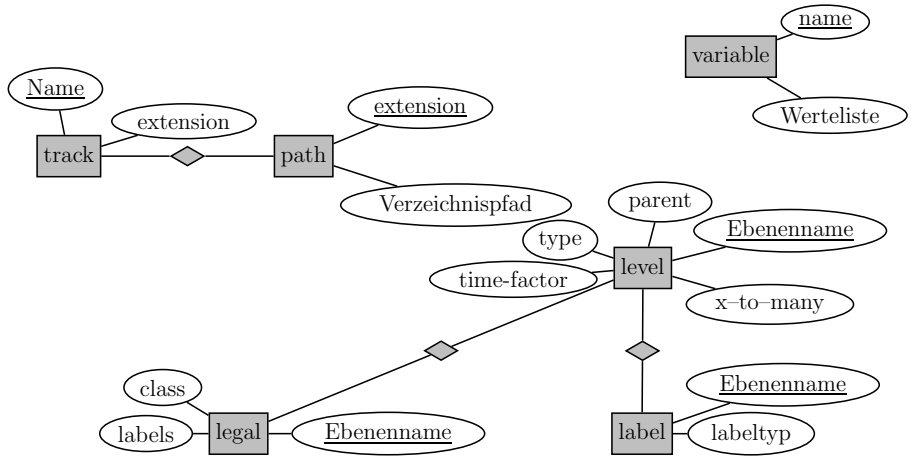


Abbildung 8.5.: ER modelliertes Templateschema für das EMU-System im relationalen Ansatz analog zu Abbildung 3.7.

Abbildung 8.5 zeigt ein mögliches Templateschemadiagramm, das für die Verwendung der relationalen Datenbank angepasst ist. Hier werden genauso wie im aktuell verwendeten Templateschema in Abbildung 3.7 **level**- und **track**-Entitätstypen bereitgestellt. Der **level**-Entitätstyp hat jedoch keine Verbindung mehr mit dem **path**-Entitätstypen, aufgrund der alternativen Datenspeicherung in einer relationalen Datenbank. Aus gleichem Grund und dem Fehlen des **HIERARCHY**-Entitätstypen im internen Schema ist der Entitätstyp **primary** überflüssig. Weiterhin fehlt der **labfile**- und **hlb**-Entitätstyp, da die Art der Speicherung für alle Ebenentypen gleichermaßen stattfindet, nämlich wiederum in den relationalen Tabellen der Datenbank. Aufgrund des fehlenden Entitätstyps **labfile**, der über die Attribute **type** intervall-zeitgebundene Ebenen von event-zeitgebundenen Ebenen unterscheidbar machte und **time-factor** sind die beiden Attribute nun Attribute des **level**-Entitätstyps.

Durch die Änderungen im Templateschema müsste die EMU-Applikation `GTed` entsprechend angepasst werden, da sie, wie aus Kapitel 4.3 hervorgeht, direkt anhand des erstellten Schemadiagrammes für das aktuelle EMU-Modell entworfen wurde. Das Schema in Abbildung 8.5 könnte wiederum als Grundlage für die Anpassung Verwendung finden.

Fazit zum Ausblick Zusammenfassend wird der Ansatz der Verwendung einer relationalen Datenbank im EMU Speech Database System nicht nur als bewährt, sondern auch als machbar eingestuft. Der Vorschlag sollte in Zukunft umgesetzt werden, um mit bereits vorhandener Datenbanktechnologie:

- die Daten wirklich vom Nutzer abschirmen zu können,
- den Nutzern die Möglichkeit der Abfrage mit regulären Ausdrücken zu bieten,
- den Nutzern eine zeiteffiziente Abfrage auch für sehr große Datenbanken zu bieten,
- den Nutzern zeiteffiziente Abfragen von unter Praat etikettierten Sprachkorpora zu ermöglichen, indem eine Konvertierung der Praat-TextGrids in das EMU-Format über die EMU–Praat-Schnittstellen genutzt wird (vgl. Kapitel 6).

9. Zusammenfassendes Fazit

Das EMU Speech Database System konnte in dieser Dissertation zu einem dokumentierten, funktional erweiterten, leicht handhabbaren Werkzeug für Benutzer mit wenig Informatikkenntnissen und als verwendbare mögliche Instanz in den Verarbeitungsketten der Forscher weiterentwickelt werden. In dieser Instanz können durch die Weiterentwicklung Signaldaten unter Verwendung einer grafischen Benutzerschnittstelle zugeschnitten werden und Etikettierungsstrukturen anhand von konzeptuellen Modellen mit vorgeschlagener Notation konzipiert werden. Weiterhin können EMU-Datenbankschemata anhand des erzeugten konzeptuellen Templateschemas konzipiert und schließlich auch entsprechend des Schemas grafisch deklariert werden. Die Datenbanketikettierung kann mit einem neuen Etikettierungstool mit zoombaren Sonagramm und diversen Funktionen, u. a. mit ähnlichen Methoden wie in Praat und auch über direktes Skripten oder über die indirekte Stapelverarbeitung von Skripten, erfolgen. Außerdem kann die Datenbank grafisch abgefragt und das Ergebnis direkt im Etikettierungstool geöffnet werden. Die fertige Datenbank kann weiterhin noch über das Internet zur Verfügung gestellt bzw. leicht in andere System exportiert werden. Es zeigt sich also, dass die Weiterentwicklung im Rahmen dieser Dissertation an jedem Punkt in der Verarbeitungskette der Daten angesetzt hat. Jeder einzelne Punkt wurde auch dokumentiert.

Das System hat auch nach der Weiterentwicklung noch Schwächen, zu denen jedoch in dieser Arbeit zum Teil *workarounds*[1] gezeigt wurden. Weiterhin belegt diese Arbeit, dass es sinnvoll ist, dass der Programmentwickler auch der Programmnutzer ist. Auf diese Weise kann der Anspruch des Benutzers direkt in der Software umgesetzt und für andere Forscher zur Verfügung gestellt werden, denn in dieser Kombination hat der Entwickler selbst einen fachlichen Anspruch an das Programm. Workarounds sollten daher in Zukunft weiterhin über die Mithilfe der Nutzer durch Lösungen ersetzt werden.

[1] Lösungen zum Umgehen eines Problems

Als Ausblick wurde beschrieben, dass eine relationale Strukturierung der Daten im EMU-System, bezogen auf die Geschwindigkeit, die Basis für die effiziente Abfrage mithilfe der Standard Query Language (SQL) bietet. Dieser Ansatz wurde auch für das EMU-System in früheren Jahren in Betracht gezogen. Die Arbeit wurde neu aufgegriffen und bezüglich der Umsetzbarkeit evaluiert. Das bereits vorgeschlagene relationale Datenbankschema, die damit verbundenen Unterschiede in den möglichen Abfragen sowie die Methode zur Überführung einer Datenbank wurden dargestellt. Im Ergebnis zeigte sich der Ansatz als umsetzbar und erstrebenswert. Einhergehend mit einer Diskussion der Alternativen müssen sowohl in der Überführungsmethode als auch im Schema Veränderungen vorgenommen werden. Einige Lösungsansätze wurden hierfür bereits in dieser Arbeit vorgeschlagen. Ein Ziel für EMU sollte es somit sein, Etikettierungen in einer relationalen Struktur ohne den Verlust von Abfragemöglichkeiten so aufzubereiten, dass sie effizient abgefragt werden können. Weiterhin kann die Verwendung von SQL die aufgezeigten Schwächen der EMU Query Language beheben. Durch die Komplexität der Etikettierungsstruktur ist dies nicht trivial, aber lösbar.

In der phonetischen aber auch in der allgemeinen linguistischen Forschung gibt es eine Vielzahl an Programmen und Korpora. Aus der eigenen Arbeit und der Zusammenarbeit mit anderen Forschern ist bekannt, dass verschiedene Programme innerhalb der Beantwortung einer Forschungsfrage verwendet werden. Ein gemeinsames Ziel der Entwickler der einzelnen Programme hat sich in den letzten Jahren herauskristallisiert: der Wunsch zur Interoperabilität der Programme. Zur Zeit werden Bemühungen angestellt, ein geeignetes gemeinschaftliches Format für Etikettierungsdateien zu entwerfen. Zum derzeitigen Stand wird das Annotation Graph Modell (Bird und Liberman, 2001) in Betracht gezogen und soll in Zukunft nach einer von Alan Black (Verantwortlich für Festvox (Black, 2010)) vorbereiteten Vorlage u. a. von EMU und Praat als Vertreter der phonetischen Forschung und Elan als Vertreter der linguistischen Forschung getestet werden (Hawkins, 2011).

Zur Interoperabilität konnte diese Arbeit dazu beitragen, den Anspruch des Systems zu erfüllen, mit vielen Sprachverarbeitungsprojekten arbeiten zu können, auch wenn es kein einheitliches Etikettierungsformat gibt. Durch die Weiterentwicklung gibt es im EMU-System eine beidseitige Kommunikation zwischen EMU und Praat, sowie die Importmöglichkeit von Daten aus u. a. Praat, WaveSurfer, Articulate Assistant und vorhandenen Sprachkorpora, wie dem Kiel Corpus. Es wurden Algorithmen konzeptuell dargestellt, mit denen versucht wird, in der Kommunikation oder dem Import so wenig Information wie möglich zu verlieren. Für den Import

stellten sich hierbei keine größeren Datenverluste heraus, da alle Etikettierungen aus anderen Programmen von der Struktur her auf das EMU-Etikettierungsmodell abbildbar sind. Auch das gänzliche Fehlen einer Struktur bzw. eines Schemas stellt für den Import kein Problem dar, wie gezeigt werden konnte. Beim Export von Etikettierungen in ein anderes Programm können aufgrund der umfangreichen und einzigartigen Etikettierungsstrukturmöglichkeiten im EMU-System Verluste nicht ausgeschlossen werden. Dafür bietet EMU nunmehr jedoch direkte Schnittstellen an, die nur gegenseitig abbildbare Etikettierungen an andere Programme übergeben und Änderungen wieder in den bestehenden Datenbankzustand einfügen. Indirekter Import und Export werden datenbankbasiert durchgeführt, d. h. über den gesamten Datenbankumfang. Der Benutzer kann hier jedoch Einschränkungen machen. Bei allen Importmöglichkeiten ist der Benutzer nach der Auswahl der zu importierenden Daten niemals mehr mit zu suchenden oder zu bearbeitenden Dateien direkt konfrontiert. Beim Export beginnt das dateibasierte Arbeiten erst im Zielprogramm des Exports.

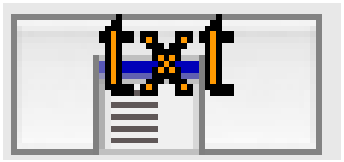
Für den Import des Kiel Corpus wurden zunächst die Etikettierungsstrukturen im Korpus selbst anhand der Speicherstrukturen in den Etikettierungsdateien gezeigt. Außerdem wurde die PROLAB Notation des Kieler Intonationsmodells erläutert, da es in manchen Teilen des Korpus in der Etikettierung enthalten ist. Die Struktur zeigte sich auf den ersten Blick übersichtlich, wobei die Zeichenwahl als wenig intuitiv eingeschätzt wurde. Später zeigte sich jedoch wieder der Nachteil des dateibasierten Arbeitens in Form von Inkonsistenzen innerhalb der Dateien, die den Import erschwerten. Es wurde eine Lösung vorgestellt, die dieses Problem für die meisten Sprachdaten lösen konnte. Für den Import wurde, ein umfangreiches, zwar nicht jede Information im Kiel Corpus abdeckendes, aber dafür neue Informationen hinzufügendes EMU-Etikettierungsschema und ein EMU-Datenbanktemplate entworfen, auf dessen Grundlage die Etikettierungen als Datenbankausprägung strukturiert und in die Datenbank eingepflegt wurden. Dadurch entstand eine große Datenbank, die innerhalb des Systems weiterverarbeitet werden kann. Hierzu gehören die Darstellungen, Modifikationen, Abfragen, Signalableitungen und diverse Stapelverarbeitungen sowie die Bereitstellung der Kiel Corpus Datenbank im World Wide Web. Aufgrund dieser Arbeit steht das Kiel Corpus in ganz neuer und leicht verarbeitbarer Form für die Forschung zur Verfügung.

Insgesamt ist die Kombination der systemeigenen Eigenschaften von EMU mit der Interoperabilität zu anderen Programmen exakt die Schnittstelle, die sich jeder Forscher im Bereich der Phonetik bzw. Linguistik wünscht. Die beschriebenen Schwä-

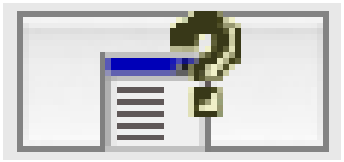
chen sind nicht unbedingt einfach und schnell zu beheben, dennoch ist das EMU Speech Database System es in seiner Gesamtheit wert, die Arbeit weiterhin fortzuführen. Daher kann nur zu hoffen sein, dass die Funktionalität, die EMU bieten kann, auch in Zukunft in der phonetischen Forschung nutzbar bleibt. Sollte das EMU-System nicht mehr parallel zu anderen Programmen weiterentwickelt werden, würde der an EMU gewöhnte Forscher die ungewöhnlichere, aber sehr effiziente Verwendung der Software schmerzlich vermissen[2]. Hohe Nutzerzahlen und Anfragen zu Weiterentwicklungen stellen eine Grundlage für den Fortbestand der Software dar. Daher sind weitere Veröffentlichungen sinnvoll, die das Augenmerk der phonetischen bzw. linguistischen Nutzer gezielter auf EMU lenken.

Aus Abbildung 9.1 geht abschließend die Hoffnung hervor, dass mithilfe dieser Arbeit (a) wenig Fragen (b) bezüglich der Nutzbarkeit und der besonderen Funktionalität des EMU-Systems offen geblieben sind. Auch zu hoffen bleibt, dass anders als in (c) die geöffnete Tür für die Verwendung des Systems in der phonetischen Forschung und für die Weiterentwicklung nicht zum Verlassen sondern zum Betreten einlädt.

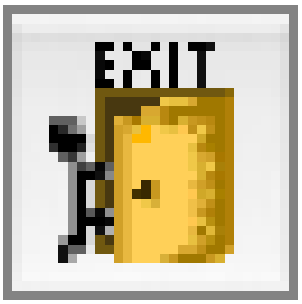
[2] aus der persönlichen Kommunikation mit EMU-Anwendern



(a) Schnittstelle zum textbasierten Template Editor



(b) Schnittstelle zur Dokumentation



(c) “Icons or images should look like the real-world objects they represent.” (Galitz 2007, S. 54)

Abbildung 9.1.: Eigens für den GTemplate Editor entworfene und als Pixelgrafiken manuell angefertigte Icons. Angefertigt, um das *Familiarity*-Prinzip nach Galitz (2007) umzusetzen.

Literatur

Allen, J. und U. Consortium (2007). *Unicode standard, version 5.0*. Addison-Wesley.

Ambrazaitis, G. und T. John (2004). On the allophonic behaviour of german /x/ vs /k/ - an epg investigation. In: J. Harrington und C. Mooshammer (Hrsg.), *Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel*, Volume 36, pp. 1–14. IPDS.

Ananthakrishnan, G. und O. Engwall (2011). Mapping between acoustic and articulatory gestures. *Speech Communication 53*(4), 567–589.

Articulate Instruments Ltd. (2010). *Articulate Assistant User Guide: Version 1.18*. Edinburgh, UK: Articulate Instruments Ltd.

AtlanticSX LLC und Goldwave, Inc. (2007). Goldwave - der ultimative wav und mp3 editor. `http://www.goldwave.de`. Abgerufen am 06.03.2011.

Bachman, C. (1973). The Programmer as Navigator: 1973 ACM Turing Award Lecture. *Communications of the ACM 16*(11), 653–658.

Barry, W., B. Andreeva, M. Russo, S. Dimitrova und T. Kostadinova (2003). Do rhythm measures tell us anything about language type. In: *Proceedings of the 15th International Congress of Phonetic Sciences, Barcelona*, pp. 2693–2696.

Batliner, A., B. Schuller, D. Seppi, S. Steidl, L. Devillers, L. Vidrascu, T. Vogt, V. Aharonson und N. Amir (2011). The Automatic Recognition of Emotions in Speech. *Emotion-Oriented Systems 2*, 71–99.

Beckman, M. und J. Hirschberg (1994). The ToBI annotation conventions. `http://www.ling.ohio-state.edu/~tobi/ame_tobi/annotation_conventions.html`. Abgerufen am 06.03.2011.

Biethahn, J., H. Mucksch und W. Ruf (2007). *Ganzheitliches Informationsmanagement 2: Entwicklungsmanagement*. Oldenbourg Wissenschaftsverlag.

Bird, S. und M. Liberman (2001). A formal framework for linguistic annotation. *Speech Communication 33*(1-2), 23–60.

Black, A. (2010). Festvox. `http://festvox.org`. Abgerufen am 07.03.2011.

Boersma, P. (2002). Praat, a system for doing phonetics by computer. *Glot international 5*(9/10), 341–345.

Boersma, P. und D. Weenink (2006). Sendpraat: sending messages to a praat shell program. (Code Version February 17, 2006)[Computer Program], `http://www.fon.hum.uva.nl/praat/sendpraat.html`. Abgerufen am 26.02.2011.

Bombien, L., S. Cassidy, J. Harrington, T. John und S. Palethorpe (2006). Recent developments in the Emu speech database system. In: *Proceedings of the 11th SST Conference Auckland*, pp. 313–316.

Bombien, L., C. Mooshammer, P. Hoole, T. Rathcke und B. Kühnert (2007). Articulatory strengthening in initial german /kl/ clusters under prosodic variation. In: *Proceedings of the 16th International Conference of Phonetic Sciences*, pp. 457–460.

Borden, G. J., K. S. Harris und L. J. Raphael (2003). *Speech Science Primer: Physiology, Acoustics, and Perception of Speech* (fourth ed.). Lippincott Williams & Wilkins.

Brugman, H. und A. Russel (2004). Annotating multimedia/multi-modal resources with ELAN. In: *Proceedings of the Fourth International Conference on Language Resources and Evaluation*, pp. 2065–2068.

Cassidy, S. (1999a). Compiling multi-tiered speech databases into the relational model: Experiments with the Emu system. In: *Proceedings of the 6th European Conference on Speech Communication and Technology*, pp. 2239–2242.

Cassidy, S. (1999b). Emu and MySQL. `http://emu.sourceforge.net/old/relational.shtml`. Abgerufen am 26.02.2011.

Cassidy, S. (2004). The Emu Speech Database System. `http://emu.sourceforge.net/new_manual/index.html`. Abgerufen am 19.09.2011.

Cassidy, S. (2011a). Chapter 9. Simple Signal File Format (SSFF). `http://emu.sourceforge.net/manual/chap.ssff.html`. Abgerufen am 19.02.2011.

Cassidy, S. (2011b). Emu help. `http://emu.sourceforge.net/new_manual/ch07s02.html`. Abgerufen am 17.02.2011.

Cassidy, S. und S. Bird (2000). Querying databases of annotated speech. In: *Proceedings of the 11th Australasian Database Conference*, pp. 12–20. IEEE.

Cassidy, S. und J. Harrington (1996). Emu: An enhanced hierarchical speech data management system. In: *Proceedings of the Sixth Australian International Conference on Speech Science and Technology*, pp. 361–366.

Cassidy, S. und J. Harrington (2001). Multi-level annotation in the Emu speech database management system. *Speech Communication 33*(1-2), 61 – 77.

Cattell, R. G. G. und T. Atwood (1993). *The Object database standard, ODMG-93*. Morgan Kaufmann San Mateo, CA.

Chen, P. (1976). The entity-relationship model–toward a unified view of data. *ACM Transactions on Database Systems (TODS) 1*(1), 9–36.

Chomsky, N. (1957). *Syntactic structures*. Mouton.

Chomsky, N. (1965). *Aspects of the theory of syntax*. Cambridge: M.I.T. Press.

CODASYL (1971). Data Base Task Group Report. Technischer Bericht, Association for Computing Machinery.

Codd, E. F. (1970, June). A relational model of data for large shared data banks. *Communications of the ACM 13*, 377–387.

Covi, L. M. und M. S. Ackerman (1995). Such easy-to-use systems!: How organizations shape the design and use of online help systems. In: *Proceedings of conference on Organizational computing systems*, New York, NY, USA, pp. 280–288. ACM.

Date, C. und H. Darwen (1998). *SQL-der Standard*. Addison-Wesley.

de Saussure, F. (1916). *Cours de linguistique général*. Lausanne/Paris: Payot.

de Saussure, F. (2001). *Grundfragen der allgemeinen Sprachwissenschaft*. Walter de Gruyter. Fr. Original 1916.

Doragh, P. (1994, August 23). Method and apparatus for processing multiple file system server requests in a data processing network. US Patent 5,341,499.

Draxler, C. (2006). *Korpusbasierte Sprachverarbeitung*. Tübingen: Gunter Narr Verlag.

Duquette, W. H. (2005). Tcllib - tcl standard library: snit. `http://www.wjduquette.com/snit/`. Abgerufen am 26.08.2011.

Ebert, C. und C. Ebert (2010). Methoden. In: *Computerlinguistik und Sprachtechnologie*, Kapitel 3, pp. 169–479. Springer.

Elmasri, R. und S. Navathe (2009). *Grundlagen von Datenbanksystemen*. Pearson Education.

EMU Developers (2000). Feature Request Tracker. `http://sourceforge.net/tracker/?group_id=16757&atid=366757`. Abgerufen am 01.03.2011.

EMU Developers (2011a). SourceForge.net Repository - [emu] Index of /. `http://emu.cvs.sourceforge.net`. Abgerufen am 17.02.2011.

EMU Developers (2011b). Ssff2wav. `http://www.ipds.uni-kiel.de/forschung/EMUadds.en.htm#4%20ssff2wav`. Abgerufen am 06.03.2011.

European consortium (1993). Eur-accor. `http://www.cstr.ed.ac.uk/research/projects/artic/accor.html`.

Galitz, W. (2007). *The essential guide to user interface design: an introduction to GUI design principles and techniques*. Wiley.

Garofolo, J., L. Lamel, W. Fisher, J. Fiscus, D. Pallett und N. Dahlgren (1993). *Darpa Timit: Acoustic-phonetic Continuous Speech Corps CD-ROM*. US Dept. of Commerce, National Institute of Standards and Technology.

Geeknet, Inc. (2011). SourceForge. `http://sourceforge.net`. Abgerufen am 06.03.2011.

Gibbon, D. (1997). SAMPA-D-VMlex, 1997. `http://coral.lili.uni-bielefeld.de/Documents/sampa-d-vmlex.html`. Abgerufen 08.03.2011.

Goldfarb, C. und Y. Rubinsky (1990). *The SGML handbook*. USA: Oxford University Press.

Gordon, M. und A. Applebaum (2010). Acoustic correlates of stress in turkish kabardian. *Journal of the International Phonetic Association 40*(01), 35–58.

Gordon, M. und P. Ladefoged (2001). Phonation types: a cross-linguistic overview. *Journal of Phonetics 29*(4), 383–406.

Haigh, T. (2006). A veritable bucket of facts - origins of the data base management system. *SIGMOD Rec. 35*(2), 33–49.

Hakkarainen, H. (1995). *Phonetik des Deutschen*. Wilhelm Fink Verlag.

Harrington, J. (2010a). *Phonetic Analysis of Speech Corpora*. Wiley-Blackwell.

Harrington, J. (2010b). Phonetic analysis of speech corpora - appendices - appendix b: Some notes on emu-tcl. `http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/appendices/AppendixB.pdf`. Abgerufen am 23.09.2011.

Harrington, J., S. Cassidy, J. Fletcher und A. McVeigh (1993). The mu+ system for corpus based speech research. *Computer Speech and Language 7*, 305–305.

Harrington, J., S. Cassidy, T. John und M. Scheffers (2003). Building an interface between EMU and Praat: a modular approach to speech database analysis. In: *Proceedings of International Congress of Phonetic SciencesI*.

Hawkins, S. (2011). Stelaris meeting - minutes of final plenary session. Technischer Bericht, University of Pennsylvania.

Hedtstück, U. (2007). *Einführung in die theoretische Informatik: formale Sprachen und Automatentheorie*. Oldenbourg Wissenschaftsverlag.

Heracleous, P., P. Badin, G. Bailly und N. Hagita (2011). A pilot study on augmented speech communication based on Electro-magnetic articulography. *Pattern Recognition Letters 32*(8), 1119–1125.

Hipp, Wyrick and Company, Inc. (2011). Sqlite. `http://www.sqlite.org`. Abgerufen am 27.02.2011.

Hirschfeld, U. (2003). Deutsch. In: U. Hirschfeld, H. Kelz, und U. Müller (Hrsg.), *Phonetik international: Von Afrikaans bis Zulu. Kontrastive Studien für Deutsch als Fremdsprache*. Waldsteinberg: Heidrun Popp.

Hooper, J. (1972). The syllable in phonological theory. *Language 48*(3), 525–540.

IBM Corp. and Microsoft Corp. (1991). *Multimedia Programming Interface and Data Specifications 1.0*. IBM Corporation and Microsoft Corporation.

IPA (1999). *Handbook of the International Phonetic Association. A guide to the use of the international phonetic alphabet*. Cambridge University Press.

IPDS (1994). *The Kiel Corpus of Read Speech*, Volume 1, CD-ROM#1. Kiel: Institut für Phonetik und digitale Sprachverarbeitung.

IPDS (1995). *The Kiel Corpus of Spontaneous Speech*, Volume 1, CD-ROM#2. Kiel: Institut für Phonetik und digitale Sprachverarbeitung.

IPDS (1996). *The Kiel Corpus of Spontaneous Speech*, Volume 2, CD-ROM#3. Kiel: Institut für Phonetik und digitale Sprachverarbeitung.

IPDS (1997a). *The Kiel Corpus of Spontaneous Speech*, Volume 3, CD-ROM#4. Kiel: Institut für Phonetik und digitale Sprachverarbeitung.

IPDS (1997b). xassp user's manual (advanced speech signal processor under the x window system). In: A. P. Simpson, K. J. Kohler, und T. Rettstadt (Hrsg.), *The Kiel Corpus of Read/Spontaneous Speech – Acoustic data base, processing tools and analysis results*, Volume 32 of *Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel (AIPUK)*, pp. 31–115. IPDS.

IPDS (2002a). The Kiel Corpus of Read Speech, Vol. I. `http://www.ipds.uni-kiel.de/publikationen/kcrsp.de.html`. Abgerufen am 06.03.2011.

IPDS (2002b). The Kiel Corpus of Spontaneous Speech, Vol. I-III. `http://www.ipds.uni-kiel.de/publikationen/kcssp.de.html`. Abgerufen am 06.03.2011.

ISO/IEC 14977 (1996). Information technology — syntactic metalanguage — extended bnf (1st edition).

ISO/IEC 646 (1991). Information technology - ISO 7-bit coded character set for information interchange.

John, T. (2003a). Dokumentation xassp2emu. `http://www.ipds.uni-kiel.de/forschung/infos/xassp2emuHelp.de.htm`. Abgerufen am 21.09.2011.

John, T. (2003b). EMU: Erzeugen eines query-Strings mit dem queryGUI. http://www.ipds.uni-kiel.de/forschung/infos/EMU_queryGUI.pdf. Abgerufen am 14.09.2011.

John, T. (2003c). Informationen zu labConvert. http://www.ipds.uni-kiel.de/forschung/infos/convert.de.htm. Abgerufen am 21.09.2011.

John, T. (2003d). praat2emu. http://www.ipds.uni-kiel.de/forschung/infos/praat2emuHelp.en.htm. Abgerufen am 17.02.2011.

John, T. (2004). *Eine akustische Analyse der Lenis/Fortis-Opposition in Varietäten des Sächsischen*. Kiel: Magisterarbeit am Institut für Phonetik und digitale Sprachverarbeitung der Christian-Albrechts-Universität.

John, T. (2010a). Build annotation over an entire database. http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid28.swf. Abgerufen am 21.09.2011.

John, T. (2010b). Build annotation structures semi-automatically. http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid27.swf. Abgerufen am 21.09.2011.

John, T. (2010c). Calculate formant frequencies. http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid14.swf. Abgerufen am 21.09.2011.

John, T. (2010d). Convert TextGrid file into Emu format. http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid06.swf. Abgerufen am 21.09.2011.

John, T. (2010e). Create new template file. http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid09.swf. Abgerufen am 21.09.2011.

John, T. (2010f). Enter the location of the template into emu. http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid10.swf. Abgerufen am 21.09.2011.

John, T. (2010g). A first look at the template file. http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid07.swf. Abgerufen am 21.09.2011.

John, T. (2010h). Graphical user interface to the query language. http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid26.swf. Abgerufen am 21.09.2011.

John, T. (2010i). Install database. `http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid01.swf`.

John, T. (2010j). Open utterance in labeller. `http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid02.swf`. Abgerufen am 21.09.2011.

John, T. (2010k). Praat annotation. `http://www.phonetik.uni-muenchen.de/forschung/publikationen/harrington/pasc/videotut-files/vid05.swf`. Abgerufen am 21.09.2011.

John, T. (2010l). Textbook website: Phonetic analysis of speech corpora. `http://www.wiley.com/go/harrington`. Abgerufen am 06.03.2011.

John, T. (2010m). Video tutorials: Phonetic analysis of speech corpora. `http://www.wiley.com/go/harrington`. Abgerufen am 06.03.2011.

John, T. und L. Bombien (iE). EMU. In: J. Durand, U. Gut, und G. Kristoffersen (Hrsg.), *Handbook on Corpus Phonology*. Oxford University Press.

John, T. und J. Harrington (2007). Temporal alignment of creaky voice in neutralised realisations of an underlying, post-nasal voicing contrast in German. In: *INTERSPEECH-2007*, pp. 986–989.

John Wiley & Sons, I. (2011). Wiley::wiley-blackwell. `http://eu.wiley.com/WileyCDA/Brand/id-35.html`. Abgerufen 06.03.2011.

Kemper, A. und A. Eickler (2006). *Datenbanksysteme: Eine Einführung*. Oldenbourg Wissenschaftsverlag.

Kohler, K. (1991). A model of German intonation. In: K. J. Kohler (Hrsg.), *Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel (AIPUK)*, Volume 25, pp. 295–360. IPDS.

Kohler, K. (1992a). Automatische Generierung der kanonischen Transkription und des Aussprachlexikons. In: K. Kohler (Hrsg.), *Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel (AIPUK)*, Volume 26, pp. 175–196. Kiel: IPDS.

Kohler, K. (1992b). Phonetisch-akustische Datenbasis des Hochdeutschen: Kieler Arbeiten zu den PHONDAT-Projekten 1989–1992. In: K. Kohler (Hrsg.), *Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel (AIPUK)*, Volume 26. Kiel: IPDS.

Kohler, K., G. Lex, M. Pätzold, M. Scheffers, A. Simpson und W. Thon (1994). *Handbuch zur Datenaufnahme und Transliteration in TP 14 von VERBMOBIL–3.0*, Volume 11. Verbmobil Technisches Dokument.

Kohler, K., M. Pätzold und A. Simpson (1995). From Scenario to Segment: The Controlled Elicitation, Transcription, Segmentation, and Labelling of Spontaneous Speech. In: K. J. Kohler (Hrsg.), *Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel (AIPUK)*, Volume 29. IPDS.

Kohler, K., M. Pätzold und A. Simpson (1997). From the acoustic data collection to a labelled speech data bank of spoken Standard German. *Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel (AIPUK) 32*, 1–29.

Kohler, K. J. (1995). PROLAB-the Kiel system of prosodic labelling. In: *Proceedings of International Congress of Phonetic Sciences*, Volume 3, pp. 162–165.

Lakhani, K. R. und E. von Hippel (2003). How open source software works: free user-to-user assistance. *Research Policy 32*(6), 923 – 943.

McGee, W. C. (1977, June). The information management system ims/vs: part v: transaction processing facilities. *IBM Syst. J. 16*, 148–168.

McLeod, S., A. Roberts und J. Sita (2003). The difference between /s/ and /z/: More than +/- voice. In: *American Speech-Language-Hearing Association Convention*.

Mertens, P., F. Bodendorf, W. König, A. Picot, M. Schumann und T. Hess (2004). *Grundzüge der Wirtschaftsinformatik*. Springer Verlag.

Meyer, B., T. Brand und B. Kollmeier (2011). Effect of speech-intrinsic variations on human and automatic recognition of spoken phonemes. *Acoustical Society of America Journal 129*, 388.

Microsoft (1994). *New Multimedia Data Types and Data Techniques*. Microsoft.

Microsoft (2001). *Update: Multiple Channel Audio Data and WAVE Files*. Microsoft.

Millar, J., P. Dermody, J. Harrington und J. Vonwiller (1997). Spoken language resources for australian speech technology. In: *Journal Of Electrical and Electronic Engineers Australia*, Volume 1, pp. 13–23.

Millar, J., J. Vonwiller, J. Harrington und P. Dermody (1994). The Australian National Database of Spoken Language. In: *Acoustics, Speech, and Signal Processing*, pp. 97–I100. IEEE.

Newman, C. (2004). *SQLite (Developer's Library)*. Indianapolis, USA: Sams.

Oracle Corporation (2011). Mysql workbench. `http://www.mysql.com/products/workbench`. Abgerufen am 06.03.2011.

Ousterhout, J. (1994). Tcl and the Tk Toolkit. Professional Computing Series.

Pätzold, M. (1997). KielDat-data bank utilities for the Kiel Corpus. *The Kiel Corpus of Read/Spontaneous Speech: acoustic data base, processing tools, and analysis results 32*, 117–126.

Peters, B. (1999). Prototypische Intonationsmuster in deutscher Lese- und Spontansprache. In: K. Kohler (Hrsg.), *Phrase-level Phonetics and Phonology of German*, Volume 34 of *Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel (AIPUK)*. Kiel: IPDS.

Peters, B. (2006). *Form und Funktion prosodischer Grenzen im Gespräch. Dissertation*. Kiel: IPDS.

Peters, B. und K. Kohler (2004). Trainingsmaterialien zur prosodischen Etikettierung mit dem Kieler Intonationsmodell KIM. `www.ipds.uni-kiel.de/kjk/pub_exx/bpkk2004_1/TrainerA4.pdf`. Abgerufen am 17.02.2011.

Pierrehumbert, J. und I. U. L. Club (1980). *The phonology and phonetics of English intonation*. MIT.

PKWARE (2004). Zip file format specication. version 6.2.0. `http://www.pkware.com/products/enterprise/white_papers/appnote.txt`. Abgerufen am 06.10.2011.

Prinz, M. und R. Wiese (1990, 2011/03/17). Ein nicht-lineares Modell der Graphem-Phonem-Korrespondenz. *Folia Linguistica 24*(1-2), 73–104.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0.

Ramanarayanan, V., E. Bresch, D. Byrd, L. Goldstein und S. Narayanan (2009). Analysis of pausing behavior in spontaneous speech using real-time magnetic resonance imaging of articulation. *The Journal of the Acoustical Society of America 126*(5), 160–165.

Reese, G., R. Yarger und T. King (2002). *MySQL: Einsatz und Programmierung*. O'Reilly Germany.

Russo, M. und W. Barry (2008). Isochrony reconsidered. Objectifying relations between rhythm measures and speech tempo. In: *Proceedings of Speech Prosody*, Volume 4, pp. 419–422.

Saake, G., K. Sattler und A. Heuer (2010). *Datenbanken–Konzepte und Sprachen*. mitp.

Sanders, M., N. Chang, M. Hiss, R. Uchanski und T. Hullar (2011). Temporal binding of auditory and rotational stimuli. *Experimental Brain Research 4*, 539–547.

Schalkwyk, J., J. de Villiers, S. van Vuuren und P. Vermeulen (1997). Cslush: an extensible research environment. In: *Proceedings of Eurospeech ´97*, Rhodes, Greece, pp. 689–692.

Scheffers, M. und L. Bombien (2011, 02). SourceForge.net Repository - [libassp] Index of /. `http://libassp.cvs.sourceforge.net`. Abgerufen am 19.02.2011.

Schultze-Berndt, E. (2006). Linguistic annotation. In: J. Gippert, N. P. Himmelmann, und U. Mosel (Hrsg.), *Essentials of language documentation*, pp. 213–251. Mouton De Gruyter.

Sensimetrics Corporation (2011). Speechstation2. `http://sens.com/speechstation/index.htm`. Abgerufen am 11.03.2011.

Simpson, A. P. (1998). Phonetische Datenbanken des Deutschen in der empirischen Sprachforschung und der phonologischen Theoriebildung. In: K. J. Kohler (Hrsg.), *Arbeitsberichte des Instituts für Phonetik und digitale Sprachverarbeitung der Universität Kiel (AIPUK)*, Volume 33. Kiel: IPDS.

Simpson, I. (2003). The Emu Segmentation Tool. `http://emu.sourceforge.net/manual/ch03s06.html#fig-emusegment-main`. Abgerufen am 06.09.2011.

Sjölander, K. (2004). The Snack sound toolkit. `http://www.speech.kth.se/snack`. Abgerufen am 07.03.2011.

Sjölander, K. und J. Beskow (2000). WaveSurfer - an open source speech tool. In: *Sixth International Conference on Spoken Language Processing*, Volume 4, pp. 464–467.

Sjölander, K. und J. Beskow (2010). Wavesurfer. `http://www.speech.kth.se/wavesurfer`. Abgerufen am 25.02.2011.

Stevens, K. (1989). On the quantal nature of speech. *Journal of phonetics 17*(1), 3–45.

TechSmith (2009). Jing. `http://jing.softonic.de`. Abgerufen am 06.03.2011.

The Open Group (2003). The open group base specifications issue 6. Technischer Bericht, The IEEE and The Open Group.

The PHP Group (2011). Php: Hypertext preprocessor. `http://www.php.net/`. Abgerufen am 06.03.2011.

W3C (2011). Cascading style sheets. `http://www.w3.org/Style/CSS/`. Abgerufen am 06.03.2011.

Watson, G. (1989). APS: An environment for acoustic phonetic research. In: *First European Conference on Speech Communication and Technology*.

Wells, J. (1987). Computer-coded phonetic transcription. *Journal of the International Phonetic Association 17*(02), 94–114.

Wells, J. (1997). SAMPA computer readable phonetic alphabet. In: D. Gibbon, R. Moore, und R. Winski (Hrsg.), *Handbook of Standards and Resources for Spoken Language Systems*, Kapitel IV B, pp. 684–732. Berlin and New York: Mouton de Gruyter.

Wiese, R. (2000). *The phonology of German*. Oxford University Press, USA.

Williams, B. (2008). EMU Speech Database System (Review). *Language Documentation & Conservation 2*(1), 166–175.

Wittenburg, P., H. Brugman, A. Russel, A. Klassmann und H. Sloetjes (2006). Elan: a professional framework for multimodality research. In: *Proceedings of Language Resources and Evaluation Conference (LREC)*, Paris, pp. 1556–1559.

Wrench, A. A. (2003). *Articulate Assistant (Version 1.6)*. Edinburgh: Articulate Instruments.

Anhang

A. Abbildungen

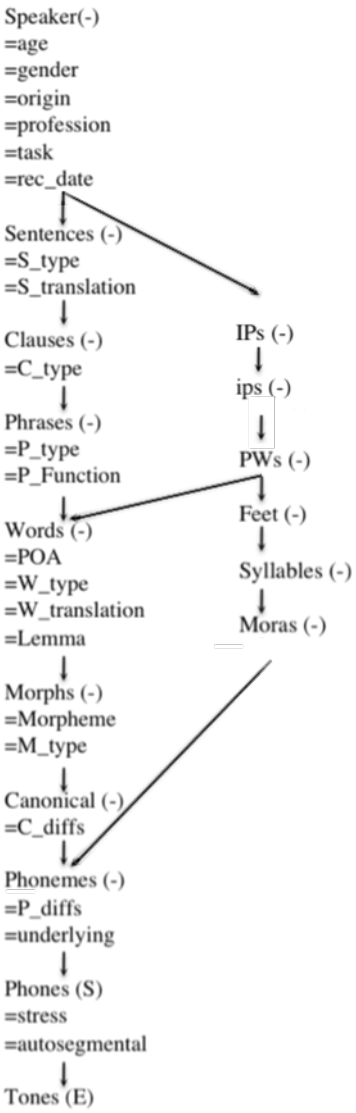


Abbildung A.1.: Längste bekannte modellierte Kette an Ebenen in einem EMU-Etikettierungsschema, modelliert in John und Bombien (iE).

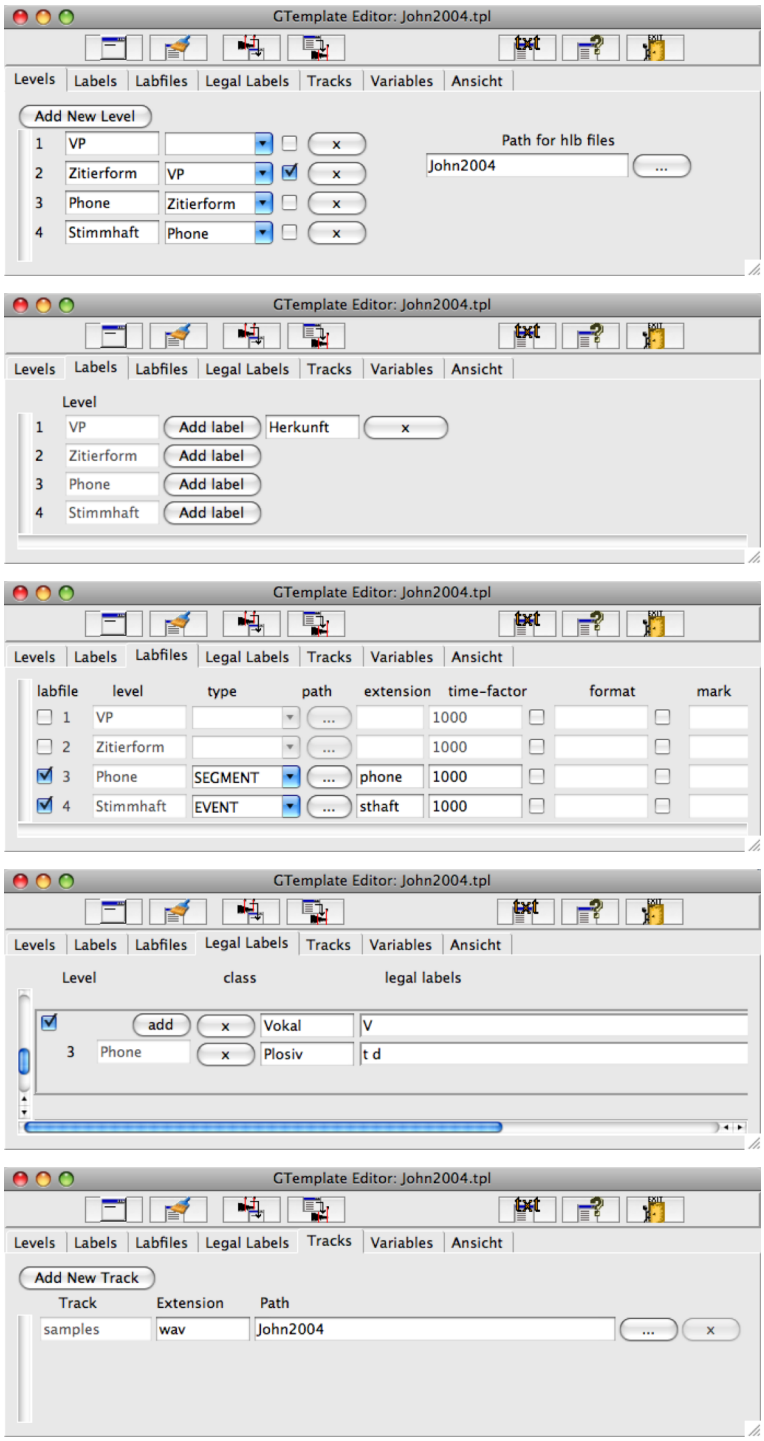


Abbildung A.2.: Datenbanktemplate aus John (2004) in der grafischen Oberfläche der EMU-Applikation GTemplate Editor (GTed) für jede Registerkarte zum Definieren des Etikettierungsschemas, der Signale und der Dateipfade (hlb in der ersten Registerkarte, *wav*-Dateien in der letzten).

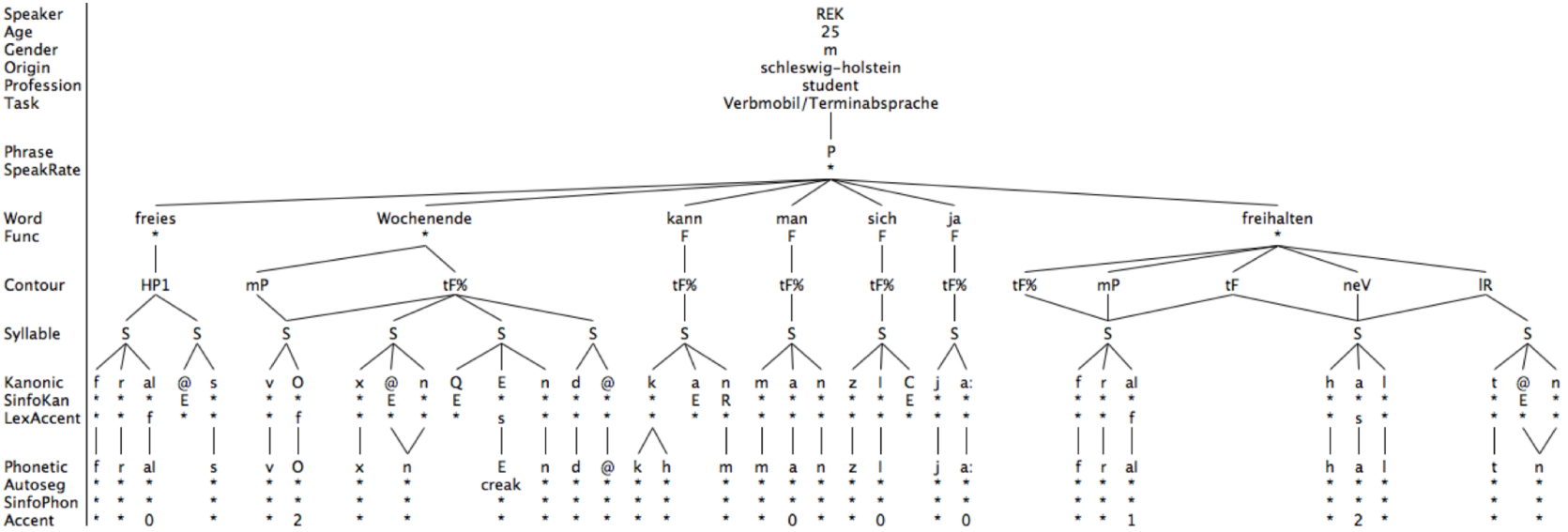


Abbildung A.3.: Ausschnitt der Kiel Corpus Äußerung g112a012 mit einem zweisilbigen phraseninitalen Wort, das in der Zitierform ein [ə] enthält, das getilgt ist, jedoch in der Zitierform nicht vor einem alveolaren Nasal vorkommt.

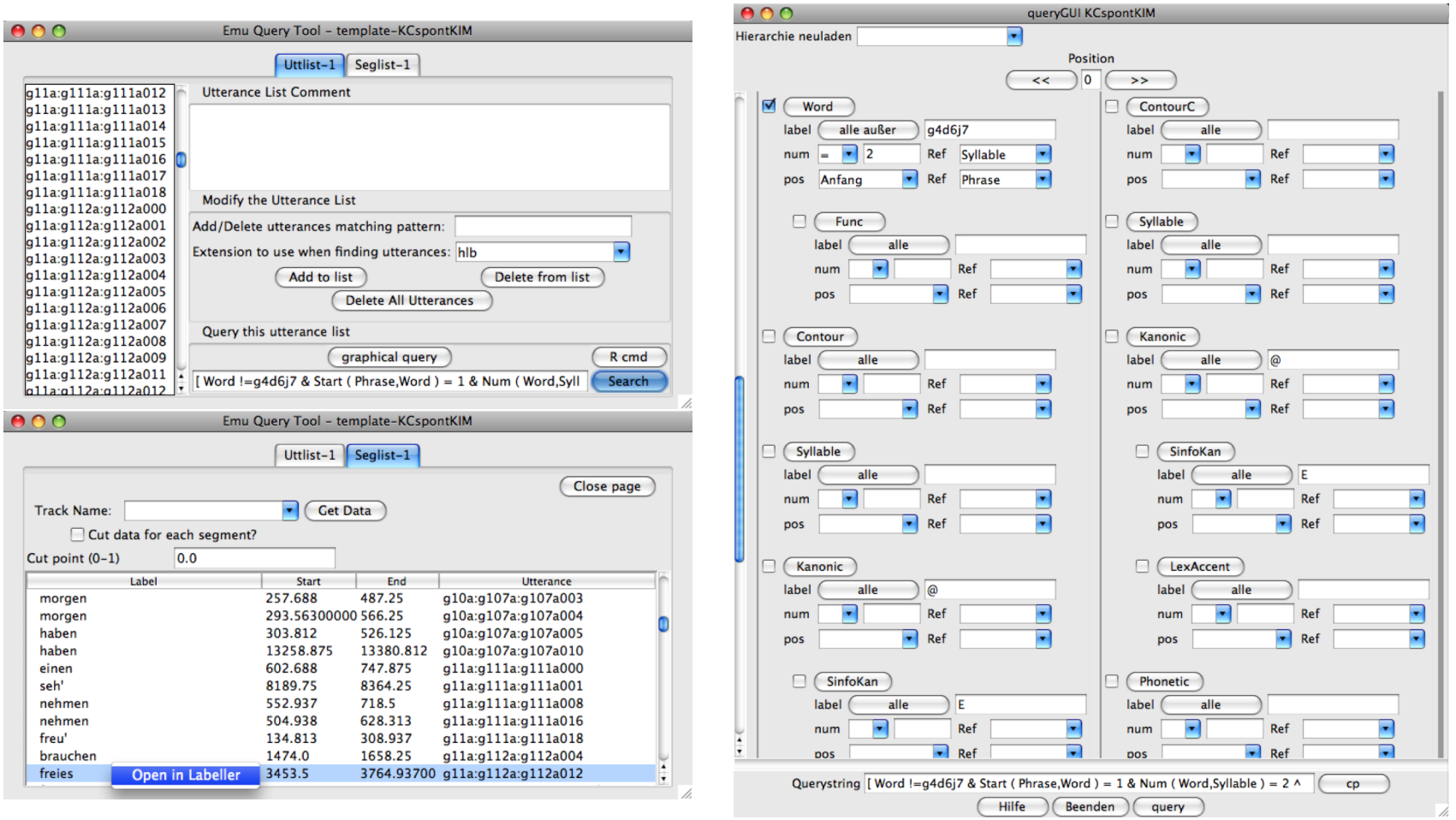


Abbildung A.4.: Das EMU Query Tool (links), die Abfrageregisterkarte (oben) mit den Einträgen für die abzufragenden Äußerungen des Kiel Corpus sowie EQ 1 und die Segmentliste in einer Registerkarte (unten) zum direkten Öffnen der Tokens im EMU Labeller. Das queryGUI (rechts) mit Ausschnitten der beiden Ketten im Etikettierungsschema 14 aus Kapitel 7 und Einträgen für EQ 1, S. 96. In der ersten Spalte ist ein beliebiges Wortetikett und die phraseninitiale Position des Wortes definiert sowie die Anzahl der vom Wort dominierten Silben und für die Kanonik ein /ə/.

B. Tabellen

Tabelle B.1.: EMU-Applikationen und Programmpakete mit ihren Bezeichnungen im Front- und Backend sowie ihrer Entwickler in der Reihenfolge der Verantwortung.

(a) Grafische Oberflächen

Applikationsname im Frontend	Paketname im Backend	Entwickler
autoDBinstaller	`autoDBinstaller`	John
AutoBuild Wizard	`autobuild`	John
EPG2SSFF	`epgconv`	Bombien
Datenbank Tool	`dbemu`	Bombien/John
G(raphical)Template Editor	`GTed` in `tpled`	John
Konfigurationseditor	`emuconf`	John
labConvert	`labConvert`	John
Labeller	`labeller2`	John
Pitch & Formant Tool	`snacktrack`	keine Angabe
Query Tool	`querytool`	Cassidy
queryGUI	`queryGUI`	John
Script	`EMU-script`	John
Segmenter	`capture`	Simpson
Tkassp	`tkassp`	Bombien
Wizard	`tkwizard`	Cassidy
Browserhilfe	`help`	Gratchev

(b) Programmpakete

Programmname	Paketname	Entwickler
EMU-Schnittstelle zur R Programming Lanugage	`EMU/R-Interface`	Cassidy/Bombien/John
EMU-core	`emu-core`	Cassidy und mehr
EMU/R Bibliothek	`EMU/R lib`	Harrington
libassp	`libassp`	Scheffers/Bombien
Sendpraat	`sendpraat`	Boersma
Snack Toolkit	`snack`	Sjölander
tclassp	`tclassp`	Bombien

Tabelle B.2.: Kiel Corpus Etiketten (KC original), die Ersetzungen in der Konvertierung nach EMU und die Bedeutung der Etiketten für Aussersprachliches.

KC original	KC in EMU	Semantik
z:	ASh.length	Längung durch Häsitation
v:	AShesit	Häsitation
n:	ASneolog	Neologismen, nicht Wort
p:	ASpause	Pause
h:	ASbreath	Atmen und Atempause
l:	ASlaugh	Lachen
q:	AScough	Husten
r:	ASthroatcle	Räuspern
s:	ASclick	Schmatzen
w:	ASswallow	Schlucken
g:	AS	Andere
:k	ASext.noise	verschiedene externe Geräusche
;	AStechn.break	technischer Abbruch

Tabelle B.3.: Kiel Corpus Etiketten (KC original), die Ersetzungen in der Konvertierung nach EMU und die Bedeutung der Etiketten für Phrasen.

KC original	KC in EMU	Semantik
PGn	P	Default Fall
=PGn	Pds	der erste Gipfel der nachfolgenden Phrase liegt tiefer als der letzte Gipfel der vorangehenden Phrase (d. h. Downstep über die Phrasengrenze hinweg)
PG/	Psb	die Phrasengrenze fällt mit einem syntaktischen Abbruch zusammen
PG;	Ptb	die Phrasengrenze wurde durch einen technischen Abbruch verursacht

Tabelle B.4.: Kiel Corpus Etiketten (KC original), die Ersetzungen in der Konvertierung nach EMU und die Bedeutung der Etiketten für Akzentstufen.

KC original	KC in EMU	Semantik
0	0	deakzentuiert
1	1	partiell deakzentuiert
2	2	default Akzentstärke
3	3	emphatisch verstärkte Akzentuierung

Tabelle B.5.: Kiel Corpus Etiketten (KC original), die Ersetzungen in der Konvertierung nach EMU und die Bedeutung der Etiketten für Akzentkonturen.

KC original	KC in EMU	Semantik
)	eP	früher Gipfel
^	mP	mittlerer Gipfel
(	lP	später Gipfel
–	nP	es bleibt eben vor, über und nach der Akzentsilbe
]	eV	frühes Tal, Einsatz liegt vor dem Akzentvokal
[	neV	nicht frühes Tal, Einsatz liegt in dem Akzentvokal
|)	UeP	Upstep des ...
|^	UmP	Upstep des ...
|(	UlP	Upstep des ...
|–	UnP	Upstep des ...
|]	UeV	Upstep des ...
|[	UneV	Upstep des ...
	D	Default

Tabelle B.6.: Kiel Corpus Etiketten (KC original), die Ersetzungen in der Konvertierung nach EMU und die Bedeutung der Etiketten für phraseninitiale Konturen.

KC original	KC in EMU	Semantik
HP1	HP1	der Anfang der Phrase liegt auf dem gleichen bzw. geringfügig höherem Niveau im Vergleich zum ersten Gipfel der Phrase
HP2	HP2	der Anfang der Phrase liegt auf dem bedeutsam höheren Niveau im Vergleich zum ersten Gipfel der Phrase und das F0 fällt häufig schnell zum Gipfel hin ab
D	D	für fehlende Verläufe im Kiel Corpus mit der Bedeutung Default

Tabelle B.7.: Kiel Corpus Etiketten (KC original), die Ersetzungen in der Konvertierung nach EMU und die Bedeutung der Etiketten für Konkatenationskonturen.

KC original	KC in EMU	Semantik
0.	nF	Ebener Verlauf bzw. kein Absinken
1.	mF	Leichtes Absinken
2.	tF	Absinken bis zur Base–Linezw.

Tabelle B.8.: Kiel Corpus Etiketten (KC original), die Ersetzungen in der Konvertierung nach EMU und die Bedeutung der Etiketten für einfache phrasenfinale Konturen.

KC original	KC in EMU	Semantik
0.	nF	nach Gipfelkonturen bleibt es eben
1.	mF	nach Gipfelkonturen fällt es leicht ab
2.	tF	nach Gipfelkonturen fällt es bis zur Base–Line ab
,	mR	nach Talkonturen: Anstieg bis zum mittleren Bereich der Sprechstimme
?	hR	nach Talkonturen: Anstieg bis zum obersten Bereich der Sprechstimme f0

Tabelle B.9.: Kiel Corpus Etiketten (KC original), die Ersetzungen in der Konvertierung nach EMU und die Bedeutung der Etiketten für komplexe phrasenfinale Konturen.

KC original	KC in EMU	Semantik
0;	nFmR	pseudoterminal: nach ebenen leichter Anstieg
1;	mFmR	pseudoterminal: nach einem nicht besonders tiefen Fallen leichter Anstieg
2;	tFmR	pseudoterminal: nach einem tiefen Fallen leichter Anstieg
1.,	mFtR	fallend–steigend: nach einem nicht besonders tiefen Fallen ein deutlicher Anstieg, der nicht bis zur obersten Grenze der Sprechstimme geht
2.,	tFtR	fallend–steigend: nach einem tiefen Fallen ein deutlicher Anstieg, der nicht bis zur obersten Grenze der Sprechstimme geht
1.?	mFhR	fallend–steigend: nach einem nicht besonders tiefen Fallen ein deutlicher Anstieg, der bis zur obersten Grenze der Sprechstimme geht
2.?	tFhR	fallend–steigend: nach einem tiefen Fallen ein deutlicher Anstieg, der bis zur obersten Grenze der Sprechstimme geht

C. Scripts

Quellcode C.1: R Befehle zur Analyse der Anzahl der zurückgegebenen Segmente

```
eq = "Word!=g4d6j7&Start(Phrase,Word)=1&Num(Word,Syllable)=2"
seg = emu.query("KCspontKIM","*", eq)
length(unique(seg$utt))
```

Quellcode C.2: R Befehle zum Einschränken einer Segmentliste mithilfe regulärer Ausdrücke in R

```
#Abfrage aller Woerter in der gesamten Aeusserung
seg = emu.query("KCspontKIM","*g112a012*","Word!=x")

# Segmentliste der Abfrage
#-----------------------------------------
# segment list from database: KCspontKIM
# query was: Word != x
#      labels          start          end            utts
#1     freies          0.100          311.537        g112a012
#2     Wochenende      311.537        761.975        g112a012
#3     kann            761.975        884.038        g112a012
#4     man             884.038        980.225        g112a012
#5     sich            980.225        1084.538       g112a012
#6     ja              1084.538       1162.475       g112a012
#7     freihalten      1162.475       1703.975       g112a012

# Extraktion der Etikettenspalte
labs = seg$label

#logischer Vektor der Ergebnisse des regulaeren Ausdrucks
lvek = regexpr("n{2,2}",labs) > 0

# Einschraenkung der Segmentliste
seg[lvek,]

# Ergebnisse in R
########################################
# segment list from database: KCspontKIM
# query was: Word != x
#      labels  start          end            utts
# 3    kann    761.975        884.038        g112a012
```

Quellcode C.3: R Befehle zur Positionsabfrage mit unbekannter Anzahl an Segmenten

```
# Segmentlistenvariablen zur Verfuegung stellen, fuer Abfragen,
# die kein Ergebnis liefern

seg1=NULL
seg2=NULL
seg3=NULL

# der Abfrageausdruck in einer Variable zur Uebersichtlichkeit
# Abfrageausdruck mit einem unbekannten Etikett x
eq = "[Word = nehmen -> [Word != x -> Word = siebzehnten]]"

# die Abfrage
seg1 = emu.query("KCspontKIM","*g103*","eq")

# Abfrageausdruck mit zwei unbekannten Etiketten x
eq = "[Word = nehmen -> [Word != x -> [Word != x -> Word = siebzehnten]]]"
seg2 = emu.query("KCspontKIM","*g103*",eq)

# Abfrageausdruck mit drei unbekannten Etiketten x
eq = "[Word = nehmen -> [Word != x -> [Word != x -> [Word != x -> Word = siebzehnten]]]]"
seg3 = emu.query("KCspontKIM","*g103*", eq)

# Zusammenfuegen der Ergebnisse
allseg = rbind(seg1,seg2,seg3)

# Ergebnis in R
segment list from database: KCspontKIM
query was: [Word = nehmen -> [Word != x -> [Word != x -> [Word != x -> Word = siebzehnten]]]]
                                labels start      end      utts
1 nehmen->wir->Montag->den->siebzehnten 3694.5 5287.187 g103a010


#########################
# Alternative Schleife: #
#########################

s1 = "[Word = nehmen -> "
s2 = "Word = siebzehnten]"
sx = "[Word != x -> "
allseg = NULL
for (i in 1:5) {

    eq = paste(s1,sx,s2,rep("]",i))
    sx = paste(sx,sx)
    seg = NULL
    seg = emu.query("KCspontKIM","*g103*",eq)
    allseg = rbind(allseg,seg)
}
```

Quellcode C.4: Angepasster Programmcode aus Cassidy(1999) des emu2dbase Skripts zum Überführen von EMU-Etikettierungen in eine relationale Struktur.

```
package require emu

namespace eval emu {}
namespace eval emu::emu2dbase {

proc sql_create_tables {handle template} {
   variable sql

    ## create the database and select it
    puts $handle "DROP DATABASE IF EXISTS [$template name];"
    puts $handle "CREATE DATABASE [$template name];";
    puts $handle "use [$template name]"

    puts $handle "CREATE TABLE levels ("
    puts $handle "id    int NOT NULL PRIMARY KEY,"
    puts $handle "name  char(20) NOT NULL"
    puts $handle ");"

    puts $handle "CREATE TABLE tokens ("

    puts $handle "      id    char(20) NOT NULL PRIMARY KEY, "
    puts $handle "      level int NOT NULL,"
    puts $handle "      stime real NOT NULL,"
    puts $handle "      etime real NOT NULL,"
    puts $handle "      seq   int NOT NULL,"
    puts $handle "      label char(20) NOT NULL,"
    puts $handle "      utt   char(50) NOT NULL,"
    puts $handle "      INDEX labelindex (label)"
    puts $handle ");"


    puts $handle "CREATE TABLE links ("
    puts $handle "      parent  char(20) NOT NULL,"
    puts $handle "      child   char(20) NOT NULL,"
    puts $handle "      INDEX parentindex (parent),"
    puts $handle "      INDEX childindex (child)"
    puts $handle ");"

    puts $handle "LOAD DATA LOCAL INFILE '$sql(tokens,file)' INTO TABLE tokens;"
    puts $handle "LOAD DATA LOCAL INFILE '$sql(levels,file)' INTO TABLE levels;"
    puts $handle "LOAD DATA LOCAL INFILE '$sql(links,file)' INTO TABLE links;"

}

proc make_segid {utt seg} {
    return "$utt/$seg"
}

proc sql_insert_hierarchy {handle template hier} {
   variable sql

    set uttname [$hier basename]
    set sql(uttid,$uttname) $uttname
    foreach l [$template getlevels] {
        set seq 0
        set segments [$hier segments $l]
        for {set i 0} {$i < [llength $segments]} {incr i} {
            sql_insert_segment $handle $template $hier $l \
                [lindex $segments $i] [incr seq]
        }
    }
    sql_insert_links $handle $template $hier
}

proc sql_insert_levels {handle template} {
   variable sql

    set levelid 0
    foreach l [$template getlevels] {
        set sql(levelid,$l) [incr levelid]
        puts $sql(levels) "$sql(levelid,$l)\t$l"
```

```
72      }
73  }
74
75  proc sql_insert_segment {handle template hier level segment seq} {
76     variable sql
77
78      set utt $sql(uttid,[$hier basename])
79      set segid [make_segid $utt $segment]
80      set label [$hier seginfo $segment label $level]
81      set times [$hier seginfo $segment times]
82
83      if {$label == {}} {set label "-"}
84
85      puts -nonewline $sql(tokens) "$segid\t"
86      puts -nonewline $sql(tokens) "$sql(levelid,$level)\t"
87      puts -nonewline $sql(tokens) "[lindex $times 0]\t"
88      puts -nonewline $sql(tokens) "[lindex $times 1]\t"
89      puts -nonewline $sql(tokens) "$seq\t"
90      puts -nonewline $sql(tokens) "$label\t"
91      puts            $sql(tokens) "$utt"
92  }
93
94  ## this needs to be done once all of the sql(segid,*) elements are set for
95  ## this utterance
96  proc sql_insert_links {handle template hier} {
97     variable sql
98
99      set utt $sql(uttid,[$hier basename])
100     foreach level [$template getlevels] {
101         foreach parent [$hier segments $level] {
102             ## we go for descendents here to get all transitive links
103             foreach cl [$template descendents $level] {
104                 foreach child [$hier seginfo $parent children $cl] {
105                     puts $sql(links) \
106                         "[make_segid $utt $parent]\t[make_segid $utt $child]"
107                 }
108             }
109         }
110     }
111 }
112
113 proc emu2dbase args {
114     variable sql
115
116     set argv $args
117 if {[lindex $argv 0] == "-append"} {
118     set options {WRONLY APPEND CREAT}
119     set templatename [lindex $argv 1]
120     set pattern      [lindex $argv 2]
121 } else {
122     set options {WRONLY TRUNC CREAT}
123     set templatename [lindex $argv 0]
124     set pattern      [lindex $argv 1]
125 }
126
127     emutemplate Template $templatename
128
129     ## create a full pathname
130     set dir [file join [pwd] $templatename]
131
132     if {![file exists $dir]} {
133         file mkdir $dir
134     }
135
136     set sql(tokens,file) [file join $dir tokens.db]
137     set sql(tokens) [open $sql(tokens,file) $options]
138
139
140     set sql(levels,file) [file join $dir levels.db]
141     set sql(levels) [open $sql(levels,file) $options]
142     set sql(links,file) [file join $dir links.db]
143     set sql(links)  [open $sql(links,file) $options]
144
145     set handle [open [file join $dir database] w]
```

```
146
147     sql_create_tables $handle Template
148     set utts [Template utterances $pattern]
149
150     set utlen [llength $utts]
151     puts "Got $utlen utterances"
152
153     sql_insert_levels $handle Template
154
155     foreach u $utts {
156         Template hierarchy Hierarchy $u
157         sql_insert_hierarchy $handle Template Hierarchy
158     }
159
160
161     close $sql(tokens)
162     close $sql(levels)
163     close $sql(links)
164     close $handle
165     puts "done"
166     }
167 }
```

Quellcode C.5: Leicht komplexe SQL-Abfrage einer EMU-Dominanzbeziehung zum Abfragen aspirierter alveolarer Plosive oder der Aspiration dieser Plosive (alternative SELECT Anweisungen).

```
1  # Finde alle aspirierten /d/ Phoneme, um zu sehen, ob es sie in der Datenbank gibt
2  SELECT DISTINCT
3    parentseg.label,
4    parentseg.stime,
5    parentseg.etime,
6    parentseg.utt
7
8  # Alternativer denkbarer SELECT Block mit gleichem FROM und WHERE Block
9  # fuer finde alle Aspirationen von /d/ Phonemen zum Messen der VOT
10 SELECT DISTINCT
11   childseg.label,
12   childseg.stime,
13   childseg.etime,
14   childseg.utt
15
16
17 # Platzhalternamen fuer zwei Tokens (childseg und parentseg) und Ebenen (childlev und parentlev)
18 # werden vergeben, um die Eigenschaften mehrerer Einheiten unterscheiden zu koennen
19 FROM
20   tokens childseg,
21   levels childlev,
22
23   tokens parentseg,
24   levels parentlev,
25
26   links
27
28
29 # die gewuenschten Eigenschaften der beiden Tokens auf den beiden Ebenen werden bestimmt
30 # und  das die beiden Tokens miteinander verknuepft sein muessen
31 WHERE
32   childseg.label = 'H' and
33   childlev.name = 'Phonetic' and
34   childseg.level = childlev.id and
35
36   parentseg.label = 'd' and
37   parentlev.name = 'Phoneme' and
38   parentseg.level = parentlev.id and
39
40   links.child = childseg.id and
41   links.parent = parentseg.id;
```

Glossar

0:1

Beschränkung im ER-Modell: genau einmal (vgl. S. 36).

0:1

Beschränkung im ER-Modell: keinmal oder einmal (vgl. S. 36).

0:N

Beschränkung im ER-Modell: keinmal bis beliebig oft (vgl. S. 77ff.).

1:N

Beschränkung im ER-Modell: einmal bis beliebig oft (vgl. S. 36ff.).

API

Application Programming Interface; Programmierschnittstelle für Entwickler (vgl. S. 116).

Attribut

Eigenschaften, über die Objekte der realen Welt als Entitäten zu Entitätstypen bzw. Relationen zu Relationstypen zusammengefasst werden können. Der Typ bestimmt sich durch seine Attribute. Notation: beschriftetes Oval im ER-Modell, **Attribut** oder **attribut** im Text (vgl. S. 34ff.).

Ausprägung

Ein Objekt aus der realen Welt als Instanz eines Typs, der ähnliche Objekte der realen Welt über ihre gemeinsamen Eigenschaften modelliert (vgl. S. 34). Im ER-Modell ist es eine Entität des Entitätstyps, im relationalen Modell eine Relation des Relationstyps.

Backend

Teil der Software, der durch den Benutzer nicht offensichtlich zugänglich ist, sondern u. a. die technische Umsetzung des Frontends enthält bzw. unterstützt (vgl. Abbildung 1.2 auf S. 19).

Beziehung

auch Relationship; Verbindung zweier Entitäten (vgl. S. 34).

Beziehungstyp

auch Relationship-Typ; Verbindung zweier Entitätstypen(vgl. S. 35). Notation: Verbindungslinie zwischen Entitätstyp-Rechtecken mit Raute im ER-Schemadiagramm.

Dateiheader

auch Header; Informationen zur Datei vor Beginn der eigentlichen Daten.

Datenbankkatalog

Zentrale Sammlung der verfügbaren Datenbanken in Form der verfügbaren Datenbankschemata (vgl. S. 32).

Datenbankschema

Strukturelle Beschreibung einer Datenbank (vgl. S. 32ff.).

Datenbanktemplate

auch Template; Externes Datenbankschema einer EMU-Datenbank (vgl. S. 71ff.); benutzereditierte Textdatei mit allen Deklarationen zum Etikettierungsschema, Signalen, Dateispeicherorten, zu verwendbaren Skripten und Ansichten im EMU-Etikettierungstool (vgl. S. 83ff.).

Datenbankzustand

Die Ausprägungen des Datenbankschemas (vgl. S. 38).

DBMS

Datenbankmanagementsystem – Teil des Datenbanksystems, der die Routinen der zentralen Verwaltung übernimmt (vgl. S. 28).

DDL

Datendefinitionssprache zur Beschreibung der Datenobjekte (vgl. S. 40ff.).

DML

Data Manipulation Language zur Abfrage, Anfrage und Manipulation der Daten und Datenobjekte (vgl. S. 40ff.).

Eltern

Aus dem EMU-Etikettierungsmodell (vgl. Kapitel 3.3, S. 56): Ein Token auf einer anderen Ebene dominierendes Token.

Elternebene

Aus dem EMU-Etikettierungsmodell (vgl. Kapitel 3.3): Eine Kindebene dominierende Ebene.

Entität

Basisobjekt des Entity-Relationship Modells. Ausprägungen/Instanzen eines Entitätstypen. Es stellt Objekte oder Konzepte aus der realen Welt dar (vgl. S. 34ff.).

Entitätstyp

Sammlung von Entitäten mit gleichen Attributen (vgl. S. 35ff.). Notation: beschriftete Rechtecke im ER-Schemadiagramm; **ENTITÄTSTYP** oder zum besseren Verständnis im Kontext: **Entitätstyp** oder **entitätstyp** im Text.

ER-Modell

Entity-Relationship Modell (S. 34).

event-zeitgebundene Ebene

Etikettierungsebene im EMU-Etikettierungsmodell mit dem Attribut der Zeitgebundenheit EVENT, konzeptuell modelliert mit EBENE (E). Es können Tokens mit einer Zeitmarke auf dieser Ebene etikettiert werden (vgl. S. 55ff.).

Frontend

Teil der Software, der für den Benutzer grafische und funktionale Schnittstellen bietet (vgl. Abbildung 1.2 auf S. 19).

Gesamtsicht

Alle Informationen und auch interne Datenstrukturen in einer Datenbank.

GUI

Graphical User Interface; grafische Benutzerschnittstelle.

Header

auch Dateiheader; Informationen zur Datei vor Beginn der eigentlichen Daten.

Hierarchie

auch Kette; Die Verknüpfung von u. U. unterschiedlich assoziierten Etikettierungsebenen im EMU-Etikettierungsmodell. Es können mehrere Ketten innerhalb eines Etikettierungsschema vorliegen, wobei eine Kette als Hauptkette definiert sein muss (vgl. S. 58).

Informationsschema

Strukturelle Beschreibung von möglichen Datenbankschemata (vgl. 'INFORMATION SCHEMA', S. 32).

intervall-zeitgebundene Ebene

Etikettierungsebene im EMU-Etikettierungsmodell mit dem Attribut der Zeitgebundenheit SEGMENT, konzeptuell modelliert mit EBENE (S). Es können Tokens mit Start- und Endzeit somit einer Dauer auf dieser Ebene etikettiert werden (vgl. S. 55ff.).

Kette

auch Hierarchie; Die Verknüpfung von u. U. unterschiedlich assoziierten Etikettierungsebenen im EMU-Etikettierungsmodell. Es können mehrere Ketten innerhalb eines Etikettierungsschema vorliegen, wobei eine Kette als Hauptkette definiert sein muss (vgl. 56ff.).

Kind

Aus dem EMU-Etikettierungsmodell (vgl. Kapitel 3.3, S. 56): Ein Token auf einer Etikettierungsebene, das ein Token auf einer anderen Ebene dominiert.

Kindebene

Aus dem EMU-Etikettierungsmodell (vgl. Kapitel 3.3): Von Elternebene dominierte Ebene.

labfile

Externe Etikettierungsdatei für zeitgebundene Ebenen im EMU-Etikettierungsmodell (vgl. S. 79), bzw. der Entitätstyp in der Entity-Relationship Modellierung, um diese Art von Objekten zu modellieren (vgl. S. 74).

legal labels

Aus dem EMU-Etikettierungsmodell: Legale Zeichenmenge als Angabe für die möglichen jedoch nicht ausschließlich möglichen Etiketten auf einer Etikettierungsebene (vgl. S. 61). Sie sind pro Ebene in Klassen organisiert, die wiederum als Etikett z. B. in der Abfrage mit der EMU Query Language verwendet werden können und referieren dabei auf alle Zeichen in der Klasse.

Mapping

Das aufeinander Abbildung zweier Objekte, Konzepte, etc. (vgl. S. 41).

Meta-Identifier

Terminus aus der formalen Sprache als Bezeichner für ein nicht terminales Symbol (vgl. S. 48).

min:max Notation

Einschränkungen auf die Anzahl beteiligter Entitäten in einer Beziehung des ER-Modells, z. B.: In einer Beziehung zwischen Etikettierungsebene und Token ist eine Ebene beteiligt und 0 bis unendlich viele Tokens. So kann eine Ebene kein, ein oder mehrere Tokens enthalten (0:N). Aber das Token darf nur und muss aber auch mit einer Ebene (1:1) in Beziehung stehen (vgl. S. 36).

N

Beschränkung im ER-Modell: beliebige Zahl (vgl. S. 36ff.).

nicht terminales Symbol

Terminus aus der formalen Sprache für definierte Zeichen oder Zeichenketten, die als solche in der jeweiligen Sprache in andere terminale oder nicht terminale Symbole zerlegt werden können (vgl. S. 48).

QL

Query Language, Teil der DML zur Abfrage der Daten (vgl. S. 40ff.).

Relation

Ausprägung eines Relationstyps, z. B. KUNDE(0001,Kiel,Hanson) (vgl. S. 38).

Relationstyp

Abstrakte Beschreibung einer Relation im Relationalen Datenbankmodell. Er modelliert die Objekte der realen Welt über gemeinsame Attribute z. B. KUNDE(Kundennr,Anschrift,Name). Notation: **relationstyp** (vgl. S. 38).

Rolle

Die Funktion einer Entität in einem Beziehungstyp, z. B.: Eine Etikettierungsebene im EMU-System kann als Rolle eine Elternebene oder eine Kindebene in der Beziehung zu anderen Ebenen darstellen (vgl. S. 37).

SAMPA

Speech Assessment Methods Phonetic Alphabet; Maschinenlesbares Phonetisches Alphabet in ASCII Kodierung (vgl. S. 6).

Segmentliste

Liste der durch eine EQL-Abfrage gefundenen Tokens in einer Datenbank aufbereitet in einer Tabelle von vier Spalten mit dem Etikett des Tokens, der Start- und Endzeit sowie der Äußerung, in der das Token gefunden worden ist (vgl. S. 89).

Segmentnummer

Index eines EMU-Tokens im internen Schema des EMU-Systems (vgl. S. 76).

Sicht

Ein Ausschnitt der Datenbank oder auch abgeleitete Daten aus der Datenbank. (vgl. S. 29).

Template

Im Allgemeinen ein Muster; für die Verwendung in EMU vgl. Datenbanktemplate.

Templatedatei

Vgl. Datenbanktemplate.

Terminal (terminales Symbol)

Terminus aus der formalen Sprache für Zeichen oder Zeichenketten, die als solche in der jeweiligen Sprache nicht mehr zerlegt werden können (vgl. S. 48).

TextGrid

Praat-Etikettierungsdatei (vgl. S. 172).

Track

Signale, die im EMU-System als *Tracks* in der Datenbanktemplate deklariert werden (vgl. S. 72).

Tupel

Ausprägung der Attribute in einer Relation eines Relationstyps, z. B. (0001,Kiel,Hanson) (vgl. S. 38).

Danksagungen

In allererster Linie möchte ich mich bei Prof. Dr. Jonathan Harrington nicht nur für die qualitativ hochwertige zeitgenössische Ausbildung während des Studiums und als Doktorvater bedanken, sondern auch dafür, dass er mir die Chance gab, mich auf unbekanntes Gebiet zu wagen. Er unterstützte mich immer auf dem Weg zur Vielseitigkeit auch als das Meer Berge versetzte. Jonathan, vielen Dank, ich weiß das wirklich unheimlich zu schätzen.

Auch an meine Hochschullehrer als solche geht ein Dank für die gute Ausbildung: Prof. Dr. Klein aus der Informatik, der durch seine Rhetorik und Terminologiefestigkeit in der Lehre fasziniert, so dass sich vielleicht erst gegen Ende aber dann gewiss der Kreis schließt, Prof. Dr. em. Ulrike Mosel, die mich auch in der Linguistik Fuß fassen ließ, Prof. Dr. em. Kohler und Prof. Dr. Harrington, von denen mir die Welt der Phonetik von mehreren Seiten zugänglich gemacht wurde, Prof. Dr. Simpson, dessen Hang zum richtigen Formalismus und der Überprüfung der Daten prägend war, nicht zuletzt Herrn Prof. Dr. Schulz für die Chance einen Hauptseminarschein im Nebenfach Computerlinguistik machen zu dürfen. Auch an PD Dr. Christoph Draxler geht ein ehrlicher Dank für die lehrhafte Erfahrung, die auch ich mal machen sollte und das zum einzig richtigen Zeitpunkt.

Ein Dank geht auch an alle IPSler in München, die mich so freundlich unter sich aufgenommen und unheimlich viel Geduld mit mir haben, sowie an alle IPdSler, die mir bis zum letzten Tag die Türen offen hielten. An alle EMU-Entwickler für die gute Zusammenarbeit: Lasse Bombien, Steve Cassidy, Mich(a)el Scheffers, ohne die das System nicht so weit gekommen wäre, wie es jetzt ist und den EMU-Nutzern, die uns als Entwicklungsteam mit Fehlermeldung, Fragen und Anregungen versorgen, geht ein besonderer Dank.

Dieser Dank geht auch alle meine Freunde, auch jene, die ich nicht namentlich erwähnen kann. Der Dank gilt im Besonderen Ariana und Ramona, die beide auf ihre Weise maßgeblich an dieser Arbeit beteiligt waren, die ‚mit Robert Spieler', die unbewusst immer das Richtige tun, Arne, für den Spaß auf und neben dem neuen

Parkett, Gunni, dem zuverlässigen Held auf dem schwarzen Motorrad und nicht zuletzt Henrike, die sich beherzt auf neues Gebiet begab.

Das sind Freunde.

Lebenslauf

— Tina John —

Geburtsdatum: 13.05.1981 Geburtsort: Karl-Marx-Stadt
Matrikelnr.: 10203887

Ausbildung:

1999 Abitur am Werner-Heisenberg Gymnasium in Chemnitz

1999 - 2003 Magisterstudium am Institut für Phonetik und digitale Sprachverarbeitung der Christian-Albrechts-Universität zu Kiel

Hauptfach: Phonetik und digitale Sprachverarbeitung
1. Nebenfach: Allgemeine Vergleichende Sprachwissenschaft
2. Nebenfach: Informatik

2003 - 2004 Magisterarbeit zu „Eine akustische Analyse der Lenis/Fortis-Opposition in Varietäten des Sächsischen“ und Abschlussprüfungen in Phonetik, Linguistik und Informatik

Beruflicher Werdegang:

2002 - 2004 studentische Hilfskraft am Institut für Phonetik und digitale Sprachverarbeitung an der Christian-Albrechts-Universität zu Kiel

Softwareentwicklung

2004 - 2006 wissenschaftliche Zeitangestellte am Institut für Phonetik und digitale Sprachverarbeitung an der Christian-Albrechts-Universität zu Kiel

Lehre und Softwareentwicklung

2006 - 2011 wissenschaftliche Zeitangestellte am Institut für Phonetik und Sprachverarbeitung an der Ludwig-Maximillian-Universität München

DFG Schwerpunktprogramm: “Phonological and phonetic competence: between grammar, signal processing, and neural activity” — “Acoustic, articulatory and perceptual analyses of the post-vocalic voicing contrast in two varieties of German.”

Seit 2008 Lehrkraft für besondere Aufgaben am Institut für Allgemeine und vergleichende Sprachwissenschaft an der Christian-Albrechts-Universität zu Kiel

Akademische Lehrer:

Phonetik: Prof. Dr. Kohler, Prof. Dr. Simpson, Prof. Dr. Harrington

Linguistik: Prof. Dr. Mosel, Prof. Dr. Dr. habil Pieper

Informatik: Prof. Dr. Koch, Prof. Dr. Simon, Prof. Dr. Klein

Computerlinguistik: Prof. Dr. Schulz