

LIZENTIATSARBEIT
DER PHILOSOPHISCHEN FAKULTÄT
DER UNIVERSITÄT ZÜRICH

A CORPUS QUERY TOOL
FOR SYNTACTICALLY
ANNOTATED CORPORA

Author: Charlotte Merz
Supervisor: PD Dr. Martin Volk

May 2003

Contents

1	Introduction	1
2	Components of Corpus Query	3
2.1	Use and Types of Corpora	3
2.1.1	Corpus Design	3
2.1.2	Corpus Preprocessing	6
2.1.3	Part-of-Speech Tagging	10
2.1.4	Morphological Analysis and Lemmatization	11
2.1.5	Parsing and Treebanks	13
2.1.6	Existing Corpora	16
2.2	Corpus Query	16
2.2.1	Corpus Query Languages	20
2.2.2	Corpus Query Tools	22
3	Database Systems	36
3.1	Introduction to Database Systems	36
3.1.1	Characteristics and Advantages of the Database Approach	36
3.1.2	The Relational Data Model	37
3.2	Database Systems in Corpus Linguistics	37
3.2.1	Corsica	38
3.2.2	San Remo	38
3.2.3	ANNOTATE Database Format	39

4	My Own Corpus Query Tool	41
4.1	Aim, Scope, and Technical Resources	41
4.2	Database Format	43
4.2.1	Entity-Relationship Model Approach	43
4.2.2	N-table Approach	47
4.3	Corpora	51
4.3.1	Resources: ComputerZeitung and Tages-Anzeiger	51
4.3.2	Corpus Transfer to Database	59
4.4	Query	65
4.4.1	Simple Query	65
4.4.2	Complex Query	67
4.4.3	Query Optimization Strategies	69
4.5	Interface Functions	70
4.6	Output Presentation	72
4.6.1	Simple Query Output: KWIC and Tag-Display	72
4.6.2	Complex Query Output: Table with Syntactic and Semantic Structure	74
5	Analysis and Evaluation	80
5.1	Interface Evaluation	80
5.2	Output Evaluation	81
5.3	Database Performance	82
5.3.1	Comparison of Database Formats	82
5.3.2	Database Performance	90
5.3.3	Database Evaluation	93
5.3.4	Suggestions for Improvement	94
5.4	Conclusion	96
	Bibliography	98
	Glossary	100

<i>CONTENTS</i>	iii
A SGML-Tags in the CZ and TA Corpora	103
B List of Database Relations	105
B.1 Database Format Version 1	106
B.2 Database Format Version 2	109
C Simplified STTS Tagset	116
D PHP-programs on a Separate CD-ROM	117

Note on the Text

In order to make this thesis readable for persons who have not studied computational linguistics, I have included a glossary of the most important terms used in the text. Terms which are explained in the glossary on page 101 are printed in CAPITALS.

Examples or emphasized words are displayed in *italicized* font; names of entities in entity-relationship models or names of tables in a relational database system are printed in `type-writer` font.

The corpus query system programmed in this project is at the time of writing running on a server of the Department of Computer Science at the University of Zurich. It can be tested at <http://www.ifi.unizh.ch/cl/chmerz/CorpusQuery/start.html>. A login-name and password can be obtained from the author.

Chapter 1

Introduction

Corpus linguistics is a growing area of research in language studies as well as in computational linguistics. Despite its increasing popularity, corpus linguistics still represents a question mark for many scholars of traditional linguistics. This, on one hand, is due to the fact that the extent of corpus linguistics is not comparable to the focus of any of the established linguistic fields such as syntax, morphology or sociolinguistics. Corpus linguistics is rather a methodology which can be applied to all linguistic paradigms. On the other hand, corpus linguistics makes extensive use of computers for the analysis of specific problems. This leads to a certain degree of mistrust. Allow me explore these two points and correct any misunderstandings about corpus linguistics from the beginning.

Theoretical linguistics, as it is traditionally taught, emphasizes the study of language structure. It establishes systems which explain language as a theoretical model. A linguist can compare language in use with these theories and make statements about the grammatical correctness of a sentence or the order morphological components in a word. In Chomsky's words, theoretical linguistics strives to achieve EXPLANATORY ADEQUACY in explaining a speaker's COMPETENCE.

As opposed to this, corpus linguistics aims at achieving DESCRIPTIVE ADEQUACY of the speaker's PERFORMANCE. Corpus linguists study the actual patterns of use of language in naturally occurring texts, thereby relying on large collection of texts in order to conduct empirical analyses. This is based on the idea that language has to be explored as it is used and not restricted to the domain covered by a theoretical model. The increasing use of computers allows processing of large amounts of data, thereby increasing and accelerating the possibilities of corpus linguistics immensely. It is important to note, however, that corpus linguistics is first of all a methodology which can be applied in every branch of linguistics¹.

This is the ideological gap which lies between theoretical and descriptive linguistics. Although the positions taken up by these parties seem to be far apart, Meyer ([Meier 2002]:xiv)

¹For an extensive list of the use of corpora in language studies see chapter 4 in [McEnery and Wilson 1996] or pp. 11-28 in [Meier 2002].

concludes:

[W]hile the goals of the corpus linguist and the generative grammarian are often different, there is an overlap between the two disciplines and, in many cases, the findings of the corpus linguist have much to offer to the theoretical linguist.

It is therefore important that corpus linguistics is seen as what it is: a complementary methodology from which many useful insights can be gained.

The other resentment against corpus linguistics is that its application is strongly based on computers. If a linguist would like to conduct a quantitative study, the use of computers is unavoidable. While being told about this project, a fellow student said: “So you are constructing what I am not even capable of using?” I understand her concern. It is not that she does not know how to handle computers or how to analyze empirical data. If confronted with conventional corpus query systems, a linguist is at least challenged into learning a query language in order to retrieve the data desired. This takes for granted some knowledge about the corpus structure and the technical handling of the query. I am convinced that if confronted with a more intuitive corpus query system, the enigma of this field would disappear quickly.

Having elaborated possible resentments against corpus linguistics, I would like to argue why it should be included in linguistic studies as much as possible. The background of this discussion is a project of the German Department at the University of Zurich. In this project, a team is creating a CD-Rom which will be attached to the mandatory study book for all introductory linguistic courses ([Linke et al. 1996]). The purpose of this CD-Rom is – among others – to optimize the process of studying through additional media and to stimulate the students’ analytical capabilities as well as their critical access to linguistic theories². It seems to me that this CD-Rom will be a great opportunity to make corpus linguistics accessible to these students. Through an intuitive interface students will be able to conduct their own researches on a corpus according to the theory presently studied. It would allow them to test linguistic theories on real data and enable them to gain more insights to the problems of the study of language in general.

The project described in this thesis is the construction of a corpus query tool designed for beginners in corpus linguistics. Whether the CD-Rom will include a link to it is at this point of time not decided. It will, however, serve as a corpus query tool for students in projects in corpus linguistics and can be adapted to other purposes as well. Its main purpose is to allow a user to retrieve data from corpora in an easy-to-handle way.

To construct such a system, several areas of research have to be explored. They include questions of data storage, data retrieval and data output. For all of these steps, a decision had to be made in order to implement a new corpus query tool. In this paper I will show how other projects proceeded and which results they yielded. Based on this knowledge, I will argue my position and present my corpus query system. Last but not least, this system will be analyzed and evaluated, and suggestions for improvement made.

²For a more detailed description of this project see <http://www.unizh.ch/~lingued>

Chapter 2

Components of Corpus Query

When working with CORPORA, it is necessary to have some knowledge about the linguistic as well as the technical resources involved. First of all, there are different kinds of corpora with different kinds of uses. Depending on these uses, corpus query systems with different corpus query languages are constructed. In this chapter, I would like to present a survey of corpus types and formats. I will then move on to a theoretical part about corpus query languages and corpus query tools which will be concluded by the presentation of some corpus query tools.

2.1 Use and Types of Corpora

2.1.1 Corpus Design

Commonly, a corpus is a machine-readable body of authentic texts. This definition, however, needs to be improved because it does not state anything about the purpose or of the design of the corpus. According to Biber ([Biber et al. 2000]:246), a corpus is “not simply a collection of texts. Rather, a corpus seeks to represent a language or some part of a language.” An incoherent compilation of texts therefore does not embody a corpus. Additionally, every corpus is constructed with the purpose of giving insights to one or several research questions. The answer to these questions obviously influences the design of the corpus. Based on all these consideration, corpora are best defined as collections of texts representing some part of a language in order to answer specific research questions.

Whereas the purpose of a corpus is made out in the context of its research project, several criteria are needed to distinguish among the various types and characteristics of corpora. These criteria include linguistic factors as well as technical aspects. The following list of criteria is gathered from several introductory books for corpus linguistics and does not claim completeness. Rather, it gives an overview of the complexity of issues in corpus design. A discussion of the keywords presented in the list follows in the subsequent paragraphs.

- size of corpus (number of signs, tokens, or sentences);
- text selection:
 - written vs. spoken texts;
 - date of texts (synchronic vs. diachronic);
 - genre and REGISTER selection;
 - multi-purpose vs. special-purpose text selection;
 - representativeness of texts (balanced vs. unbalanced, homogeneous vs. heterogeneous);
 - language(s) of texts (monolingual vs. multilingual text selection, parallel vs. comparable text selection);
- additional linguistic information:
 - plain text (unannotated) corpus;
 - TAGGED corpus;
 - tagged and PARSED corpus (so-called TREE-BANKS);
 - other kinds of tagging: semantically tagged, discourse tagged, pragmatic/stylistic tagged, problem-oriented tagged corpus;
- additional context information:
 - identity of author, speaker, or addressee (including sociolinguistic variables such as gender, age, education, dialect, and social context);
 - recordings of prosody, facial expressions, or gestures.

The first feature of a corpus is usually its size. It is declared in the number of words, sentences, or tokens. Meyer ([Meier 2002]:32-34) states that the size of a corpus depends first of all on the resources available (funding, research assistants, computing facilities) and only secondarily on the research question. It is clear, however, that research projects involving linguistic structures which do not occur in as often as others (e.g. conditional clauses) rely on larger corpora in order to gather the necessary amount of information about the feature observed. Contemporary corpora vary from some hundred thousand words of spoken text (London-Lund corpus, British National Corpus) to around half a billion words of written text (Bank of English Corpus). Most corpora will be set somewhere in this range. Corpora of written texts are usually larger than corpora of spoken text.

When collecting texts for a corpus, a first distinction is made between speech and writing. It is obvious that written data is much easier to obtain. Although human communication is the primary mode of communication, it involves a great amount of preprocessing to obtain a sample

of speech in a machine-readable (i.e. textual) format¹. By means of the availability of corpora and the interest in computational matters, the focus of this paper will be set on written data.

Other decisions for a particular text selection also depend on the research question which is to be answered. If historical comparisons are pursued, a corpus must consist of texts with various dates of origin. This is called a diachronic approach. Its counterpart is the synchronic approach in which issues of language at one point in time are observed.

Even within texts of one language at one point of time the characteristics vary considerably among their genre and register. The term *genre* refers to a text's literary category; REGISTER denotes "a language variety associated with a particular situation of use" ([Finegan 1999]:594). Possible genres are fiction vs. non-fiction writing. Further distinctions within these genres such as science fiction vs. romantic fiction or academic vs. press texts can be made. Possible registers are spontaneous vs. prepared speech or telephone conversations with friends vs. face-to-face conversations. The grade of distinction – which can be specified to an unlimited degree – depends on the purpose pursued.

When compiling a corpus, decisions as to which genres and registers are included have to be taken. A multi-purpose corpus seeks to cover a large part of natural language use in order to serve as many research projects as possible and therefore includes a broad range of genres. Examples for multi-purpose corpora are the ICE-Corpus and the British National Corpus. On the other hand, a special-purpose corpus is developed to serve one distinct research area. For example, the Penn Treebank contains 4.9 million words of different genres and serves as training corpus for taggers and parsers. Although the genre of texts such as from the Penn Treebank may influence the performance of taggers and parsers, the exact distribution of these genres is of less importance than when observing sociolinguistic variables.

The representativeness of texts is concerned with the consistency of the text collection. A corpus is balanced if it represents equal samples of different genres; an unbalanced corpus contains a selection of texts with no importance paid to their genres. A homogeneous corpus consists mainly of one genre of texts, whereas a heterogeneous corpus includes many different kinds of texts. A balanced corpus is therefore usually heterogeneous.

Corpora may consist of texts of more than one language. Such a multilingual corpus allows a linguist to study genre variations in two languages. In order to study issues of translation, the texts in each language of the corpus need to be the same. Such a corpus which contains the same text in different languages is called a parallel corpus. If sentences which correspond to one another in different languages are specifically identified, the corpus is aligned.

As to the additional linguistic information which can be part of a corpus, subsections 2.1.2 to 2.1.5 are devoted to this topic. A paragraph on the semantic tagging of the corpora used in this project can be found in subsection 4.3.1. Discourse and other kinds of tagging are only mentioned for the completeness of the list but not explored any further because they are not part of the corpus which is used in this project.

¹For a more detailed survey of collecting samples of speech see pp. 56-61 and 69-78 in [Meier 2002].

Additional context information depends on its availability and necessity. Whereas the identity of the author, speaker, or addressee will normally be known or reconstructed and included at some point in the research project, recordings of prosody, facial expressions, or gestures require additional technological methods, mostly video recordings. Since the number of such corpora is very small and the focus of this paper is set on text corpora in machine-readable form, this kind of context information will subsequently be neglected in this paper.

A table with an overview of the currently most important English and German corpora and their characteristics can be found in section 2.1.6 on pages 17 and 18.

2.1.2 Corpus Preprocessing

It is obvious that more information can be extracted from a corpus enriched with interpretative and linguistic information than from a raw corpus. The information added to plain text is called markup, and three different kinds of markup may be distinguished: structural markup (descriptive information), part-of-speech markup (TAGGING), and grammatical markup (PARSING). Since structural markup only transforms the textual structure of a corpus into a layout-independent form – it refers to the SGML-markers which are inserted in place of text formatting (e.g. headers, italicized text, etc.) – it is not part of the linguistic annotation but rather an essential element of corpus preprocessing. Sometimes structural markup is called “markup” by itself, whereas a linguistically marked-up corpus is referred to as an annotated corpus.

The gathering of linguistic information could theoretically be done manually in countless hours. Since corpus linguistics is interested in large amounts of data, the automatic or semi-automatic annotation of corpora has developed into an area of research on its own. In this subsection, I will present how corpora are prepared by adding structural markup and which other steps are necessary to bring the corpus into a format required for subsequent steps of linguistic annotation.

Before explaining the various steps of marking up a corpus, I will make some general remarks about the annotation of corpora. In Garside et al. ([Garside et al. 1997]:6-7), Geoffrey Leech lists some standards for corpus annotation which are valuable to keep in mind while dealing with this topic. The following list is a paraphrased version of his maxims:

- The annotated corpus should be revertible to its original state, that is, the raw corpus should be recoverable.
- The annotation markers should be extricable independently from the corpus.
- The following issues concerning the annotation should be documented in a easy accessible place:
 - the guidelines of the annotation scheme;
 - the identity of the annotator(s);
 - indications to the quality of the annotation.
- Keep in mind that corpus annotation is not infallible.
- The analysis should be performed as theory-neutral as possible in order to guarantee a wider acceptance and understanding.
- No annotation scheme is standard because they are created for practical reasons.

Keeping these standards in mind, the first step of a corpus annotation is structural markup. A text may contain various formatting which will be lost when it is transformed into a machine-readable format. Examples for such formatting are font type and size, italicization or boldface fonts, sections, titles, and so forth. One way to keep the information included is to add SGML-labels. The labels are marked by opening and closing brackets (*<label>*), and each element is framed by a start- and end-label (*<label> element </label>*).

The subsequently presented examples are based on the linguistic resources which will be used for my corpus query tool. More information about the corpora can be found in section 4.3.1 on page 51. Their preparation and annotation was part of Martin Volk's habilitation thesis described in [Volk 2001]. Examples (2.1) and (2.2) show a reconstruction of the beginning of an article in the *ComputerZeitung* before and after being structurally marked up.

(2.1) Example of the formatting in the ComputerZeitung

CZ 1+2/1997 S.1

Griechen haben Pech

Brüssel (kg) – Beim ersten Internet-Chat-in von EU-Kulturkommissar Marcelino Oreja mußten die Griechen “leider draußen bleiben”.

(2.2) Example of the ComputerZeitung marked up with SGML-Tags

<LOC> CZ 1+2/1997 S.1 </LOC>

<H2>Griechen haben Pech</H2>

<P><CITY>Brüssel</CITY>

<AU_ABBR>kg</AU_ABBR>

Beim ersten Internet-Chat-in von EU-Kulturkommissar Marcelino Oreja mußten die Griechen “leider draußen bleiben”. </P>

The tags which are inserted are <LOC> (location of the document within the newspaper (volume and page numbers)), <H2> (Header), <P> (Paragraph), <CITY> (document anchor city), and <AU_ABBR> (author abbreviation). Each of them encloses a section and marks its end with the corresponding tag which includes a slash (e.g. </LOC>). A detailed list of all the tags which have been inserted by Volk into the ComputerZeitung as well as the Tages-Anzeiger corpus can be found in Appendix A.

A committee of scholars called Text Encoding Initiative (TEI) developed a set of guidelines as to how SGML (Standard Generalized Markup Language) is to be used to encode a vast range of text types and textual features. For example, TEI divides the text into a header and the text itself, whereas the header contains meta-information about the text such as author, title, date, etc.² It seems, however, that nowadays most frequently used standard for text encoding has become XML (Extensible Markup Language). XML is a subset of SGML and due to its smaller complexity easier to handle. In the year 2000, a document type definition for the standardized encoding of corpora in XML called Corpus Encoding Standard (XCES) was established.³ Please note, however, that the SGML-tags presented in the above example and in the ComputerZeitung and Tages-Anzeiger corpora are not TEI-conform.

Besides labelling the structure of a corpus, additional steps are normally necessary to shape a corpus into the form required by a tagger, namely TOKENIZATION and sentence boundary recognition. Tokenization refers to the segmentation of a text into words. A token is automatically recognized as the unit usually enclosed by two blank characters or punctuation marks. The intention is that each token represents one morphosyntactic unit. Several cases are problematic:

²The TEI-Guidelines can be found at <http://www.tei-c.org>

³XCES can be found at <http://www.cs.vassar.edu/XCES/>

Figure 2.1: Verticalized sentence of the ComputerZeitung with SGML-tags and clause boundaries

```

Beim
ersten
Internet-Chat-in
von
EU-Kulturkommissar
Marcelino
Oreja
mußten
die
Griechen
“
leider
draußen
bleiben
”
.
<CB>

```

multitwords (e.g. “in spite of” is one preposition), mergers (e.g. “it’s” are two tokens “it” and “is”), compounds (e.g. “post-Cold War” is not to be analyzed as “post-Cold” and “War”). Such exceptions are usually explicitly listed or separately dealt with.

Another task of tokenization is the isolation of punctuation marks. Each punctuation mark is intended to represent one token. One possible procedure is to separate all punctuation marks from the text and to reunite tokens which have been wrongly separated (e.g. numbers (200 ' 000) or internet addresses (www . ifi . unizh . ch)). In many cases, full stops pose problems because they are multi-functional – they represent sentence boundaries and are constituents of abbreviations, numbers, and much more. TEI provides SGML-tags for six types of full stops, including a stop used to end an abbreviation (<stop.abbr>), a stop used to end a sentence (<stop.sent>), and a stop used to both end an abbreviation and to end a sentence (<stop.abse>).⁴ While disambiguating full stops for tokenization, each recognized sentence can be framed by the TEI-conform SGML-tag (<s>). For further processing reasons, the corpus can be verticalized by positioning one token per line.

The result of these preprocessing steps should be a cleaned corpus consisting of a finite number of tokens. Figure 2.1 shows the last part of the sentence in examples (2.1) and (2.2) after tokenization, verticalization, and clause boundary recognition as it has been conducted for the corpora used in this project. Clause boundaries are marked with <CB>.⁵ For a more detailed description of the difference between clause and sentence boundaries, see section 4.3.1.

In such a format, the corpus is ready to be tagged. There is one final remark to be made

⁴More details about the TEI-encoding of punctuation marks can be found at <http://www.tei->

about preprocessing. Preprocessing seems to be a rather trivial matter. Mistakes, however, will percolate through the whole corpus analysis. If, for example, a sentence boundary is marked incorrectly, the parse of a sentence may be wrong and a correct syntax analysis will never be found. It is therefore a significant matter to have clean and correct data to work with.

2.1.3 Part-of-Speech Tagging

When TAGGING a corpus, a part-of-speech tag is automatically assigned to each token. The range of possible tags is determined by the selection of a TAGSET. A tagset constructed for computational purposes will usually look different than one intended for manual analysis. This is due to the fact that a tagger relies on the immediate local context of a word and cannot take remote information into account (e.g. a distinction between the English indicative and subjunctive is not possible without further context information because both verb forms are the same). Tagsets intended for computerized use may distinguish between different subclasses of larger word classes, thereby increasing the number of tags. Such a tagset allows for example a distinction of different pronouns such as possessive pronouns (e.g. *my*) and personal pronouns (e.g. *I*). It has been found out that – despite one’s intuition might differ – a larger tagset does not influence tagging accuracy negatively but rather leads to increased correctness ([Meier 2002]:91). Current tagsets vary from 30 to 200 members.

There are two kinds of tagging algorithms: probabilistic tagging and rule-based tagging.⁶ Probabilistic tagging (also called stochastic or statistical tagging) works with Hidden Markov Models which determine the most likely tag in a given context. Rule-based (e.g. Brill taggers) assign a tag to each word and in a next step go through a set of rules extracted from a training corpus to correct the mistakes. Both algorithms rely on a training corpus to generate the rules.

Generally, taggers assign the correct tag in about 95% of all cases. Volk and Schneider ([Volk and Schneider 1998]) found that in a comparison of a stochastic and a rule-based tagger the stochastic tagger achieved slightly better results for German texts.

After having been tagged with a tree-tagger for German ([Schmid and Kempe 1996]) applying the Stuttgart-Tübingen tagset ([Schiller et al. 1999]), the sentence presented in examples (2.1) and (2.2) and in figure 2.1 looks as in figure 2.2.

Figure 2.2 shows how the STTS distinguishes between different types of nouns (NN are regular nouns (e.g. *EU-Kulturkommissar*); NE are proper nouns (e.g. *Marcelino Oreja*)), different types of verbs (VMFIN labels the infinitive of a modal verb form (e.g. *mußten*); VVINFIN the infinitive of a regular verb form (e.g. *bleiben*)), and punctuation marks (e.g. \$(and \$.). Unknown words, however, always pose a problem to a tagger. In this example, *Internet-Chat-in* is marked as an adjective although it should have been recognized as a noun. The following subsection

c.org/P4X/CO.html#COPU

⁵The TEI-conform SGML-tag for clause boundary is `<cl>`.

⁶For a general introduction to tagging and tagging algorithms see [Jurafsky and Martin 2000]:287-321.

Figure 2.2: Verticalized sentence of the ComputerZeitung with SGML-tags, clause boundaries, and part-of-speech tags

Token	Pos-tag
Beim	APPRART
ersten	ADJA
Internet-Chat-in	ADJD
von	APPR
EU-Kulturkommissar	NN
Marcelino	NE
Oreja	NE
mußten	VMFIN
die	ART
Griechen	NN
“	\$(
leider	ADV
draußen	ADV
bleiben	VVINF
”	\$(
.	\$.
<CB>	

about morphological analysis and lemmatization shows in more detail how this mistake happened.

The part-of-speech tagging in figure 2.2 is similar to the part-of-speech tagging of the corpora used in this project.

2.1.4 Morphological Analysis and Lemmatization

Depending on the purpose of a corpus analysis, the annotation may additionally include information to the morphological components and the LEMMAS of the words. Both are usually obtained from a morphological analyzer and may contribute – besides adding linguistic information – to a more refined analysis of the syntactic structure.

Figure 2.3 shows how each word of the example sentence is enriched with morphological information and a LEMMA. A lemma is the base form of a word representing all word forms belonging to the same word (e.g. *begin*, *began*, and *beginning* are lemmatized to *begin*). This information can be used to study the overall behavior of a word based on all its different forms.

The morphological analysis in figure 2.3 was done by GERTWOL.⁷ GERTWOL is a system for the morphological analysis of German words based on two-level morphology. It segments

⁷For more information about GERTWOL see <http://www.lingsoft.fi/cgi-bin/gertwol/>

Figure 2.3: Verticalized sentence of the ComputerZeitung with SGML-Tags, clause boundaries, tags, morph, and lemma

Token	Pos-tag	Morpho.-tag	Lemma
Beim	APPRART	PRÄP ART DEF SG DAT MASK	bei-der
ersten	ADJA	NUM ORD SG DAT MASK	erst
Internet-Chat-in	ADJD		Internet-Chat-in (1+)
von	APPR	pre PRÄP Dat	von
EU-Kulturkommissar	NN	S MASK SG DAT	EU-Kultur#kommissar
Marcelino	NE		Marcelino (?)
Oreja	NE		Oreja (?)
mußten	VMFIN	V IND PRÄT PL 3	müss~en
die	ART	ART DEF PL NOM	die
Griechen	NN	S MASK PL NOM	Griech~e
“	\$(
leider	ADV	ADV	
draußen	ADV	ADV	
bleiben	VVINF	V INF	bleib~en
”	\$(
.	\$.		
<CB>			

the words into their principal stem and the surrounding morphemes. The following segmentation boundaries are possible:

strong segmentation (#) separates elements which can exist as independent words (e.g. *Berg#wiese*, *Schreib#maschine*);

weak segmentation (-) separates prepositions, prefixes, and dependent elements which can never exist as independent words (e.g. *Vor-Schule*, *geo-morpho-log~isch*);

joining element (\) occurs either at the boundary of a compound or in front of a suffix. The stem of the word is usually on the left hand side of the joining element. (e.g. *Ein#famil~ie\n#haus*, *Friede\ns#freund*);

suffix (~) a suffix is a morphological element which is added at the end of a word, contributing information to either flexion or derivation (e.g. *Lehr~er*, *buchstab~ier~en*).

A morphological analyzer also produces information about part-of-speech tags which can be used to correct tagger outputs. Cases in which GERTWOL overruled the tagger are marked as follows:

- (?) GERTWOL does not recognize the word form. The word form becomes the lemma and the part-of-speech tag from the tagger is kept. In figure 2.3 this was the case with proper nouns *Marcelino* and *Oreja*.
- (1) GERTWOL supplies exactly one part-of-speech tag which does not correspond to the part-of-speech tag of the tagger. GERTWOL overrules the tagger; the lemma and the GERTWOL part-of-speech tag are used. This case does not occur in figure 2.3.
- (1+) GERTWOL supplies more than one part-of-speech tag but none of them corresponds to the part-of-speech tag of the tagger. If the part-of-speech tag of the tagger is close enough to one of the GERTWOL tags, the tag is replaced by GERTWOL's tag. If this is not possible, a random GERTWOL part-of-speech tag is selected. The corresponding lemma is in all cases taken from GERTWOL. In figure 2.3, this is the case with the word *Internet-Chat-in*. Since the word was not known to GERTWOL and no part-of-speech tag similar to the tag proposed by the tagger was found, a random GERTWOL part-of-speech tag was selected. In this case, the selection was completely wrong.

In the corpora used in this project, each word was analyzed by GERTWOL as described in this subsection. The only difference between the annotation presented in figure 2.3 and the annotation of the *ComputerZeitung* and *Tages-Anzeiger* corpora is that the morphological tags were not included in the corpora used for this project.

2.1.5 Parsing and Treebanks

There are various manners to PARSE a tagged corpus. The goal of parsing is to find syntactic functions and constituents of a text. The decision as to which type and level of grammatical entities are retrieved is left to the annotator, and various theories and stages are possible. Compared to tagging, parsing is a much more time-consuming undertaking because it involves the analysis of complex and often ambiguous sentence structures. Consequently, an automatic parser achieves accuracy rates around 70-80 percent depending on type of text and definition of correctness.

The fifth item of Leech's maxims (see section 2.1.2 on page 7) declared that the annotation should be as theory-neutral as possible; in reality, this is not the case. In parsing, this is difficult to achieve because a parser always reflects the concepts of an underlying grammatical theory. Accordingly a wide range of formats for syntactical annotation is available. Similar to taggers, parsing algorithms are either probabilistic or rule-based. Corpora are often semi-automatically annotated in order to guarantee both efficiency and correctness. A more detailed discussion of parsing algorithms would, however, exceed the scope of this paper.⁸

A special kind of corpus consisting of a collection of syntactically annotated sentences is called a TREEBANK. The name refers to the encoding of the sentence structure which usually

⁸For a more detailed introduction to syntactical analysis and parsing see [Carstensen et al. 2001]:203-245.

resembles a tree. There are manually as well as automatically annotated treebanks containing partial up to complete syntactic structures. The correctness of these structures depends on the parsing method used. The syntactic structure of a manually annotated corpus is prone to inconsistencies based on the differing interpretation of syntactical ambiguities by human annotators or human errors due to the lack of concentration. An automatically annotated corpus may be consistent in the type of parsing errors but all syntactic structures can be recognized and processed. The annotation of a manually annotated corpus is after all always more correct than the annotation of an automatically annotated one. Data from manually annotated treebanks is used to train parsers and to extract lexical information. Beside these two disciplines, a number of research projects in linguistics also relies on large amounts of annotated material (e.g. the automatic resolution of prepositional phrase attachment ambiguities).

The most important English treebank is the Penn Treebank; the most important German treebanks are the NeGra and the TIGER corpus, whereas the TIGER corpus is a continuation of the NeGra corpus. Figure 2.4 shows how a sentence from the TIGER Corpus is displayed in TIGERSearch. More information about TIGERSearch can be found in subsection 2.2.2.

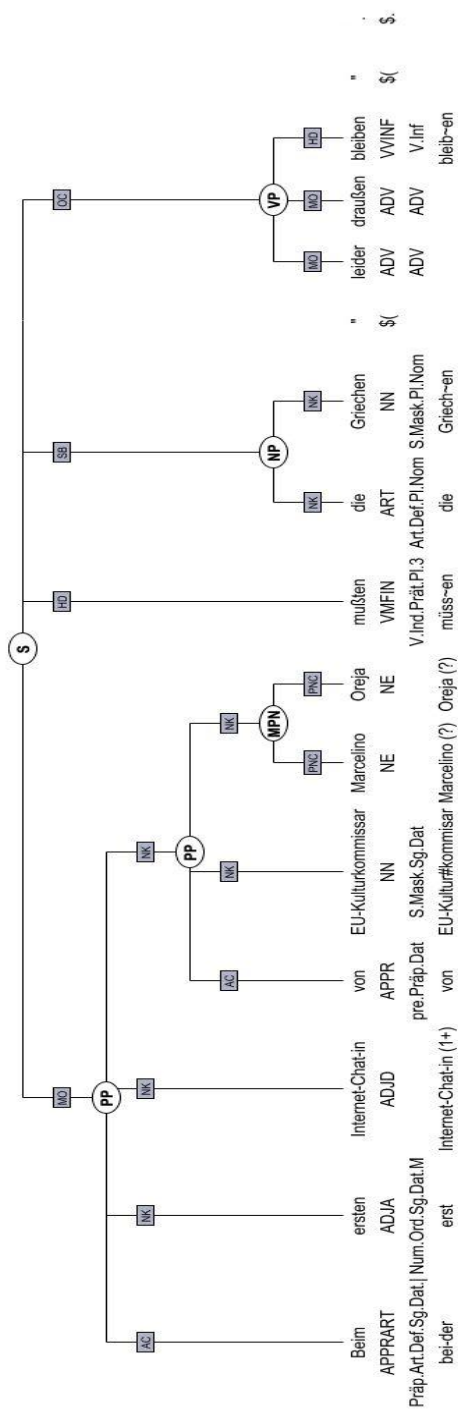


Figure 2.4: TIGER treebank

Figure 2.4 shows the sentence from figures 2.1 to 2.3 with its syntactic tree structure. The root of the tree is the sentence node *S* which branches off into different syntactic constituents until it reaches the words which function as leaves. Each word is annotated with a STTS part-of-speech tag and morphological information and a lemma from GERTWOL. In the ovals, each node is supplied with a tag describing the syntactic constituent. The edges are labelled according to their function within this syntactic constituent. The graphic representation of figure 2.4 is exported from TIGERSearch.

Figure 2.4 shows a complete syntactic annotation of a sentence. One corpus used in this project is annotated likewise; the remaining two are only partial treebanks, meaning that they contain several trees of syntactic constituents which are not necessarily connected to one sentence. More information about the syntactic annotation of the corpora used in this project can be found in subsection 4.3.1 starting on page 51.

2.1.6 Existing Corpora

There is a large number of corpora, some of them commercially available, some of them for purely academic purposes and of restricted use for the respective department only. Copyright issues are always an obstacle to the public availability of corpora. The following tables on pages 17 and 18 gives an overview of the currently most prominent corpora in English and German. The table includes information to their size, characteristics, and annotation. Another row indicates corpus query tools which are available to extract linguistic information from the respective corpus. The information about the corpora was gathered in February 2002 from the internet pages listed in the last row.

2.2 Corpus Query

When in possession of an annotated corpus, the focus will shift from adding information to and encoding to retrieving data from it. Information can be retrieved by using a programming language such as Perl to write programs designed for a particular query purpose. A number of corpus query systems has been built to serve a wider range of uses. They vary considerably in their design, performance, and user-friendliness. In this section, I will show the possibilities of corpus query and corpus query languages. This is followed by a list of five English and German corpus query systems in which information about their system architecture, data storage, query options, and output functions is collected.

There are, however, a number of prerequisites which each corpus query system has to observe. The following list shows which are necessary.

Table 2.1: Overview of English Corpora

Name	Size (in Words)	Characteristics	Annotation	Corpus Query Tool(s)	Information Source
Bank of English Corpus	415 mio.	speech and writing from various genres	wordbank (no linguistic annotation; structural markup)	Collins-COBUILD	http://titania.cobuild.collins.co.uk/boe_info.html
BNC (British National Corpus)	100 mio.	10 mio. spoken words, 90 mio. written words	Pos-Tags	SARA BNCWeb	http://www.hcu.ox.ac.uk/BNC/
British National Corpus Sampler (BNCSampler)	2 mio.	sampler of speech and writing representing the BNC	Pos-Tags	SARA BNCWeb	http://www.hcu.ox.ac.uk/BNC/
Brown Corpus of Standard English	1 mio.	written texts divided into 2000-word samples from various genres	wordbank version/ tagged version	WordSmith	http://www.hit.uib.no/icamente/
International Corpus of English	1 mio.	600'000 spoken words, 400'000 written words	manually checked treebank including complete syntax structures ICE teams for 15 English-speaking countries	ICEUP III	http://www.ucl.ac.uk/english-usage/ice/
London-Lund Corpus	500'000	spoken British English from various genres	prosodic features	WordSmith	http://www.hit.uib.no/icamente/
Penn Treebank	4.9 mio.	American written English	Pos-Tags and (skeletal) syntactical annotation	tgrep online	http://www.cis.upenn.edu/~treebank/home.html

Table 2.2: Overview of German Corpora

Name	Size (in Words)	Characteristics	Annotation	Corpus Query Tool(s)	Information Source
IDS-Textkorpora	1181 mio.	collection of various corpora of spoken and written German	mainly wordbanks	COSMAS	www.ids-mannheim.de/kt/corpora-ges.shtml
Frankfurter Rundschau Corpus	34 mio.	German newspaper texts	Pos-tags, case, lemmas		http://jens.zorba.dk/corp/page18.html
NEGRA Corpus (Nebenläufige grammatische Verarbeitung) Version 2	355'096 (20'602 sentences)	German newspaper texts (selection from the Frankfurter Rundschau Corpus)	manually checked tree-bank with Pos-tags, morphological analysis, syntax (allowing crossing edges)		www.coli.uni-sb.de/sfb378/negra-corpus/negra-corpus.html
TIGER Corpus	35'000 sent. (goal: 55'000 sentences)	written German	manually checked tree-bank with Pos-tags, morphological analysis, syntax (allowing crossing edges)	TIGER-Search	http://www.uni-stuttgart.de/projekte/TIGER

Completeness The corpus query system must guarantee that every occurrence which matches a query is retrieved;

Efficiency The time of retrieval must be limited to a reasonable interval;

Result Reproduction The results of a query must be displayed in a meaningful way which allows a user to find the important structures easily. Basic statistical information such as the number of hits per corpus size is necessary.

Reconstructability The location of retrieved data must be clearly discernible (e.g. corpus, sentence, etc.).

According to Stefan Evert and Arne Fitschen in [Carstensen et al. 2001] (375-376), there are three types of corpus query systems.

1. concordancing systems;
2. pattern-based query systems;
3. systems for statistical analysis.

Concordancing programs retrieve items from a corpus and display the results by showing their immediate context. Items can be words, word forms, phrases, or words restricted by a tag, just to name a few of the most common. The graphical output is often in the form of a KWIC-list (Key Word In Context). A KWIC representation resembles a table containing a left-hand column with context of the given item to the left, a right-hand column with the context to the right, and the key word in a center column in between both contexts.

Pattern-based query systems allow a user to specify regular expressions which find certain patterns. Since these systems allow more complex queries, they are usually used to search for syntactical constituents. The regular expression in example (2.3) shows what a query for noun phrases could look like.

(2.3) (DET)? ((ADV)? ADJ)* NN

The expression matches every instance of a noun (NN) which is preceded by zero or one determiner (DET) and zero or more occurrences of the combination of an adverb (ADV) and an adjective (ADJ), whereas the adverb may occur once or not at all. Depending on the encoding of a corpus, pattern-based query systems may also work with XML-structures.

Systems for statistical analysis are usually based on concordancing programs and give additional information about the frequency of terms in the context of the keyword. They may calculate co-occurrence rates of repeatedly occurring words or part-of-speech tags or find word clusters.

Contemporary corpus query systems usually include one or more components of these three corpus query system types. The type of retrieval is in most cases based on the annotation of a corpus; a part-of-speech tagged corpus will usually result in the retrieval of concordances, whereas it makes more sense to search a treebank with a pattern-based query language. Extensive corpus query systems will make use of both methods. Systems designed purely for statistical analysis are rare because statistical information is an essential part of both concordancing as well as pattern-based query systems.

Stefan Evert and Arne Fitschen in Carstensen et al. ([Carstensen et al. 2001]:375-376) notice a trend of corpus query systems towards a client-server architecture. This makes sense insofar that large amounts of corpus data can be stored on a high-performance server instead of occupying workspace on a local machine. Local systems also have the disadvantage of having to be platform-independent, whereas a client-server architecture allows access by a standard web browser.

2.2.1 Corpus Query Languages

When retrieving data from a corpus, a corpus query language may be used. A corpus query language is a formal construct designed especially for retrieval of corpus structures. Corpus query languages vary considerably depending on the format of the corpus and the queries intended. Some corpus query systems do not make use of corpus query languages but rely on other means of retrieval.

Corpus query languages usually consist of elements to describe the following matters:

- symbols denoting constituents (words, phrases, part-of-speech tags, markup-tags, syntactic constituents, etc.);
- symbols to describe the sequence of these constituents;
- boolean operators to combine (sequences of) constituents;
- further options such as case-sensitiveness, number, etc.

Three of the five corpus query systems presented in the next subsection (ICEUP III, SARA, and TIGERSearch) have designed their own corpus query language. An additional paragraph to each entry of these systems will treat the corpus query language so that the different interpretation of the above-mentioned elements can be seen in the respective corpus query system. In my

corpus query tool, I will not make use of a corpus query language but instead work with input fields which are transformed into SQL query statements.

Another query language which can be said to belong to the group of corpus query languages but which occurs independently of any specific corpus query project is *tgrep*. Tgrep is a Unix query based on *grep* but specifically designed to retrieve data from tree structures. The advantage of *tgrep* over *grep* is that *grep* only searches single lines whereas *tgrep* looks for bracketed tree structures. Tgrep is not designed to be especially user-friendly but focuses on fast retrieval times.

Tgrep can only be used with an encoded corpus file. This encoding is achieved by another Unix utility called *_t_g_r_e_p*. The bracketed *tgrep* syntax structure of a sentence is displayed in example (2.4).

```
(2.4) (TOP (S (NP-SBJ my best friend)
              (VP gave
                (NP me)
                (NP chocolate)
                (NP-TMP yesterday))
              .))
```

Tgrep enables a user to specify a pattern consisting of nodes and relationships between nodes. As a next step, this pattern is matched against the corpus. To formulate a query, a user types a Unix command of the format displayed in example (2.5) (italicized expressions indicate the position of a parameter).

```
(2.5) tgrep options 'pattern' corpus-file.crp
```

The *options*-statement sets user-defined configurations about query execution and result output. As an example, the option *-w* prints the whole sentence instead of only the matching part.

The *'pattern'*-statement is a combination of node names and expressions to specify their position. A pattern consists of node names and symbols to describe the order of these nodes. For example, the symbol *>* marks a parent-child relationship, whereas the child stands on the left hand side of the operator. Similarly, *>>* stands for an ancestor relationship. Besides their vertical order, the horizontal sequence of nodes can also be specified. A dot *.* marks a node followed by its sibling node which stands on the right of the dot.⁹

The last parameter *corpus-file.crp* defines a corpus file which will be searched. Examples (2.6) and (2.7) show two different *tgrep* query statements which retrieve a match in sentence (2.4) (*My best friend gave me chocolate yesterday*).

⁹See <http://www ldc.upenn.edu/ldc/online/treebank/> or a Unix-manual to read more about *tgrep* query statements.

(2.6) `tgrep -w 'chocolate' corpus.crp`

(2.7) `tgrep -w '(NP . NP) > VP' corpus.crp`

The `tgrep` query statement in example (2.6) matches all instances of the word *chocolate*. The query statement in example (2.7) retrieves a sentence containing a verbal phrase VP which immediately dominates two nominal phrases NP in a sibling relationship. In sentence (2.4) this is the case with VP *gave* and NPs *me* and *chocolate*.

Tgrep is a very powerful and fast corpus query tool for the Unix operating system requiring some basic programming skills. It is therefore suitable for experienced corpus linguists pursuing research of complex syntactic structures and working on a Unix computer.¹⁰

There is, however, a wide range of corpus query tools which are more suitable for beginning or intermittent users. The following section presents a selection of five corpus query tools developed for but not limited to English and German corpora and treebanks.

2.2.2 Corpus Query Tools

The selection of corpus query tools presented in this section is based on their quality, availability, and the availability of documentation papers. The following corpus query tools are some of the currently finest systems in this field. All of them focus on one or several goals, including but not restricted to user-friendliness, efficiency, versatility, and design. Each entry is divided into several subparagraphs about the topics system architecture, data storage, corpus query language, query options, output functions, and a conclusion. If information to one or several topics is not available, the paragraph is omitted.

¹⁰Persons who would like to give `tgrep` a try may do so at <http://www ldc.upenn.edu/ldc/online/treebank/> where the Penn Treebank can be searched online with `tgrep`.

SARA

SARA (SGML-Aware Retrieval Application) is a query system for the British National Corpus and the BNC Sampler.¹¹ Developed in 1996, its goal was “to make the corpus available and usable as widely as possible” ([Burnard 1996]:1). It has become a convenient program for an intermittent user which can be downloaded as a trial version one month for free or purchased for one year. SARA is specialized to acknowledge the SGML-structure of the BNC, thus allowing queries for SGML-tags or in distinct SGML-sections.

Architecture SARA is designed in a client/server-architecture. It offers platform-specific client programs which interact with the SARA protocol and a server program. The SARA protocol transforms (if necessary) user queries into a Corpus Query Language (CQL) which is subsequently processed by the server program.

Data storage Data from the BNC is stored in files and indexed through hashing techniques. The position of each token (including SGML-tags) and its respective part-of-speech tag is stored in an index-file, and for particularly frequent used elements a secondary index is kept in an accelerator-file. According to Burnard ([Burnard 1996]:2), the performance achieved by this proceeding “provides equally good retrieval times for any kind of query”, although McEnery and Rayson ([McEnery and Wilson 1996]:204) criticize that “types of query, which do not match the way the index is organized [...] are much slower”. With regards to the size of the BNC (100 million tagged words, six and a quarter million sentences), SARA achieves fast retrieval times. One has to note, however, that the BNC does not contain lemmas or syntactic information.

SARA Corpus Query Language (CQL) The designers of SARA developed a corpus query language to retrieve data from the BNC. Burnard ([Burnard 1996]:3) characterizes CQL as “a fairly typical Boolean style retrieval language, with a number of additional features particularly useful for corpus work. It is emphatically not intended for human use.” Its elements are atomic queries, unary and binary operators, and scope delimiters. Examples of CQL are displayed in (2.8) to (2.11).

¹¹Information about SARA can be found in the internet at <http://sara.natcorp.ox.ac.uk/> or in [Burnard 1996] or [McEnery and Wilson 1996]:204-206

(2.8) friend, “my best friend”, “friend”=NN1

(2.9) @ friend

(2.10) my * friend

(2.11) my * friend /4

Example (2.8) shows three samples of atomic queries in CQL. An atomic query can be a word, a delimited string, or a word and part-of-speech pair. Other atomic queries are punctuation marks, SGML-tags (e.g. *<pause>* searches for pause-tags in spoken texts), patterns (e.g. “*colo?r*” retrieves all instances of *colour* and *color*), or wildcard characters (e.g. *my _ friend* matches *my best friend* as well as *my only friend*). Example (2.9) shows the CQL unary operator header (@) which searches for matches in the header as well as in the body of SGML-tagged documents. Other unary operators are case-sensitiveness (\$) and negation (!). Example (2.10) shows a binary operator combining two atomic queries. The join operator (*) matches cases where both queries are satisfied in the order specified (i.e. query (2.10) finds occurrences of ‘my’ followed by ‘friend’ at any distance within the <body> of a text). Other binary operators are the sequence operator (blank space between two queries), the join operator # which matches cases where both queries are satisfied in either order, and the disjunction operator | which matches cases where either query is satisfied. Example (2.11) shows the use of a CQL scope delimiter. A join followed by a / operator and a number matches cases where the joined query is satisfied within the number of atoms specified (e.g. “my * friend /4” finds *my very best friend* but not *my ever so needed friend*).

Query options SARA offers the following query possibilities: word query, phrase query, pattern query, part-of-speech query, and SGML-query. Queries can be expressed in the SARA Corpus Query Language as explained in the table above or chosen from a graphical query interface. The most helpful tool to create complex queries is the SARA Query Builder. It comprises on the left hand side a so-called scope node which delimits the query to the scope of a SGML-element. On the right hand side there is the content node which allows to formulate any of the mentioned queries.

Output SARA returns the results to a query in the KWIC-format, thereby either presenting all matches or one sentence per page. A user may chose between four different formats of sentence formatting, namely plain, POS, SGML, and custom. Plain offers no markup, POS allows the user to configure different colors for part-of-speech tags, SGML returns SGML-tags uninterpreted, and custom describes a user-defined mixture of the preceding modes.

SARA conclusion SARA offers useful query options for the British National Corpus, including SGML-aware retrieval. Due to the lack of syntactic annotation is its overall complexity rather

small. Although it offers a formal query language, not every search option is possible. A delexicalized search for a part-of-speech tag without specifying a word is not available because the BNC is not indexed for part-of-speech-tags. Output options are limited to KWIC-format and colors. It seems that SARA – as one of the earliest web-based query systems – is a useful tool but has been improved by BCNWeb, an addition to it offering more features.

BNCWeb

Based on SARA, the SGML-Aware Retrieval Application for the British National Corpus (see previous subsection), BNCWeb offers extended query options for this corpus. BNCWeb's developers focused on user-friendliness and efficiency and achieved both goals. BNCWeb is currently available in its second version.¹²

Architecture: BNCWeb is a web-based query system. It relies on a UNIX-server with a fully installed SARA server program and some other resources. An intermittent user will therefore not seek full installation but rather try to get permission for the web-based access to an already installed system.

Data storage The file structure of the BNC and the SARA query functions remain the same; BNCWeb is an extension to these features. Whereas SARA only allowed downloads of 2000 sentences, BNCWeb does not restrict the number of hits. Moreover, while BNCWeb displays the first fifty hits, the rest of the results are downloaded in the background. BNCWeb also makes use of a relational database system (MySQL) to extend the functionality of SARA to enable post-query options which do not rely on SARA directly.

Query Options BNCWeb offers a wide range of query option and statistical corpus analysis. The most important query functions are query for word, query for lemma, and a corpus browsing function. Queries for word are stated either in the SARA CQL (cf. subsection 2.2.2 for examples) or in regular expressions. Examples (2.12) and (2.13) show possible regular expressions; examples (2.14) and (2.15) show lemma queries.

(2.12) {critici[sz]e}

(2.13) {spr[[^]eo]ngs?i?n?g?}

(2.14) fly VERB

(2.15) fly SUBST

In BNCWeb, regular expressions are indicated with curly brackets. Example (2.12) finds *criticise* and *criticize*; example (2.13) finds *spring*, *springs*, *sprang*, *sprung*, and *springing* but not *spreng* or *sprong*. For a lemma query, the user types in the lemma and selects the lemma type from a list. In examples (2.14) and (2.15), the lemma type is capitalized. Example (2.14) finds all verbal word forms belonging to lemma *fly*, namely *fly*, *flies*, *flew*, and *flying*. If lemma

¹²More information about BNCWeb can be found at the project's homepage at <http://homepage.mac.com/bncweb/home.html> or in a discussion by Rolf Kreyer and Joybrato Mukherjee at www.linguistlist.org/issues/13/13-2840.html

type SUBST is selected as in example (2.15), the query retrieves all instances of word forms belonging to the noun *fly*, namely *fly* and *flies*.

Besides the queries for words and lemmas, texts and genres can be restricted or browsed. Additionally, BNCWeb contains a set of post-query options such as *thin*, *sort*, *collocations*, *delete*, and *save*-functions. It also offers detailed descriptive statistics about the distribution of a keyword.

Output The results of searches are displayed sentence by sentence or in the KWIC-format. Several options are available to arrange the data, including frequency counts and collocation analysis.

BNCWeb Conclusion BNCWeb is a powerful and versatile tool in corpus query, unfortunately yet limited to the use with the BNC World edition.¹³ Due to its use of regular expressions and a number of select-options, the functionality of SARA has been profusely extended. Since BNCWeb is still relies on the original SARA indexing structure, a delexicalized search is still not possible.

¹³BNC Word Edition refers to the second edition of the BNC.

ICEUP III

ICEUP (currently in its third version) is a query system originally developed for the British component of the International Corpus of English (ICE-GB). The ICE seeks to represent the language of currently fifteen English-speaking countries by equally designed but separate components for each country. Each ICE component consists of one million words from both spoken as well as written texts which are fully tagged and parsed and even hand-checked.¹⁴ All components of the ICE can be searched with ICEUP.¹⁵

Architecture ICEUP III is sold as a full version including the complete ICE-GB which can be installed on a personal computer (the program is tested for Windows). There is, however, a free download package with complete query software and a selection of ten texts from the ICE-GB (20'000 words) available.

Data Storage Data from the ICE is stored in files. All features stored in the files are indexed thus allowing fast queries on large and complex corpora. The entry for the spoken sentence *Well you know Chapel Street* is stored as displayed in example (2.16).

(2.16) [`<#5:1:A><sent>`]
 PU, CL(main, montr, pres)
 DISMK, CONNEC(ge) {Well}
 SU, NP()
 NPHD, PRON(pers) {you}
 VB, VP(montr, pres)
 MVB, V(montr, pres) {know}
 OD, NP()
 NPHD, N(prop, sing) {Chapel Street}
 [`<#B>`]

After an identification line with the sentence number, the sentence is classified as a main clause (CL) in the present tense. The discourse marker (DISMK) *well* precedes the subject (SU), a verbal clause (VB) and a direct object (OD). The subject consists of a noun phrase (NP) whose head (NPHD) is a personal pronoun (PRON(pers)) *you*. The verbal clause consists of a verb phrase (VP) in present tense *know*; the direct object of a noun phrase whose head is a singular proper noun (N(prop, sing)) *Chapel Street*.

¹⁴Find more information about the ICE corpora at <http://www.ucl.ac.uk/english-usage/ice/index.htm>

¹⁵Find more information about ICEUP III at <http://www.ucl.ac.uk/english-usage/ice-gb/iceup.htm>

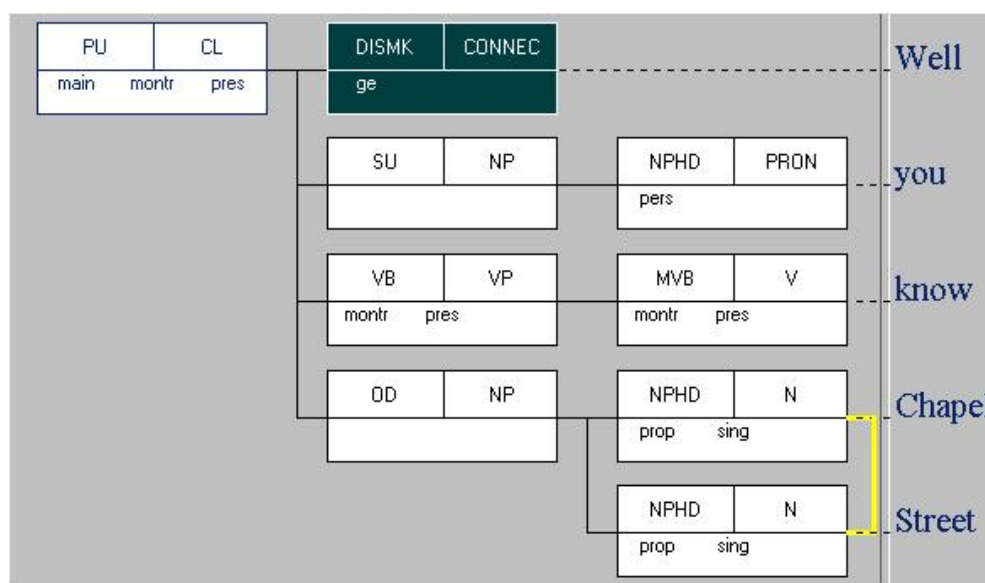


Figure 2.5: ICEUP III Tree Structure

Corpus Query Language ICEUP developed a grammatical query methodology for parsed corpora called Fuzzy Tree Fragments (FTF). FTFs are descriptive formal representations of syntax trees consisting of nodes and text unit element (such as words, pauses, or punctuations). Text unit elements are always at the leaf position of a FTF. FTFs in corpus query are abstract grammatical subtrees which match parts of the complete syntax trees. To specify the position of a node or text unit element, FTFs include unary operators. Binary operators determine the positions of nodes and text unit elements in relation to another.

When transformed into Fussy Tree Fragments, example sentence (2.16) looks as displayed in figure 2.5.

Query Options ICEUP III offers a graphical query interface which enables a user to draw FTFs. Due to its intuitive graphical abstraction of FTFs, it is designed for experienced corpus linguists as well as for beginners. ICEUP III also provides a Creation Wizard which allows cyclic corpus exploration, i.e. the results of a query can be examined and their structure re-used to define a refined query.

Output Query results are displayed as syntactic trees in which the FTF is highlighted or in a line-based concordance view in which a queried node is focused.

ICEUP III Conclusion ICEUP III is a very useful and fast corpus query tool. Installing it is simple and includes a tutorial which explains how to use ICEUP III. FTFs allow a user to formulate complex queries in an intuitive way. Unfortunately, ICEUP is restricted to the use with the ICE.

TIGERSearch

TIGERSearch is a corpus query tool developed in connection with the TIGER Project.¹⁶ This project aims at the creation of a large syntactically annotated German Corpus, involving the participation of the Computational Linguistics Departments of the Universities of Saarbrücken, Stuttgart, and Potsdam.¹⁷ It is a continuation of the NeGra project, with the goal of extending the NeGra corpus and its annotation scheme. The following list describes the main components of the TIGER corpus query system.

TIGER-XML TIGER-XML is a XML-based representation format for syntactically annotated corpora. All existing corpus formats should be convertible into this theory-neutral exchange format which can be processed by TIGERSearch. An example of an annotated sentence in the TIGER-XML format can be found in section 4.3.1. TIGER-XML is described in [Mengel and Lezius 2000].

TIGERRegistry TIGERRegistry is the tool to convert the currently most common corpus formats into TIGER-XML which can be processed by TIGERSearch. Corpora which are already in the TIGER-XML format are indexed and imported to TIGERSearch.

TIGERSearch The TIGER corpus and other syntactically annotated corpora in the TIGER-XML format can be accessed through the TIGERSearch Corpus Query Tool. It consists of a query interface programmed in Java allowing queries for all kinds of word-specific information (e.g. part-of-speech tag, lemma, morphological information, and semantic class) but also to dominance, precedence, and arity of the non-terminal nodes in the syntactic tree. Queries are formulated in the TIGER query and description language or designed by the graphical input tool called TIGERin.

TIGERin TIGERin is the graphical input tool for queries with TIGERSearch. Instead of having to learn the TIGER query and description language, a user can construct queries with the help of graphical representations which resemble the structure of the TIGERSearch output. Each node is represented by a box which contains the necessary constraints. Dominance, precedence, and arity relationships between boxes are indicated by different connectors between the boxes. A graphical query is automatically translated into the TIGER query and description language. TIGERin is described in [Voormann and Lezius 2002].

Corpus Query/Description Language Since the TIGER project aims at the creation of its own annotated corpus, a scheme for syntactic annotation of German texts had to be developed.

¹⁶A short introduction about TIGERSearch can be found in [Lezius and König 2000] or on the TIGER website, located at <http://www.ims.uni-stuttgart.de/projekte/TIGER/>.

¹⁷Before becoming involved in the TIGER project, the IMS of the University of Stuttgart constructed two different corpus query systems named IMS Corpus Workbench and Verbmobil. Since these projects are older and Stuttgart's new focus lies on TIGER, I will not present the other two systems.

It is of special interest to the TIGER people to create an annotation scheme which is as theory-independent as possible in order to be of wide acceptance and re-usability. The representation language will be used as query language as well and therefore requires to be intuitive to the user as well as efficient. The TIGER representation language describes syntactic trees with labelled edges. Crossing edges are allowed in the annotation scheme. There are three levels to this TIGER language: the node level, the node relation level, and the graph description level. On the bottom level, the nodes are filled with feature value pairs and can be combined with Boolean expressions, as examples (2.17) and (2.18) show.

(2.17) [word="fly"]

(2.18) [word="fly" & pos="NN"]

Example query (2.17) finds all instances of the word *fly*, whereas example query (2.18) finds only instances of the word *fly* as a noun.

Node relations are represented by the following key operations: dominance (>), precedence (.), and siblings (\$). The set of these operation is of course extended to enable the description of every possible node relationship. On the graph description level, Boolean expressions (excluding negation) are used to combine node relations. Example (2.19) shows how a query for a verbal phrase VP immediately dominating two nominal phrases NP in a linear precedence relationship in TIGERSearch is formulated.

(2.19) "1": [cat="VP"] &
 "2": [cat="NP"] &
 "3": [cat="NP"] &
 "1" > "2" &
 "1" > "3" &
 "2" . "3" &

In example (2.19), a number is assigned to each node, namely "1" for the verbal phrase VP, "2" for the first noun phrase NP, and "3" for the second noun phrase NP. Verbal phrase "1" dominates both NPs "2" and "3". The first NP "2" stands before NP "3". This query would match the sentence displayed in example (2.4) (*My friend gave me chocolate yesterday*).

Query options TIGERSearch, the corpus query tool of the TIGER project, is programmed in Java in order to support all platforms. It includes features for the selection of a corpus which will be searched, a query window, and folders helping to pose queries. The query is expressed in terms of the TIGER representation language described above.

Output The output is graphically displayed in a program called GraphViewer, where each match to the query is highlighted. The matches can also be exported. The design of the output can be seen in figure 2.4 on page 15.

TIGERSearch Conclusion The TIGER project is by far the most important project in German corpus linguistics. TIGERSearch offers a powerful corpus query language and graphical query output. TIGERSearch is not restricted to one corpus format because TIGERRegistry includes filters to convert the most common corpus formats to TIGER-XML. Users may also write their own filters.

Gsearch

The four system presented before are corpus query tools for annotated corpora. The Gsearch system, however, includes an annotation and a query tool for unparsed corpora in which users can define their own grammar or annotation scheme. GSearch is therefore not only a corpus query system but also a corpus annotation tool.¹⁸

The aim of Gsearch is to provide access to large amounts of text for the investigation of lexical and syntactical phenomena. Corley et al. ([Corley et al. 2001]) argue that the interest in large syntactically annotated corpora increases with the availability of digitalized texts in the world wide web so that a tool which adds syntactic structures to large numbers of texts is indispensable. It is not the intent of GSearch, however, to return perfect syntactical analyses but rather to process large amounts of data. For this reason, GSearch works highly overgenerating, often supplying more than one analysis per sentence.

Architecture GSearch is a platform-dependent program running only on Unix machines. It is designed as a modular system consisting of a corpus in a uniform input format, a bottom up chart parser, and a uniform output format. This allows users to define their own filters in order to process any corpus in the way which they desire. GSearch additionally provides filters for the most commonly used annotated corpora, namely for the British National Corpus, the Brown Corpus, the SUSANNE Corpus, the Wall Street Journal Corpus, the Frankfurter Rundschau Corpus, and the NeGra corpus, so that queries on these corpora can also be conducted.

Query Options Queries on any corpus in the GSearch format are posed in the form of a Unix-command. The general command is of the form displayed in example (2.20) (italicized expressions indicate the position of a parameter; square brackets indicate an optional parameter).

(2.20) `gsearch [options] corpus grammar goal`

The argument *corpus* determines the corpus in which the query is executed. The argument *grammar* specifies a context-free grammar file according to which the corpus will be parsed. The argument *goal* is the query statement consisting of a sequence of terminal and non-terminal symbols corresponding to the grammar which was chosen. Corley et al. ([Corley et al. 2001]) provides the following example query.

(2.21) `gsearch bnc GrammarBNC np pp pp`

The query in example (2.21) retrieves all sentences which contain a nominal phrase followed by two prepositional phrases in the GSearch sampler corpus from the British National Corpus.

¹⁸The project's homepage can be found at <http://www.hcrc.ed.ac.uk/gsearch>

In its original, the BNC is only part-of-speech tagged. With the help of a context-free grammar stored in file *GrammarBNC* GSearch performed a syntactical analysis. Other query options for complex queries are available, including regular expressions, conjunctions, and negation. GSearch provides no operators concerning dominance relations of nodes.

The *options*-parameter in the query statement defines the output of the results. A user may specify which fields of the sentence are to be displayed. The options-statement also includes variables which affect parser and context output.

Output GSearch's default output is a full sentence marked up with SGML-tags. This textual output may be visualized by one of two graphical programs called Viewtree and Thistle which are included in GSearch.

GSearch Conclusion The intention of GSearch is different from the corpus query tools which allow queries on annotated corpora. Whether the results of the automatic annotation of raw text conducted by GSearch are useful depends on the grammar specified and the demands of the user. GSearch is, however, not suitable for corpus linguists without knowledge of the Unix operating system because the parameters have to be configured individually in order to run GSearch. Additionally, the user needs to have a clear-cut idea as to how the corpus is to be annotated, i.e. which grammar is to be used. This requires a strong background in syntactical theories. The advantage, however, is that one may specify one's own grammars and is not forced to use a pre-set annotation scheme. GSearch therefore remains a tool for experienced computational linguists who rely on large amounts of automatically annotated data.

Chapter 3

Database Systems

3.1 Introduction to Database Systems

Database systems are software systems which store large amounts of data. Elmasri and Navathe ([Elmasri and Navathe 2000]:4) define a database as “a collection of related data. By data, we mean known facts that can be recorded and that have an implicit meaning.” Traditionally designed for industrial purposes, it seems that database concepts may be used to store linguistic data as well.

Databases are administered by DATABASE MANAGEMENT SYSTEMS (DBMS). DBMS allow a user to define, construct, and manipulate a database for various purposes. Some DBMS are commercially available (e.g. Oracle, IBM DB2, Microsoft Access, etc.), others are freeware (e.g. MySQL, SAP DB¹). DBMS constructed for heavy-use applications are usually a costly venture. Database and DBMS combined are called DATABASE SYSTEM.

3.1.1 Characteristics and Advantages of the Database Approach

A database approach is characterized by a list of features which I will cover briefly. First of all, data stored in a database is combined with a set of meta-data which describes its definition, structure, and constraints. The encoding of the data will therefore always be reconstructible. Otherwise, data stored in a database is independent of application programs. This independence is accomplished by data abstraction. A conceptual model serves to abstract data by allowing a user to communicate with the data without explicitly accessing physical data storage. Additionally, database systems allow multiple views of the data and multiuser transaction processing.

There are several advantages which arise from these characteristics. First and most importantly, a database system helps to control redundancy. Since all data is modelled according to a

¹cf. <http://www.sapdb.org>

scheme, redundancy can be avoided or deliberately employed to improve performance. Since access to the database can be controlled and data is independent of application programs, multiple uses may be pursued, each being controlled according to the different needs of the application. Moreover, a database systems enforces integrity constraints of the data and provides backup and recovery features for worst-case scenarios.

3.1.2 The Relational Data Model

Before storing data in a database, data has to be modelled according to a scheme. Several data models have been proposed, among them the network data model, the hierarchical data model, the object-based data model, and the relational data model. The relational data model is currently the most widely used. This is due to its relatively simple mathematical foundation and its implementation by the most important database system producers.

The relational data model is based on mathematical relations. A relation resembles a table but – since relations represent mathematical sets – its fields are not in a fixed order. Operations on relations are similar to operations in set theory. The two main categories of operations carried out on relations are retrievals and updates, whereas updates include inserts, deletions, and modifications of the database and retrievals specify selection, projections, renames, and joins.

Each relation consists of a set of columns called attributes and a set of rows called tuples. A minimal subset of attributes must be chosen to form an unambiguous identification of each tuple. This subset is called a primary key. To combine two relations, the primary key of one table must be identical to a subset of attributes in another relation. This subset is called a foreign key.

Probably the most important advantage of relational database systems is a standardized query language for the relational data model called SQL (Structured Query Language). SQL is based on relational algebra but is easier to use than complex mathematical operations. It is a descriptive, set-oriented language which can be used independently as well as embedded in other host languages (e.g. COBOL, PASCAL, C, PHP, etc.). SQL defines statements for data definition and data manipulation. A basic database query in SQL looks as displayed in example (3.1).

```
(3.1) SELECT <list of attributes>  
      FROM <relations>  
      WHERE <conditions>
```

Having these means of data storage and retrieval at disposal, we will now see how database systems have been used in corpus linguistics.

3.2 Database Systems in Corpus Linguistics

It seems that relational database systems are seldom used in corpus linguistics. This may be due to the increasing popularity of the encoding in XML which has become standardized for

corpora and allows efficient data processing for corpus query (cf. section 2.1.2). But why should a concept designed to store large amounts of data not be used and succeed in linguistics as well? Within German corpus linguistics, I am aware of three projects exploring the use of database systems.

3.2.1 Corsica

In his Magister thesis, Marco Zierl ([Zierl 1998]) is pursuing the goal of designing and implementing a database system to store and process corpora. In a first step, he transforms any textual corpus into a TEI-conform SGML-format, going through the preprocessing steps of tokenization and sentence boundary recognition as described in section 2.1.2. Each document is supplied with a TEI-header in which information about the text is encoded.

Zierl, however, does not make use of a commercially available database system but constructs his own. He assigns a number to each word of the text, whereas identical word forms are mapped to the same number. Each token is therefore accessible by an integer of the same size. These integers are indexed to allow faster access. Zierl makes use of binary trees and special search algorithms to retrieve data from the corpus. One of his goals is to ensure that linguistic attributes can be added as well as removed from the corpus, thus allowing the user to chose between any kind of annotation. In the database, information from the TEI-header is stored independently from the text.²

The query language used to retrieve data is based on the query language used for the IMS Corpus Workbench. This language is a precursor of the TIGER query and description language and thus shares certain features with it. It specifies nodes and operators and also allows regular expressions to match a query.³ Corsica is programmed in C++.

3.2.2 San Remo

In his Magister thesis, Thomas Künneth ([Künneth 1998]) poses the question of whether commercial database systems are suitable for use in corpus linguistics. Instead of programming a complete corpus query system, he tests the use and efficiency of a relational database system for corpus import and export, corpus storage, and corpus query. He argues that relational database systems are due to their availability, portability, and proven stability preferable to proprietary systems, and that SQL is a suitable query language for corpora.

There are, however, a few restrictions to using SQL in corpus query systems: Since learning SQL is for most corpus-users not an option, a query language needs to be constructed which

²Since the TEI-header information is very complex, only a part of it is processed in this project.

³For more information about the IMS Corpus Workbench consult Oliver Christ "A Modular and Flexible Architecture for an Integrated Corpus Query System" in Proceedings of COMPLEX-94, pp.23-32.

maps user commands to SQL query statements. After retrieval, SQL-output needs to be displayed in a KWIC-table. Also, additional tools for the import and export of corpora need to be programmed. Künneth therefore proposes a three-layer model for corpus database systems based on relational database management systems: The top-most level consists of communication programs between the user and SQL, including a query language, result output, and import and export filters. The second level manages the exchange of information between the database and the user by translating user commands from the top level into SQL-statements. The bottom level is the relational database itself.

According to Künneth's results, the data scheme underlying the relational database is crucial for the system's performance. His basic idea is to map corpus positions (e.g. words) onto attributes. Since he tests his systems with the British National Corpus, the only attribute available is the word's part-of-speech tag. It is possible, however, to add more attributes, although Künneth does not explore the connection between the number of attributes and performance. With this basic proceeding, a word occurring more than once will be stored several times, thus causing redundancy.

A next step is to keep a running index but map repeatedly occurring word forms onto the same position. A word is therefore characterized by a unique index plus a storage position. Although this approach avoids redundancy, Künneth notes that it causes slow retrieval times with large corpora. If the corpus is small, however, retrieval times are improved compared to the first experiment.

To be able to save textual information such as text boundaries, author, or text genre, additional tables are necessary. Künneth describes the use of the n-table structure in which an additional table is created for each text saving the words which belong to it.

Künneth dedicates a subsection of his Magister thesis to the optimization of corpus queries. The most obvious optimization, generating an index, is costly to maintain with the n-table structure. Künneth therefore omits creating an index file besides the indices which are internally generated by the DBMS. He adds, however, a list in which he saves the number of occurrences of each word and the section of the corpus in which it occurs. This strategy especially improves retrieval times for low-frequency words and still guarantees easy import of additional corpora.

Künneth chose to work with IBM DB2 and Perl. To test San Remo, he programmed an import filter for the SGML-based British National Corpus and tested his system on it. He judged the retrieval times – which depended strongly on the database scheme – as acceptable. The use of relational database system in corpus linguistics is therefore possible.

3.2.3 ANNOTATE Database Format

ANNOTATE is a program to support manual or semi-automatic annotation of corpora with syntactical information.⁴ It is part of the NEGRA project whose goal is to build a large treebank

⁴Find more information about the ANNOTATE project at <http://www.coli.uni-sb.de/sfb378/negra-corpus/annotate.html>

for German. Plaehn ([Plaehn 1998]) discusses the pros and cons of the use of text files versus relational database systems for data storage. Since the NEGRA annotation format allows crossing edges as well as secondary edges, he decided in favor of a relational database system. The relational database management system he chose is Mini SQL.

The emphasis of Plaehn's paper lies on the description of the database scheme. It is designed to support morphological and syntactical information as well as part-of-speech tags. A designated corpus consists of a few thousand to ten thousand sentences. Since the ANNOTATE database scheme has served as basis for my own database scheme, I will describe it in more detail in section 4.2.

Chapter 4

My Own Corpus Query Tool

4.1 Aim, Scope, and Technical Resources

The goal of the project described in this paper is to develop a corpus query tool which enables a user to query partially as well as fully syntactically annotated corpora and presents the query's results. The demands on this corpus query tool are, on the one hand, that it observes all prerequisites listed in section 2.2, namely completeness of query result retrieval, efficiency in retrieving the data, result reproduction, and reconstructability of the results' origins. On the other hand, the corpus query tool is in a final version required to store large, automatically annotated corpora. For this project, however, the aim is set on a prototype version running on syntactically annotated corpora of at the most 20'000 sentences. It will be the task of another project to optimize and improve my corpus query tool to access corpora exceeding 100'000 sentences.

The designated user of this corpus query tool is a beginner in corpus linguistics. This is, on one side, due to a project of the German Department at the University of Zurich in which a tutorial CD-ROM is developed which might include a glimpse on corpus linguistics. Students would be able to answer simple research questions with an easy-to-handle corpus query tool. On the other side, the corpus query tool will make use of a relational database. Instead of requiring a user to learn SQL (which is the standard query language of relational database systems), the corpus query tool includes an interface with different pre-set query types. Each query type is designed as a sequence of commented input fields which constitutes a different query. Other query types than the one proposed in the current version are available by updating the query interface. For the query interface, the emphasis is set on intuitive operation and user-friendliness.

As mentioned before, the corpora will be stored in a relational database. The choice of database management system is MySQL because it is fast and available for free as well as on a server at the Department of Computer Science at the University of Zurich. MySQL supports most features of ANSI SQL.¹ Some SQL features, however, were deliberately omitted to improve

¹SQL was standardized by a joint effort of the American National Standards Institute (ANSI) and the

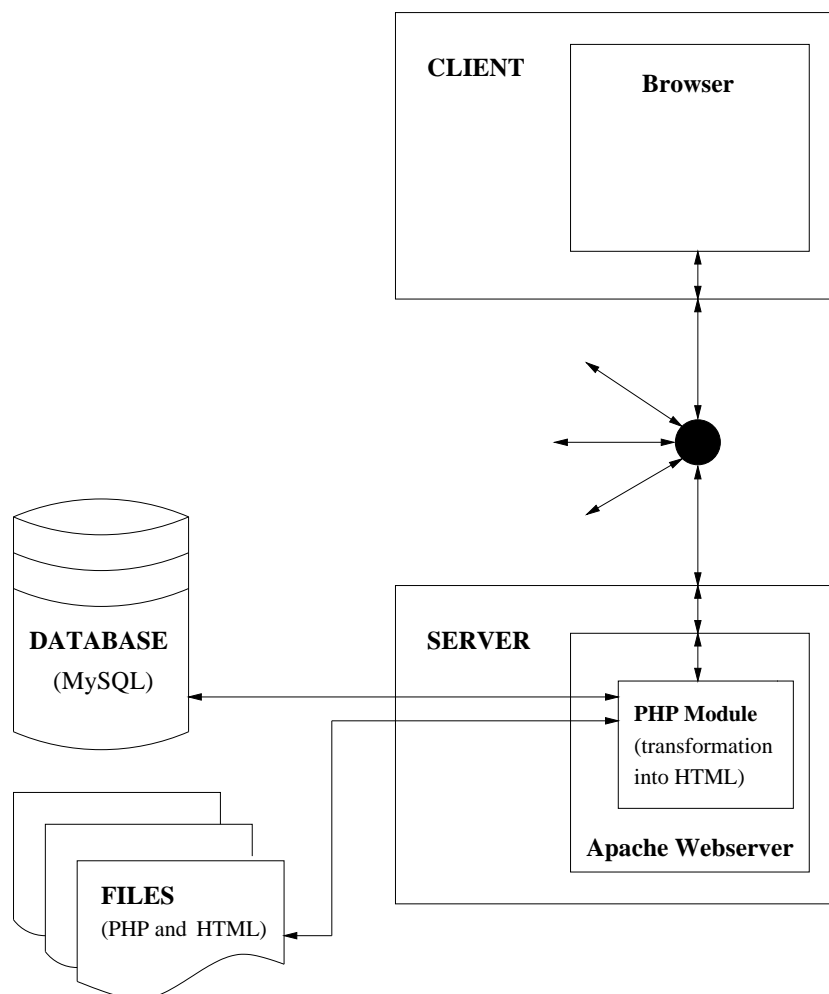


Figure 4.1: Overview of System Components

speed. MySQL is very fast in comparison with other relational database management systems because it stores its data in a B-tree-indexing structure.²

The implementation was done in HTML and PHP. PHP is a scripting language designed retrieve data from MySQL database systems and display embedded HTML on a web page. PHP's syntax is oriented towards the programming language C but is also influenced by elements of Java (object oriented programming) and Perl (string processing and regular expressions). The acronym PHP alternatively stands for **P**ersonal **H**omepage **T**ools or **P**HP **H**ypertext **P**reprocessor.

International Standards Organization (ISO). ANSI SQL refers to the SQL standard defined in 1992 ([Elmasri and Navathe 2000]:244 and [Reese et al. 2002]:7).

²A B-tree is a balanced tree structure designed for fast queries in a dynamic situation. The balancedness of a B-tree is kept by retaining a number of spare tree nodes which take up nodes after insertion. Searching or inserting a node

Figure 4.1 shows a schematic overview of the architecture of my corpus query tool. It is based on a client/server structure in which the client accesses the server through a regular internet browser and the world wide web. The server is equipped with an Apache web server which includes a module to interpret PHP-scripts.³ The PHP-scripts may contain SQL-statements to manipulate the MySQL database. In the corpus query system, a user triggers the respective SQL query statements to retrieve information from the database by submitting elements in a HTML-form to the server. The results of a query are processed in PHP. Instead of displaying the PHP-source code, the result of this conversion is displayed in a HTML-page which is then submitted to the client.

In the subsequent sections of this chapter, each component of my corpus query tool is treated separately and in more detail. The core component of the corpus query tool is its database format. For testing purposes, this database is filled with three different corpora whose characteristics are described in another section. This is followed by a description of query proceedings and interface functions. As a last component, the result output is presented.

4.2 Database Format

In the following section, I will present two different database formats for my corpus query system. The first approach follows the textbook model to produce a relational database by means of the entity-relationship model; the second is an adaption and extension of Oliver Plaehn's database model designed for corpus storage as described in [Plaehn 1998] which I will call N-table approach.

4.2.1 Entity-Relationship Model Approach

When designing a database, a database administrator will follow a sequence of four steps which result in the implementation of a database. The four steps are *requirements analysis*, *conceptual design*, *logical design*, and *physical design*. The following subsection comments on these steps and shows the resulting data models based on the database format of my own corpus query tool.

Requirements Analysis As a first step of database modelling, a database designer has to analyze the purpose and requirements of the database application. The goal of this step is to gather information and to clear misunderstandings so that the designer has a set of correct facts to work with.

in a B-tree constituted of N nodes takes a time of the order of $\log N$. It is one of the most efficient data structures for large amounts of data.

³For more information about Apache web servers see <http://www.apache.org>.

The task of my corpus query tool is to retrieve linguistic data from a relational database. The information which is to be stored in the database is defined by the annotated corpus files. Retrieval times should be of reasonable speed; database updates while the system is running are of no concern because once a corpus is transferred to the database system no more changes will be made. The only update operation performed is to remove or upload a whole corpus, and this occurs very seldom.

The persons involved with the usage of my corpus query system are one system administrator and a number of users. The system administrator is granted all permissions to update or delete from the database. The users are only allowed to read from the database and perform no other operations. Their access to the database is restricted to the queries which can be submitted from the web-based query interface. The number of users is not limited because a relational database system is designed to allow multiple users.

A natural language description of the information in the corpus database might look as follows: A corpus consists of words, and several words together form sentences. Each word is associated with a part-of-speech tag. Additionally, a word may be annotated with a morphological tag and semantic information. The semantic information is composed of semantic name tags or semantic type tags. There may also be a partial or complete syntactic structure for each sentence. The graph of the syntactic structure is a tree consisting of a root node, edges, nodes, and leaves. The leaves of the syntactic tree are the words of the sentence. The nodes represent syntactic constituents which are annotated with parentlabels; the edges are annotated with their syntactic functions called edgelabel. With the exception of the root node, every node has exactly one parent node. If a sentence is fully syntactically annotated, there is exactly one syntax tree; if the syntactic structure is only partially annotated, more than one syntax tree or partial tree exist.

Conceptual Design The conceptual design of a database includes all data requirements and constraints but is independent of any implementation or DBMS. The most widely used conceptual data model for relational database applications is the entity-relationship model. It specifies entity types and their relationships to one another. Such an entity type most closely resembles an independent “thing” in the real world. It can be described by a set of attributes. Relationships may also have attributes but are moreover specified by their degree, i.e. the number of participating entity types.

Based on the above text about the corpus database, three entity types can be made out, namely `word`, `sentence`, and `parent`. Each one of them is a separate linguistic entity. Entity `word` is described by the attributes `part-of-speech tag`, `morphological tag`, and `semantic tag`. A semantic tag can either be a named entity or a semantic type tag and is therefore in an entity-relationship model represented as compound attribute. Entity `sentence` has no attributes, and entity `parent` is described by a `parentlabel`.

The relationship between `word` and `sentence` is that each word belongs to exactly one sentence and each sentence contains one or more words, whereas `word` is used in the meaning of *token*, i.e. the same word form may appear more than once in the same sentence. The same

applies to the relationship between `parent` and `sentence`. Each `parent` belongs to exactly one `sentence`, and each `sentence` contains zero or more `parents`, depending on the completeness of the syntactic annotation. The relationship between `word` and `parent` also depends on the annotation. If the `sentence` is annotated with a complete syntactic structure, each `word` has exactly one `parent` node and the tree ends in exactly one root node. In partially annotated sentences, the syntactic structure is rather a set of nodes and partial trees, meaning that each `word` does not necessarily have a `parent`. The attribute `edgelabel` describes the relationship between `word` and `parent`. Additionally, there is a recursive relationship between `parent` and `parent`. In completely annotated sentences, each `parent` except the root node has a `parent` node; in partially annotated sentences, a `parent` may or may not have a `parent`, thus forming a set of subtrees.

In an entity-relationship schema diagram, entities are represented by rectangles and relationships by diamond-shaped rhombi. Attributes are displayed as ovals which are connected to the entity or relationship which is described by them. Primary key attributes (or identifying attributes) are in bold-face print.

Entities and relationships are connected with lines which may be labelled with their cardinality. The cardinality of a relationship describes the number of associated entity instances in the relationship. The basic types of cardinality are one-to-one, one-to-many, and many-to-many. In diagram 4.2, the cardinality is split up in two parts so that each part can be displayed as a bracketed expression between the participating entity and the relationship diamond next to the relationship. The first number within the brackets specifies the minimal, the second the maximal number of participants.

Figure 4.2 shows how my corpus database may be modelled according to the entity-relationship model.

Logical Design The next step in designing a database is to transform the conceptual data model into a logical model. The logical model of a database depends on the choice of a DBMS because it makes use of the DBMS's particular constructs. There are rules which allow a precise transformation of an entity-relationship model to a logical model called relational model for relational DBMS.

Instead of listing every rule which is needed to achieve the transformation from an entity-relationship model to a relational model⁴, I will comment on a selection of the most important correspondences. An entity type is transformed into a relation. Graphically, a relation corresponds to a table by the name of the entity type. Attributes are added as columns. In the corpus database, this results in the three relations `word`, `sentence`, and `parent` with the respective attributes. Relationships between entity types are included in these relations by foreign keys referring to the corresponding attributes in the other relations. The degree of the relationships may be included.

⁴For a complete listing of the rules for entity-relationship to relational model mapping see [Elmasri and Navathe 2000]:290-294.

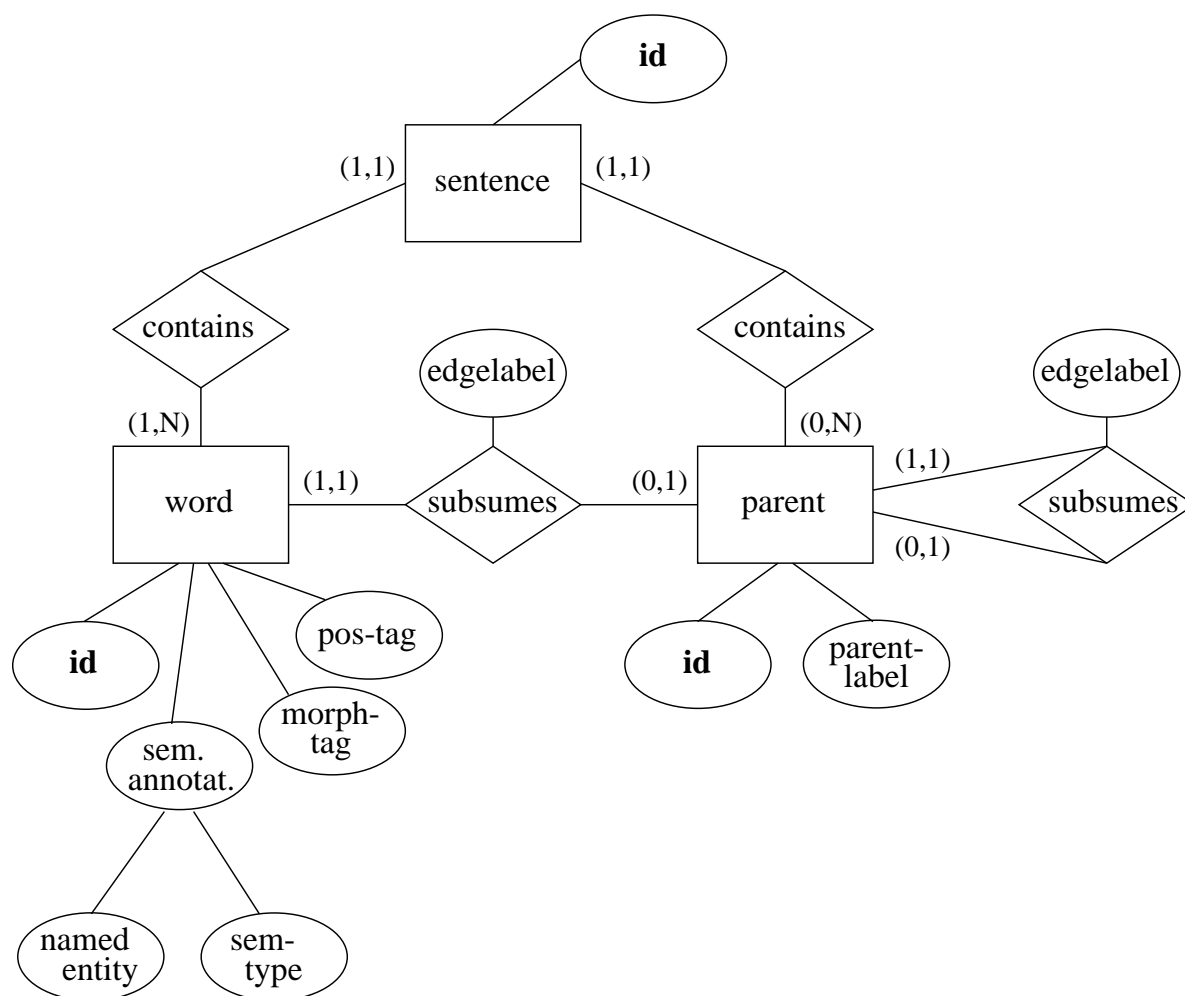


Figure 4.2: Entity-relationship model of the corpus database

A transformation of 4.2 based on these rules results in the three relations displayed in figure 4.3. The name of a relation is in bold-face print. Primary keys are underlined; foreign keys are italicized and are connected to the respective attribute in another relation by an arrow. The relations in figure 4.3 can directly be implemented in a relational DBMS. They constitute the first version fo the database format of my corpus query tool.

Physical Design The last step of database modelling named physical design refers to the task of organizing data storage on a computer hard disk. Internal storage structures, access paths, and file organizations are specified and database queries optimized. Although most commercially available DBMS take care of this matter, some optimizations may be achieved manually. In case of MySQL, frequently accessed attributes are indexed in order to accelerate response time.

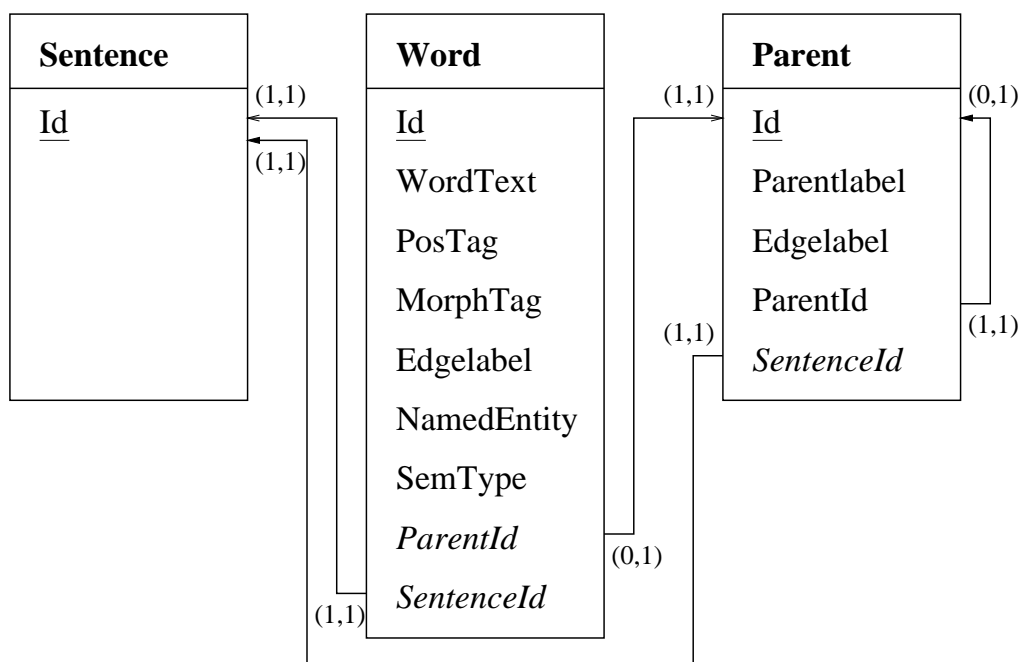


Figure 4.3: Database format (version 1) of the corpus database

4.2.2 N-table Approach

When observing the data which will be stored in the relations `word`, `sentence`, and `parent`, one will notice that many attribute values are redundant. Each finite full verb, for example, is annotated with the five-letter part-of-speech tag `VVFIN`. This redundancy can be avoided if each part-of-speech tag is associated with a unique number. The correspondences between numbers and part-of-speech tags can be stored in a separate table or even in a PHP-script. Since there are only fifty-four distinct part-of-speech tags, the number will in the worst case take up storage space of two digits and can be represented in one byte.

In the N-table approach, an additional table is created for each attribute and the attribute value replaced with an identifying number. Oliver Plaehn's ANNOTATE database format is based on this system ([Plaehn 1998]). He distinguishes thirteen different tables including `word`, `sentence`, `parent`, `part-of-speech tag`, and other replacement tables. I decided to adapt this database format and adjust it to the requirements of my corpus query tool.

Figure 4.4 on page 48 shows the database format of my corpus query tool. All attributes of table `Word` have been sourced out to additional tables.

The database scheme in figure 4.4 relies heavily on Plaehn's documentation of the ANNOTATE database. There are, however, a number of changes which have been made in order to specifically suit my project.

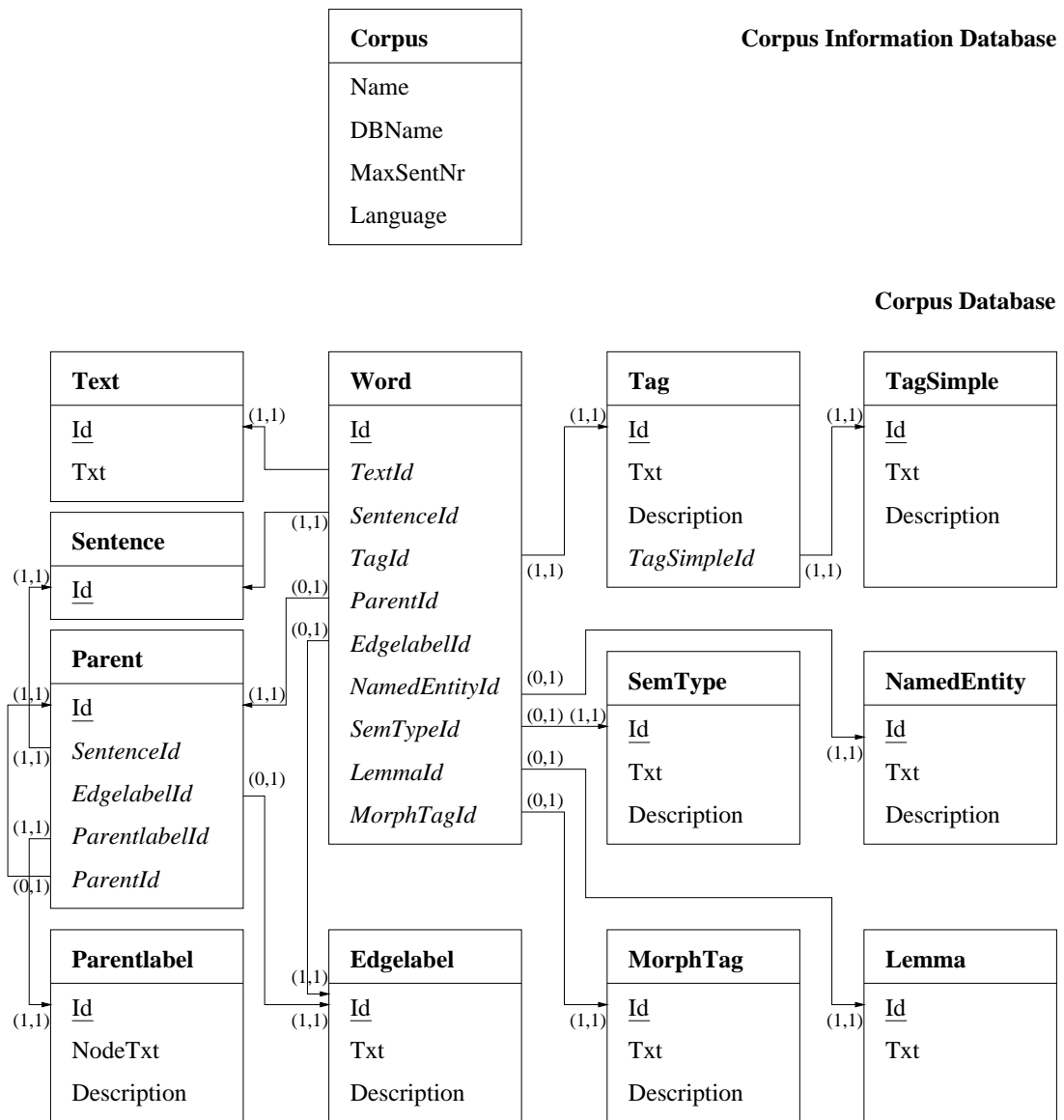


Figure 4.4: Database format (version 2) of the corpus database

- Some tables of the ANNOTATE database scheme are not included in my scheme because they contain information specific to the use of the ANNOTATE tool. The omitted tables are `Origin`, `Status`, `SentComment`, `WordComment`, `SecDependencies`, and `SecEdgelabel`.
- Many attributes in the tables of the ANNOTATEDatabase scheme are not necessary for the `ComputerZeitung` and the `Tages-Anzeiger Corpora` or refer to tables which are omitted as mentioned in the entry above. The following list shows the omissions in detail by specifying a table followed by colon and the omitted items.
 - `Sentence`: `OriginId`, `EditorId`, `LastEdited`, `DepStatusId`, `TagStatusId`, `SentComment`
 - `Word`: `WordComment`
 - `Tag`: `ToBeBound`
 - `Tag`, `Morph`, `Edgelabel`, `Parentlabel`: `Shortcut`
 - `Parent`: `MorphId`
- Two new tables `NamedEntity` and `SemType` for the semantic annotation and the corresponding links are introduced. The distinction between two different types of semantic annotation is made so that each type can be retrieved separately .
- Each table is supplied with an identification number which serves as primary key. In the ANNOTATE database, identification numbers of words and parent nodes are calculated by a special formula which includes the sentence identification number in order to identify the sentence number of each word and parent node. Plaehn ([Plaehn 1998]) adds, however, that a more elegant database format would abandon the calculations and instead introduce an additional foreign key in tables `word` and `parent` which relates each word to a specific sentence. Although the storage space of the keys in the respective tables is doubled and SQL queries complicated, the time for calculations is reduced. In my corpus database, an additional foreign key `SentenceId` is introduced in tables `word` and `parent`.
- Since my corpus query tool is designed for beginners in corpus linguistics, an additional table `TagSimple` containing a simplified version of the Stuttgart-Tübingen tagset is introduced. This simplified version contains a reduced number of tags which cover broader structural categories. Each STTS-tag is supplied with an identifying number of one of these simplified tags. A list showing which STTS-tags are mapped onto which simplified tags can be found in Appendix C.

Plaehn distinguishes between a program-specific database and a corpus database. The corpus database includes the information discussed up to this point; the program-specific database contains information about the corpus itself. It lists its name, editor(s), number of sentences, language, and the date of its last revision. Since this obviously adds a great deal of information, I will organize my database in the same way.

A detailed list of all the tables used in my corpus query tool can be found in Appendix B.

Database Format: Conclusion Kuenneth noted that the design of the database greatly influences the performance of a corpus query tool ([Künneth 1998]). In this section, I have presented two different designs of database formats which can be used for the corpus database, namely the database format based on the entity-relationship model as shown in figure 4.3 and the database format based on Plaehn's N-table approach as shown in figure 4.4. Instead of making assumptions about their performance, I will in the subsequent sections show how I implemented the second database format. It has for a long time been the sole basis of my corpus query tool. A comparison of the performance of the two models will follow in chapter 5.

4.3 Corpora

4.3.1 Resources: ComputerZeitung and Tages-Anzeiger

The following list of corpora shows three syntactically annotated corpora which were used to test my corpus query tool. Further descriptions of their characteristics will be supplied in the subsequent paragraphs.

- 3000 sentences from the ComputerZeitung 1997 automatically annotated corpus;
- 3000 sentences from the ComputerZeitung 1996 manually annotated corpus;
- 20'000 sentences from the Tages-Anzeiger 2000 automatically annotated corpus.

ComputerZeitung 1997 automatically annotated ComputerZeitung is a weekly published computer magazine. Its articles are of similar length as articles of daily newspapers. The focus of the ComputerZeitung, however, is set on topics from information technology. Linguistically, these semi-technical articles are characterized by an increased use of abbreviations and company and product names. It is not a scientific publication but addresses computer professionals. The ComputerZeitung automatically annotated corpus contains 3000 sentences from the year 1997.

The ComputerZeitung corpus was automatically annotated to serve Martin Volk's project of resolving prepositional phrase attachment ambiguities based on unsupervised learning methods. All annotation steps described in the subsequent paragraphs were planned and conducted by him. More information about the procedures can be found in [Volk 2001]:53-88.

For the automatic annotation of the ComputerZeitung corpus, the texts were preprocessed as described in section 2.1.2. Sentence boundaries were recognized and marked with separate lines containing BOS- (beginning of sentence) and EOS-tags (end of sentence). In order to preserve the newspaper formatting, SGML-tags were added to the plain text. The complete list of the inserted tags can be found in Appendix A. A sentence from the ComputerZeitung automatically annotated corpus at this stage looks as displayed in figure 4.5. The sentence is repeated from section 2.1.2.

In a next step, named entities were recognized and classified. This step was completed before part-of-speech tagging because it reveals important information about the category of the words which helps to improve part-of-speech tagging. In the automatically annotated ComputerZeitung corpus, person names, geographical names, and company names were distinguished and marked with an SGML tag. Since most named entities are compounds, a number ascending with each component is added to the tags. The table 4.1 shows which tags for named entities were applied.

Subsequently, the ComputerZeitung corpus was part-of-speech tagged and lemmatized. As mentioned in section 2.1.3, the part-of-speech tagging was conducted with the Tree-Tagger by Helmut Schmid ([Schmid and Kempe 1996]), making use of the Stuttgart-Tübingen tagset

Figure 4.5: Verticalized sentence of the ComputerZeitung automatically annotated corpus with SGML-tags and sentence boundaries

```
#BOS 6
Beim
ersten
Internet-Chat-in
von
EU-Kulturkommissar
Marcelino
Oreja
mußten
die
Griechen
“
leider
draußen
bleiben
”
.
<P>
#EOS 6
```

Table 4.1: Tags for named entities in the ComputerZeitung automatically annotated corpus

person name	<PERS>
geographical name	<GEO>
company name	<FA>

([Schiller et al. 1999]). Likewise, the corpus was lemmatized by using GERTWOL.⁵ Although morphological information was available, it was not included in the final version of the automatically annotated ComputerZeitung corpus because it did not contribute to the analysis of prepositional phrase attachment ambiguities.

Figure 4.6 shows the previous sentence after corpus preparation, named entity recognition, tagging and lemmatization. A person name *Marcelino Oreja* has been recognized and the tags <PERS1> and <PERS2> added to its components. As already noted in section 2.1.4, the noun *Internet-Chat-in* has been tagged incorrectly because it is unknown to the tagger.

Since a complete syntactical analysis of a corpus is always a time-consuming and error-prone matter, Volk decided to conduct a chunk parsing for noun and prepositional phrases. He made use of a pattern matcher which recognizes most common forms of noun and prepositional phrases up to the depth of two levels. The patterns include adjective and conjoined noun phrases.

⁵For more information about GERTWOL see <http://www.lingsoft.fi/cgi-bin/gertwol>.

Figure 4.6: Verticalized sentence of the ComputerZeitung automatically annotated corpus with SGML-tags, sentence boundaries, named entities, part-of-speech tags, and lemmas

#BOS 6			
Beim	APPRART	bei	
ersten	ADJA	erst	
Internet-Chat-in	ADJD	Internet-Chat-in (1+)	
von	APPR	von	
EU-Kulturkommissar	NN	EU-Kultur#kommissar	
Marcelino	NE	Marcelino (?)	<PERS1>
Oreja	NE	Oreja (?)	<PERS2>
mußten	VMFIN	müss~en	
die	ART		
Griechen	NN	Griech~e	
“	\$(
leider	ADV		
draußen	ADV		
bleiben	VVINFINF	bleib~en	
”	\$(
.	\$.		
<P>			
#EOS 6			

In a next annotation step of the ComputerZeitung automatically annotated corpus, prepositional phrases were semantically classified. The semantic categories investigated were local and temporal prepositional phrases. Local prepositional phrases describe a direction or position in space (e.g. *far from home*); temporal prepositional phrases describe a point or duration in time (e.g. *during their vacation*). Their recognition was carried out with the help of a list of prepositions, adverbs, and nouns which are always or often part of a prepositional phrase of this semantic type. Table 4.2 shows which tags for semantic prepositional phrase types were applied in the ComputerZeitung automatically annotated corpus.

Table 4.2: Tags for semantic prepositional phrase types in the ComputerZeitung automatically annotated corpus

temporal PP	<temp>
local PP	<loc>

As a last annotation step, clause boundaries were inserted. Volk ([Volk 2001]:75) defines a clause as “a unit consisting of a full verb together with its (non-clausal) complements and adjuncts.” A clause therefore contains exactly one full verb which need not be finite. It is important to note that there is a distinction between clause boundary recognition and clause recognition. Clause recognition determines the full extent of a clause including its discontinuous elements.

Figure 4.7: Sentence 6 of the ComputerZeitung automatically annotated corpus in the NeGra Export format with SGML-tags, sentence boundaries, named entities, part-of-speech tags, lemmas, NP and PP chunks, and clause boundaries

#BOS 6					
Beim	APPRART	-- --	0	%%bei	
ersten	ADJA	-- --	0	%%erst	
Internet-Chat-in	ADJD	-- --	0	%%Internet-Chat-in (1+)	
von	APPR	-- AC	501	%%von	
EU-Kulturkommissar	NN	-- NK	501	%%EU-Kultur#kommissar	
Marcelino	NE	-- PNC	500	%%Marcelino (?) <PERS1>	
Oreja	NE	-- PNC	500	%%Oreja (?) <PERS2>	
mußten	VMFIN	-- --	0	%%müss~en	
die	ART	-- NK	502		
Griechen	NN	-- NK	502	%%Griech~e	
“	\$(-- --	0		
leider	ADV	-- --	0		
draußen	ADV	-- --	0		
bleiben	VVINF	-- --	0	%%bleib~en	
”	\$(-- --	0		
.	\$.	-- --	0		
<P>	--				
<CB>		-- --	0		
#500	MPN	-- --	0		
#501	PP	-- --	0		
#502	NP	-- --	0		
#EOS 6					

Clause boundary recognition only detects boundaries between clauses but does not identify discontinuous or nested elements. It seems obvious that clause recognition is a much more difficult task and therefore destined for more incorrectness. In the ComputerZeitung corpus, clause boundaries are marked by the SGML-tag <CB>.

The final representation of the ComputerZeitung corpus is based on the NeGra Export format for annotated corpora as described in [Brants 1997]. Figure 4.7 shows the above sentence in its final stage in the NeGra Export format.

The NeGra Export format displayed in figure 4.7 stores one word per line and adds the information of each annotation category in a new column. The order of columns is defined as a word followed by its part-of-speech tag, its morphological tag, and if syntactically annotated its edge label and the number of the parent node dominating it. The last column stores a comment field marked by two percent signs (%%). Since there is no separate column for lemmas or semantic information in the NeGra export format, these annotation categories are stored in the comment section following the two percent signs. Morphological tags and product names are omitted in the final annotation of the ComputerZeitung corpus. Clause boundaries and #BOS-

and #EOS-markers take up separate lines.

The word lines of a sentence are followed by a list of phrase nodes. Each node occupies one line and is identified by its number. The numbers of the phrase node start at 500 in each sentence in order to be distinguished from the numbering of the words which starts at 1. The columns following phrase nodes are of the same order as the word lines. The number of a phrase node is thus followed by its tag (i.e. the parentlabel), its morphological tag, and if further part of syntactical annotation its edge label and the number of the parent node dominating it. Nested syntax structures can thus be stored and identified.

Each sentence in the NeGra export format is surrounded by a line marking the beginning of sentence (#BOS) and a line marking the end of the sentence (#EOS). If a position is not known, the field is filled with two minus signs (- -).

In Volk's project, the NeGra Export format has been used for historical reasons. A more transparent and portable annotation format using XML has been proposed in the TIGER project. Transformed into TIGER-SML, the sentence in figure 4.7 in TIGER-XML looks as displayed in figure 4.8 on 56.

ComputerZeitung 1996 manually annotated The ComputerZeitung manually annotated corpus contains 3000 sentences from the year 1996 which include at least one full verb and one sequence of a noun followed by a preposition. The reason for these selection criteria lies in the fact that Volk needed sentences which contain an ambiguously positioned prepositional phrase so that its attachment to either verb or noun phrase could be determined.

The characteristics of the texts are the same as described for the ComputerZeitung automatically annotated corpus. The difference between the two corpora is that the manually annotated corpus is supplied with a complete syntactic structure which is hand-checked.

The annotation was conducted in a semi-automatical manner. In a first phase, the corpus was automatically preprocessed, part-of-speech tagged, and chunked for nominal and prepositional phrases. This annotation was subsequently manually checked, if necessary corrected and finally completed to a full syntactic sentence structure. Lemmas, morphological tags, clause boundaries and semantical information were not included in the ComputerZeitung manually annotated corpus.

Figure 4.9 on page 57 shows a sentence from the ComputerZeitung manually annotated corpus in the NeGra export format. A tree display of the same sentence can be seen in figure 4.10 on page 58.

Tages-Anzeiger 2000 The Tages-Anzeiger is a daily newspaper based in Zurich covering all regions in Switzerland. It runs approximately 750'000 copies. The texts cover topics from international and national news, local news, weather, culture, sports, and life-style. A daily television program guide and tips for leisure-time activities are also included.

Figure 4.8: Sentence 6 of the ComputerZeitung automatically annotated corpus in TIGER-XML format

```

<s id="s6">
  <terminals>
    <t id="s6_1" word="Beim" lemma="bei-der" pos="APPRART"/>
    <t id="s6_2" word="ersten" lemma="erst" pos="ADJA"/>
    <t id="s6_3" word="Internet-Chat-in" lemma="Internet-Chat-in (1+)" pos="ADJD"/>
    <t id="s6_4" word="von" lemma="von" pos="APPR" />
    <t id="s6_5" word="EU-Kulturkommissar" lemma="EU-Kultur#kommissar" pos="NN"/>
    <t id="s6_6" word="Marcelino" lemma="Marcelino (?)" pos="NE"/>
    <t id="s6_7" word="Oreja" lemma="Oreja (?)" pos="NE"/>
    <t id="s6_8" word="mußten" lemma="müss~en" pos="VMFIN"/>
    <t id="s6_9" word="die" lemma="die" pos="ART"/>
    <t id="s6_10" word="Griechen" lemma="Griech~en" pos="NN"/>
    <t id="s6_11" word=""" lemma="- -" pos="$"/>
    <t id="s6_12" word="leider" lemma="- -" pos="ADV"/>
    <t id="s6_13" word="draußen" lemma="- -" pos="ADV"/>
    <t id="s6_14" word="=bleiben" lemma="bleib~en" pos="VVINF"/>
    <t id="s6_15" word=""" lemma="- -" pos="$"/>
    <t id="s6_16" word="." lemma="- -" pos=".$"/>
    <t id="s6_17" word="</P>" lemma="- -" pos="- -"/>
  </terminals>
  <nonterminals>
    <nt id="s6_500" cat="MPN">
      <edge label="PNC" idref="s6_6" />
      <edge label="PNC" idref="s6_7" />
    </nt>
    <nt id="s6_501" cat="PP">
      <edge label="AC" idref="s6_4" />
      <edge label="NK" idref="s6_5" />
      <edge label="NK" idref="s6_500" />
    </nt>
    <nt id="s6_502" cat="NP">
      <edge label="NK" idref="s6_9" />
      <edge label="NK" idref="s6_10" />
    </nt>
  </nonterminals>
</s>

```

Figure 4.9: Sentence 13 of the ComputerZeitung manually annotated corpus in the NeGra Export format

#BOS 13				
Nach	APPR	--	AC	510
einer	ART	--	NK	510
Befragung	NN	--	NK	510
unter	APPR	--	AC	509
den	ART	--	NK	507
100	CARD	--	MO	500
größten	ADJA	--	HD	500
werbetreibenden	ADJA	--	CJ	505
Unternehmen	NN	--	NK	507
,	\$,	--	--	0
Agenturen	NN	--	CJ	508
sowie	KON	--	CD	508
Zeitschriften	NN	--	CJ	501
und	KON	--	CD	501
Sendeanstalten	NN	--	CJ	501
wird	VAFIN	--	HD	512
dem	ART	--	NK	506
Thema	NN	--	NK	506
“	\$(--	--	0
neue	ADJA	--	NK	502
Medien	NN	--	NK	502
“	\$(--	--	0
auch	ADV	--	MO	503
in	APPR	--	AC	503
Zukunft	NN	--	NK	503
ein	ART	--	NK	504
großer	ADJA	--	NK	504
Stellenwert	NN	--	NK	504
eingerräumt	VVPP	--	NK	511
.	\$.	--	--	0
#500	AP	--	CJ	505
#501	CNP	--	CJ	508
#502	NP	--	APP	506
#503	PP	--	MO	511
#504	NP	--	SB	512
#505	CAP	--	NK	507
#506	NP	--	DA	511
#507	NP	--	CJ	508
#508	CNP	--	NK	509
#509	PP	--	MNR	510
#510	PP	--	MO	511
#512	VP	--	OC	512
#512	S	--	--	0
#EOS 13				

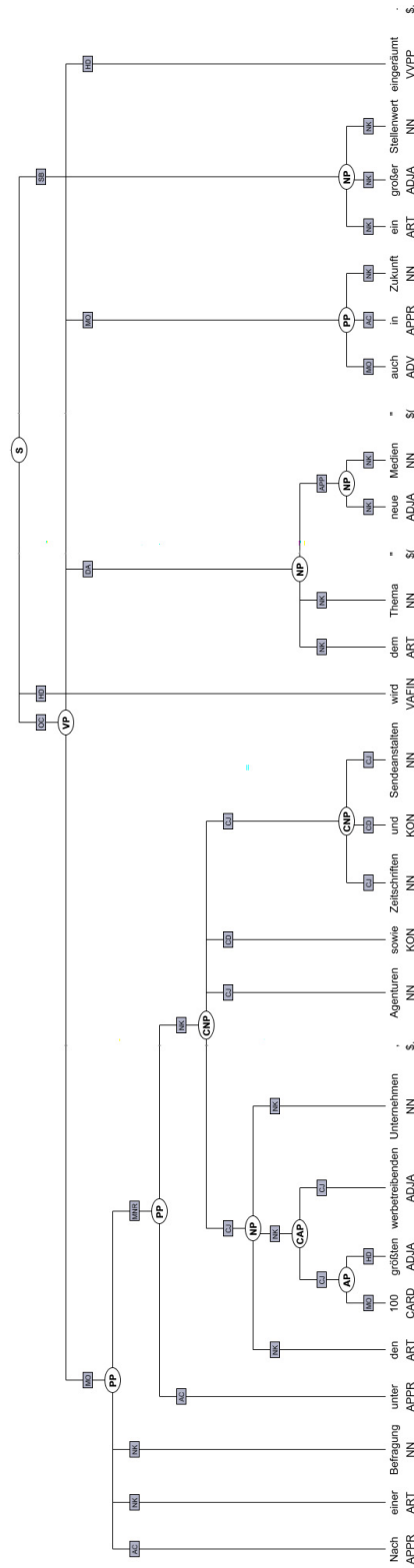


Figure 4.10: Sentence 13 of the ComputerZeitung manually annotated corpus displayed as tree

The annotation of the Tages-Anzeiger was done along the lines of the automatically annotated ComputerZeitung corpus. The only deviation from this procedure was that the format of the Tages-Anzeiger was already marked up with SGML-tags. A complete list of these tags and their meanings can be found in Appendix A.

4.3.2 Corpus Transfer to Database

In order to establish a MySQL-database in the format described in section 4.2, a database has to be created on a MySQL-server and a corpus file has to be parsed and filled into this database. The following programs accomplish this task:

create_db.php creates a MySQL-database on a server;

fill_db.php and **fill_db_manually_annotated.php** take a corpus in the NeGra export format and fill it into an existing MySQL-database.

The program to create a database establishes a connection to a MySQL server through the PHP-function `mysql_connect()`. This function returns an integer which functions as connection handler and is taken up by the PHP-function `mysql_create_db($mysql_db, $db_handler)` which creates a database with the name of the PHP-variable `$mysql_db`. If an error occurs, the process is interrupted and the PHP-function `mysql_error()` returns the respective error message, thus allowing the user to find a mistake in this subtle matter promptly.

The same procedure is chosen for any data which is transferred to the database by means to the function `send_sql($mysql_db, $sql)`. This function first of all transfers an SQL-query `$sql` to a database `$mysql_db` by means of the PHP-function `mysql_db_query($mysql_db, $sql)`. It additionally includes the above-mentioned PHP-function `mysql_error()` in order to facilitate the search for mistakes.

The data transfer to this database occurs in three steps, whereas the first of them is found in the program `create_db.php` and the two ones in `fill_db.php`. The three steps are the following:

1. create MySQL-table structure;
2. fill in annotation scheme (e.g. tags, tag simple, etc.);
3. fill in data from a corpus in the NeGra export format.

1. create MySQL-table structure Each of the thirteen MySQL-table structures which can be seen in figure 4.4 on page 48 is created by a call of the function `send_sql($mysql_db, $sql)` with a different SQL command. All SQL commands of this step are of the format displayed in example (4.1).


```
(4.1) CREATE TABLE Edglabel (
      Id int(2) NOT NULL,
      Txt varchar(4) NOT NULL,
      Description varchar(60) NOT NULL,
      PRIMARY KEY (Id),
      UNIQUE Txt (Txt))
```

The SQL command in example (4.1) creates a table named `Edglabel` with three columns, `Id`, `Txt`, and `Description` respectively. The column named `Id` must consist of an integer value with a maximum length of 2; columns `Txt` and `Description` consist of varied characters, mainly text. According to the SQL-command `NOT NULL`, all fields must be filled with a value. Column `Id` is indexed as primary key of this table, and column `Txt` may only contain unique values in order to prevent double entries. SQL keywords are capitalized to be distinguished from user-defined names.

If the thirteen MySQL-tables specified in the database scheme as displayed in figure 4.4 are established, step one is completed.

2. fill in annotation scheme The data transfer of the second step is similar to the first one. Instead of creating tables, however, step two reads information from additional text files which contain lists with the data of the annotation scheme. If a corpus is annotated in a different format, only a simple update of these text files is required to adapt the changes. An excerpt from the text file which lists the edglabels and their identification numbers and abbreviations used in this project is displayed in example (4.2). It could be filled into table `Edglabel` created with the SQL-command presented in example (4.1).

```
(4.2) 1,AG,Genitivattribut
      2,APP,Apposition
      3,CM,Vergleichskonjunktion
      4,MNR,postnominaler Modifikator
      5,MO,Modifikator
```

Each line in this text file represents a row in a table, whereas the column boundaries are recognized by a column separator, in this case a comma. Table `Edglabel`, to which this excerpt belongs, will therefore be completed with a continuous identification number in the first column, an abbreviation of the edglabel in the second column, and a description in the third column. Text files of this sort exist for tables `Edglabel`, `Parentlabel`, `Semname`, `Semtype`, `Tag`, and `TagSimple`. The SQL command used to fill data from input files into MySQL tables is shown in example (4.3).

```
(4.3) LOAD DATA INFILE '$file_loc/edglabel.txt'
      INTO TABLE Edglabel
      FIELDS TERMINATED BY ','
```

The SQL-command in example (4.3) reads a file from a location specified in the PHP-variable `$file_loc` into a table. A column separator terminating fields is also indicated; here it is a comma. The values of general variables such as `$file_loc` are stored in a separate file called `my_data.php` to guarantee a faster adaption to a different computer. The excerpt of table `Edgelabel` at the end looks as displayed in table 4.3.

Table 4.3: Table `Edgelabel`

Edgelabel	Id	Txt	Description
	1	AG	Genitivattribut
	2	APP	Apposition
	3	CM	Vergleichskonjunktion
	4	MNR	postnominaler Modifikator
	5	MO	Modifikator

If all data from the text files is transferred to tables `Edgelabel`, `Parentlabel`, `Semname`, `Semtype`, `Tag`, and `TagSimple`, step two is completed.

3. fill in data from a corpus in the NeGra export format By far the most complicated step is number three. Information from a corpus in the NeGra export format has to be extracted and filled into the database, whereas the difficulty lies in the fact that the information needs to be extracted from the file and additionally transformed into the form required by the database.

PHP-function `fopen($file, $modus)` opens a specified text file – in this case a corpus in the NeGra export format – and returns a file handler `$fp`. PHP-variable `$modus` is set to “r” to grant read-only access of the file. At the end of the procedure, PHP-function `fclose($fp)` closes the file. As always, each step is framed by functions whose sole purpose is to output a corresponding message in case of an error.

A loop with the PHP-function `fgets($fp, $maxlength)` reads line per line from the file. There are five different types of lines which are distinguished:

1. If the line begins with `#BOS`, a new sentence begins. Extract the sentence number.
2. If the line begins with `#EOS`, the end of a sentence is reached. The line is ignored.
3. If the line begins with `<CB>`, it is a clause boundary marker. A clause boundary is treated like a word and inserted into the respective SQL-table.
4. If the line begins with `#` and a number (e.g. `#500`), it is a node information line which needs special treatment. After removing the character `#`, PHP-function `explode($separator, $string)` transforms the line string into an array, using the tabulator as separator between the fields.

5. All other cases are word information lines. The major part of the information of the corpus is found in these lines. PHP-function `explode($separator, $string)` transforms the line string into an array, using the tabulator as separator between the fields.

Out of these five cases, two have to be observed more thoroughly, namely node information and word information lines.

node information lines Table 4.4 recalls the node information lines in the NeGra Export format as shown in figure 4.9 on page 57.

Table 4.4: node information lines from the ComputerZeitung Corpus

#500	AP	--	CJ	505
#501	CNP	--	CJ	508
#502	NP	--	APP	506
#503	PP	--	MO	511
#504	NP	--	SB	512
#505	CAP	--	NK	507
#506	NP	--	DA	511
#507	NP	--	CJ	508
#508	CNP	--	NK	509
#509	PP	--	MNR	510
#510	PP	--	MO	511
#511	VP	--	OC	512
#512	S	--	--	0

The number in the first column represents the identification number of the node, the second is the label of this node, the third contains morphological information (which was not included in the corpora used in this project), the fourth represents the label of the edge going forth from this node, and the fifth gives the identification number of the parent node.

This line is converted into an array in which each field corresponds to one piece of information in the order described above. The first piece of information, the number of the parent node, is therefore easily extracted. The second, however, poses more problems insofar as the label will not be stored in the form of its abbreviated letter but rather with the corresponding digits. This is displayed in table 4.5 which shows five rows from table `Parent` in which the data from the node information lines shown in table 4.4 is inserted.

Regardless of the identification numbers for node and parent which can directly be extracted from the node information line, an identification number is assigned to each parentlabel and edgelabel tag. In the case of my corpus query tool, this number is extracted from the MySQL database. This is done by an SQL `SELECT` statement as displayed in example (4.4).

Table 4.5: the corresponding lines from table parent

Parent	Id	SentenceId	EdgelabelId	ParentlabelId	ParentId
	500	13	36	2	505
	501	13	36	9	508
	502	13	2	20	506
	503	13	5	21	511
	504	13	27	20	512
	505	13	6	5	507
	506	13	14	20	511
	507	13	36	20	508
	508	13	6	9	509
	509	13	4	21	510
	510	13	5	21	511
	511	13	7	24	512
	512	13	0	23	0

```
(4.4) SELECT Parentlabel.Id
      FROM Parentlabel
      WHERE Parentlabel.NodeTxt=' $parentlabel'
```

PHP-variable `$parentlabel` takes the value of the abbreviated parentlabel from the corpus file. The identification number in table `Parentlabel` (“Parentlabel.Id”) of the line which contains the same letters as in variable `$parentlabel` is extracted and stored in the current place in table `Parent`. In SQL, a column of table is referred to as the name of the table followed by a full stop and the name of the column.

With this procedure all information from the node information lines is gathered and inserted into table `parent`. Example (4.4) stands on behalf of all other numbers which have to be extracted in this way.

word information lines The word information lines basically pose the same problem as the node information lines, although with a slight twist in the extraction of the comment section in the last column. This comment section contains the word’s lemma and semantic information and is preceded by two percent signs. A word information line looks as in table 4.6.

The word information lines, first of all, have been transformed into a PHP-array. The information from each column is stored in one array field. The problem with the comment section can be seen in table 4.6 with the example of the named entity *Marcelino Oreja*. This field contains the word’s lemma as well as the named entity tag. The field containing both information is therefore split into two parts by separating the string delimited by the character `<`. This is accomplished

Table 4.6: Word information lines from the ComputerZeitung Corpus

Beim	APPRART	--	--	0	%%bei
ersten	ADJA	--	--	0	%%erst
Internet-Chat-in	ADJD	--	--	0	%%Internet-Chat-in (1+)
von	APPR	--	AC	501	%%von
EU-Kulturkommissar	NN	--	NK	501	%%EU-Kultur#kommissar
Marcelino	NE	--	PNC	500	%%Marcelino (?) <PERS1>
Oreja	NE	--	PNC	500	%%Oreja (?) <PERS2>

by a number of PHP-functions which extract a substring matching the pattern from the comment string.

The identification numbers of word-texts, tags, edgelabels, semantic name tags, and semantic type tags are extracted with an SQL command similar to the one shown in example (4.4). Identification numbers of words and lemmas, however, have to be created first. Each word is stored in a PHP-variable and compared to the word-texts already stored in MySQL-table `Text`. If the word can be found in the table, its identification number is extracted. If the word is not yet known, a new entry is added to table `Text`. The same procedure is chosen for lemmas which are stored in and extracted from table `Lemma`.

The information thus gathered is inserted into table `Word` which looks as displayed in table 4.7.

Table 4.7: The lines corresponding to figure 4.4 in table `Word`

Word	Id	Sen- tence- Id	TextId	TagId	Parent- Id	Edge- labelId	Sem- Name- Id	Sem- TypeId	Lem- maId
	42	6	42	5	0	0	0	0	19
	43	6	43	1	0	0	0	0	20
	44	6	44	2	0	0	0	0	21
	45	6	11	4	501	10	0	0	11
	46	6	45	17	501	6	0	0	22
	47	6	46	18	500	41	1	0	23
	48	6	47	18	500	41	2	0	24

As can be seen in table 4.7, sentence number 6 begins with the 42th word of the ComputerZeitung automatically annotated corpus. This word has been replaced by a number whose corresponding text can be found in table `Text`. The only word form which occurred before is *von* with the identification number 11.

Since the manually annotated corpus does not contain a comment section and therefore no lemmas or semantic information, a special program called `fill_db_manually_annotated.php` is

available. The only difference between this program and `fill_db.php` is that the first leaves out the comment section. If a user of the corpus query tool would like to input another corpus in a line-based corpus format, an additional program similar to `fill_db.php` has to be written.

4.4 Query

A query statement extracts all sentences which contain a specified element from the requested corpus database. In my corpus query tool, the query input originates from a web-based query interface which is described in section 4.5. The requests from the interface are transformed into an SQL query statement and applied to the MySQL database. The results are edited and displayed in various output functions which are described in section 4.6. In this section, I will comment on the various query possibilities of my corpus query tool (namely simple and complex query), the formulation of SQL query statements, and query improvement strategies.

Since my corpus query tool does not rely on a flexible corpus query language but instead makes use of HTML-based forms to formulate queries, only a selected range of queries is possible. Each type of query is fixed as a series of HTML form input fields. By pressing the “query”-button, the information from these fields is sent to a PHP-program which transforms it into an SQL query statement. Broadly speaking, the information received from the query interface consists of the choice of corpus, the range of sentences which are to be searched, the maximal number of hits, and the searched element(s). If there is more than one searched element, the order of the items is also specified. In some cases the distance between two elements can be specified. It is either a fixed number of words which stand between two elements or a minimal or maximal number of words which may come in between.

4.4.1 Simple Query

As a first step, an SQL `SELECT` statement retrieves the relevant sentence numbers from the database. In the case of a query for one word, the basic SQL query statement looks as displayed in example (4.5). It includes a `JOIN` between tables `Word` and `Text` to extract the word form matching the number stored as *TextId* in table `Word`.

```
(4.5) SELECT DISTINCT Word.SentenceId
      FROM Word, Text
      WHERE Text.Txt='<math>\$word</math>'
      AND Word.TextId=Text.Id
```

The SQL keyword `DISTINCT` removes all duplicate rows for every group of rows which are identical. If the query were formulated without the keyword `DISTINCT`, sentences which contain more than one match would be represented more than once in the final representation.⁶ Variable

⁶For the calculation of the number of hits, however, keyword `DISTINCT` is omitted to count all matches. See section 4.6 for more information about statistics.

\$word contains the letters of the word which is searched. A natural language description of the SQL SELECT statement in example (4.5) might therefore look as follows: Retrieve all sentence identification numbers from table `Word` where the letters of the word match the letters of variable \$word, and delete all duplicates.

The basic query statement can be extended to suit every query type. One query option, for example, allows a user to restrict the scope of sentences queried by specifying a beginning and an ending number. For this query type, the following lines of SQL code are added to the query statement.

```
(4.6) AND SentenceId >= $firstsent
      AND SentenceId <= $lastsent
```

PHP-variable \$firstsent epitomizes the first sentence, \$lastsent the last sentence of the search space. The result will therefore only contain sentence identification numbers which are either greater than \$firstsent or smaller than \$lastsent.

The next step of a query consists of retrieving all words and their annotation which belong to the marked sentence numbers. The extent of the extraction of annotation depends on the output format desired. In my corpus query tool, the output can be simple or complex. A complex output displays the syntactic structure of the sentence as well as the word's part-of-speech tag, the lemma, and its semantic annotation, whereas a simple output only requires the word's STTS and simplified part-of-speech tags.

Whereas up to this point the procedure was absolutely identical for both corpus database formats presented in section 4.2, a clear difference can now be seen. If no syntactic information is requested, all data is readily available in table `Word` of the first database format. In the second version, table `Word` has to be joined with tables `Text` and `Tag` in order to find the word's text and part-of-speech tag. For the simple output format, four JOINS are necessary. Assuming that the marked sentence numbers are stored in a temporary table called `Tmp`⁷, an SQL SELECT statement which retrieves the information for a simple query output looks as in example (4.7).

```
(4.7) SELECT Tmp.SentenceId, Text.Txt, Tag.Txt, TagSimple.Txt
      FROM Tmp, Word, Text, Tag, TagSimple
      WHERE Tmp.SentenceId = Word.SentenceId
      AND Word.TextId = Text.Id
      AND Word.TagId = Tag.Id
      AND Tag.TagSimple = TagSimple.Id
```

⁷An experienced user of SQL might wonder why I am working with temporary tables instead of views or nested SELECT statements. Unfortunately, MySQL does not support either of these two ANSI SQL standards so that other solutions such as temporary tables have to be used (see [Stoll and Leierer 2000]:258 about differences between ANSI SQL and MySQL).

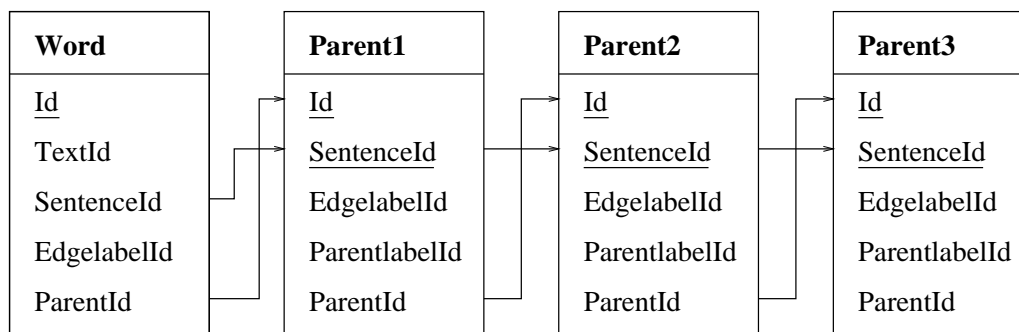


Figure 4.11: joined tables for syntactical structure

The five tables `Tmp`, `Word`, `Tag`, and `Tag_Simple` are joined by means of their foreign keys to result in a table which contains all information needed for the simple output format. A natural language description of the above SQL `SELECT` statement would look as follows: retrieve the sentence identification number, the word's text, its part-of-speech tag, and its simplified part-of-speech tag from a joined table of `Tmp`, `Word`, `Text`, `Tag`, and `Tag_Simple` of all sentence numbers which contain a hit to the query.

Instead of retrieving all data from the MySQL-database, it is possible to store the information to match the identification numbers in hash tables. The advantage of working with hash tables is that the number of time-consuming database access operation can be reduced. On the other hand, if data is permanently stored in programming files, the usage of the database as declarative source of information is flawed. In my corpus query tool, I have not explored the solution with hash tables or other information sources. It is, however, thinkable that this way of data retrieval is faster than performing operations in the database and needs to be explored further.

4.4.2 Complex Query

A complex query is in every case an even more costly matter than a simple query because the output requires additionally to the word's part-of-speech tag its lemma, syntactic structure, and semantic information. Due to the recursive manner of the syntactic structure, a different procedure to extract the syntactic tree structure has to be found. SQL does not provide recursive programming structures, so other means have to be explored. The solution to this problem is that tables can be joined with themselves. Figure 4.11 shows a visualization of this idea.

In figure 4.11, table `Word` is joined to table `Parent` by means of their shared keys `SentenceId` and `ParentId`. Table `Word` already contains the information about the word's parent identification number and the edgelabel connecting the word and the parent node. Through the `JOIN`, additional information about the label of the parent node, the edgelabel connecting the parent node to another parent that is one syntax level above, and the identification number of this

upper parent node are known. One join therefore adds one level of syntax. In order to keep all records of table `Word`, `LEFT OUTER JOINS`⁸ are used.

The resulting table called `Syntax` is added to the corpus database. An excerpt of it is displayed in table 4.8. It shows the syntactic structure of the first part of sentence number 13 from the manually annotated `ComputerZeitung` corpus which was displayed in figure 4.9 on page 57 (*Nach einer Befragung unter den 100 größten werbetreibenden Unternehmen*).

Table 4.8: Table `Syntax`

<code>Syntax</code>	Word- Id	Sen- tence- Id	Parent- Id1	Node- Txt1	Parent- Id2	Node- Txt2	Parent- Id3	Node- Txt3	Parent- Id4	Node- Txt4
	298	13	510	PP	511	VP				
	289	13	510	PP	511	VP				
	300	13	510	PP	511	VP				
	301	13	509	PP	510	PP	511	VP		
	302	13	507	NP	508	CNP	509	PP	510	PP
	303	13	500	AP	505	CAP	507	NP	508	CNP
	304	13	500	AP	505	CAP	507	NP	508	CNP
	305	13	505	CAP	507	NP	508	CNP	509	PP
	306	13	507	NP	508	CNP	509	PP	510	PP
	307	13	0							

Edgelabels are omitted in table 4.8 because they are not required in the output format. In order to further simplify a query statement, identification numbers of parentlabels have been replaced by the letters called `NodeTxt` from table `Parentlabel`. This required four more joins which can be saved in the query statement.

The four columns contain on their left-hand side the parent identification number and the label of this parent. In the corpus database, four more columns is added to table `Syntax` so that eight levels of syntax are discernible. This level is exploited by manually annotated sentences only. Although the syntactical structure of this sentence is correctly reproduced, a graphical output poses problems because elements of syntactic phrases are not necessary positioned in the same column. The constituents of prepositional phrase number 510, for example, are found in the first, second and again in the fourth column. The aligning of these constituent requires complex algorithms which are described in section 4.6.

Besides syntactical information, the complex output of a sentence structure also requires the word texts from table `Text`, the lemmas from table `Lemma`, the part-of-speech tags from table `Tag`, the semantic annotation from tables `NamedEntity` and `SemType`, and finally the relevant sentence numbers from table `Tmp`. A total of seven joins is altogether necessary for the complex query output. It seems obvious that such complicated retrieval actions cannot be fast and possibly need to be optimized.

⁸A `LEFT OUTER JOIN` keeps all records of a table on the left regardless of empty records in the table on the right.

4.4.3 Query Optimization Strategies

Reese et al. ([Reese et al. 2002]:84-93) include a subsection about MySQL query tuning. The main goal of MySQL query tuning is to minimize the number of input and output operations. They list two approaches: indexing and EXPLAIN SELECT, although the second one is rather a tracing device than an optimization strategy.

An index organizes an attribute of a table into a tree structure so that the matching record can be found without having to check every field of the table. It is crucial that all columns of the tables in the corpus database referenced to in a WHERE-clause of a SELECT statement are indexed. Additionally, Reese et al. ([Reese et al. 2002]) suggest to use unique indexes wherever possible because MySQL conducts even more optimization assumptions based on this knowledge. In my corpus database, all relevant columns are indexed. The only disadvantage of indexing is that they are costly to maintain when a table is altered. Since the corpus database – once it is created – is updated infrequently, this is of no concern here.

The other query tuning strategy is a MySQL utility called EXPLAIN SELECT which verifies if a query is executed as expected. It does not improve a query on its own but exposes efficiency flaws in SQL-queries. The main issues which are looked at are the use of indexes and the order of joins. The EXPLAIN SELECT utility does not execute a query but lists in a table – besides other information – which keys are used and how many rows have to be examined. The difference between the execution of a query with or without indexing the relevant columns is immense. It seems that with the use of an index structure MySQL optimizes the corpus queries by itself.

In the programming of my corpus query tool, I have explored one other strategy to accelerate the execution of a query, namely delimiting the number of results which are retrieved. So far a query retrieves and displays all instances of the corpus database which match a query. It is, however, not desirable to display more than ten results per HTML page because the page – depending on the number of hits – becomes very large and takes up too much time for loading. For some complex queries matching a large number of instances it is not even possible to be displayed because the browsers are set on a time limit which is by far exceeded.

SQL offers a command to limit the scope of queries. The line in example (4.8) is added to the end of a SELECT statement.

(4.8) LIMIT start, rows

Parameter *start* determines the record number at which the query is started; Parameter *rows* states the number of results which are to be retrieved. The LIMIT statement therefore allows the presentation of any ten query results from the pool of hits. The managing of the parameters in my corpus query tool is done in PHP.

The limiting of the number of query results per page does not optimize the query itself but improves loading times immensely. The larger the starting parameter grows, the slower becomes the retrieval because all results up to this point have to be calculated. This is, however, of no noticeable importance and duty of the DBMS to perform in a reasonable manner.

Figure 4.12: Screen Shot Wortsuche

4.5 Interface Functions

The elements of my corpus query tool described so far are not discernible to the user. The only part with which a user is confronted is the query interface and the presentation of the output. Figure 4.12 shows a screen shot of the design of the query interface.

Instead of using one large input field for all query types, the corpus query interface consist of a fixed succession of form input fields. If a new query type is desired, it has first to be implemented. The momentary interface is divided into four linguistic query categories and one browsing function. Each linguistic category comprises one or more specific query types belonging to this field. An extension of these categories as well as query types is possible as long as the query statement can be formulated in SQL.

The selection of query types in the version of my corpus query tool includes only the most basic queries. Besides formulating a number of SQL-query statements for one linguistic element of different categories, I have also explored the query for two elements in variable distance. This of course complicates the SQL-query statements immensely especially if distance operators such as *at the most*, *at least*, and *exactly* are included. I have been able to show, however, that these request can be translated into SQL.

The following list shows an overview of the query categories and the respective query types implemented in the current version of my corpus query tool.

- Query for word:
 - Query for one word (simple query);
 - Query for word followed by word in variable distance up to ten;

- Query for part-of-speech tag:
 - Query for simplified part-of-speech tag;⁹
 - Query for word with a simplified part-of-speech tag;
 - Query for a STTS part-of-speech tag followed by a STTS part-of-speech tag in variable distance up to ten;¹⁰
- Query for syntactical constituents:
 - Query for syntactical constituent
- Query for lemma:
 - Query for lemma
- Corpus-browsing:
 - Query for sentence number

In order to facilitate queries which involve a set of expressions such as part-of-speech tags or syntactical constituents, the input terms do not have to be typed in but can be chosen from a list. The list is generated by reading the parameters from the corresponding tables in the MySQL database and embedding them into a HTML select list. Changes in the tagset or annotation scheme can therefore be carried out without having to make changes in the query interface; an update of the respective MySQL table is all it takes.

At the top of the query interface shown in figure 4.12, a number of query parameters can be specified. First, a corpus must be chosen. This is done by choosing a title from a select list. The connection to the database will automatically be set to this corpus database. If a user desires to have more information about a corpus, an information file is accessible through a link at the bottom of the left-hand frame.

The remaining query options concern the part of the corpus which is to be searched. A query can be restricted to a section limited by a beginning and an ending sentence number. Additionally, the number of maximal hits can be specified.

For any problems with the corpus query, an extensive help file is available through a link on the bottom of the left-hand frame. It lists query strategies, descriptions of the different query types, and a list with descriptions to the simplified tagset, the STTS-tagset, and the syntactical categories.

To support students who are beginners in corpus linguistics, the interface includes several functions to reject erroneous inputs. If, for example, a user requests to search more sentences than

⁹For a list of the simplified part-of-speech tags see Appendix C

¹⁰For a list of the Stuttgart-Tübingen tagset see [Schiller et al. 1999].

Neue Suche Suche modifizieren Darstellung mit Tags

Die Suchanfrage nach Wort **Griechen** im Korpus CZ97_ automatisch ergab 4 Treffer (85 Treffer pro 1 Mio. Wörter).
Die Suchanfrage wurde auf 3 Treffer in Sätzen 1 bis 3000 beschränkt; diese Seite zeigt Treffer Nr. 1 bis 3.

Satz	Kontext	Keyword	Kontext
4	<H2>	Griechen	haben Pech </H2> <CB>
6	Beim ersten Internet-Chat-in von EU-Kulturkommissar Marcelino Oreja mußten die	Griechen	" leider draußen bleiben " . </P> <CB>
8	Wenig Glück hatten	Griechen	, <CB> die des Englischen nicht mächtig sind : <CB>

Figure 4.13: Screen shot of the KWIC output

the number of sentences available in the chosen corpus, a statement points to this divergence and the number of sentences is automatically set to the number of sentences in this corpus. The input in the field of the searched element is also controlled and if not matching certain restrictions – such as being a number if a sentence number is searched – rejected with a adequate comment.

4.6 Output Presentation

As mentioned in section 4.4, there are two different kinds of result output presentation, namely simple and complex output. The type of output is determined by the characteristics of the query type. The output of queries involving more than one constituent or syntactic structures is displayed in a table with syntactic and semantic structure; the output of queries for one element or part-of-speech tags are displayed in the KWIC output which can be used interchangeably with the tag-display.

4.6.1 Simple Query Output: KWIC and Tag-Display

The output of a simple query includes two different graphical representations: the sentence is either displayed in a KWIC table or in a line-based format with the word's simplified part-of-speech tag underneath it. A button allows a user to switch from one output format to the other. This is possible because the information required for both representations is retrieved with the same query statement. Figures 4.13 and 4.14 show screen shots of the simple query outputs.

The three buttons at the top of the result display page are for the navigation within the corpus query tool. The first button named “Neue Suche” (*new search*) reloads the query interface so that

The screenshot shows a web interface for a corpus query tool. At the top, there are three buttons: "Neue Suche", "Suche modifizieren", and "Darstellung im KWIC-Format". Below the buttons, there is a text block providing information about the search results, including a link to a list of used tags and details about the search query for the word "Griechen". Three example sentences are shown, each with its corresponding part-of-speech tags. The tags are color-coded and include abbreviations like n.b., Nomen, Verb, Präp, Adj, Art, and Pro.

Neue Suche Suche modifizieren Darstellung im KWIC-Format

Die Hilfedatei enthält eine [Liste der verwendeten Tags](#) mit ihren Abkürzungen.

Die Suchanfrage nach Wort **Griechen** im Korpus CZ97_automatisch ergab 4 Treffer (85 Treffer pro 1 Mio. Wörter).
Die Suchanfrage wurde auf 3 Treffer in Sätzen 1 bis 3000 beschränkt;
diese Seite zeigt Treffer Nr. 1 bis 3.

Satz 4
Text: <H2>Griechen haben Pech </H2> <CB>
Tag: n.b. Nomen Verb Nomen n.b.

Satz 6
Text: Beim ersten Internet-Chat-in von EU-Kulturkommissar Marcelino Oreja mußten die Griechen " leider draußen bleiben " .
Tag: Präp Adj Adj Präp Nomen Nomen Nomen Verb Art Nomen | Adv Adv Verb | |

Satz 8
Text: Wenig Glück hatten Griechen , <CB> die des Englischen nicht mächtig sind : <CB>
Tag: Pro Nomen Verb Nomen | <CB> Pro Art Adj n.b. Adj Verb | <CB>

Figure 4.14: Screen shot display with pos-tags

a new query can be posed. The second button named “Suche modifizieren” (*modify search*) also reloads the query interface but displays query parameters which have already been specified so that a query does not have to be started from scratch again. The third button named “Darstellung mit Tags” (*display with tags*) or “Darstellung im KWIC-Format” (*display in the KWIC-format*) depending on the current output type changes from one output format to the other.

The maximum number of sentences displayed on one page is ten. If the number of results exceeds ten sentences, buttons for result scrolling are displayed at the end of the page. The number of maximum sentences is arbitrary and can easily be changed by assigning a different value to the PHP variable \$limit.

Below the navigation buttons, an information sentence can be found. It displays information about the query, the scope of the query, the number of results, and the numbers of the hits displayed on this page. The information sentence enables a user to check the submitted query and keep track of the result navigation. If the output includes part-of-speech tags, a link to the help-file in which the tags and their linguistic functions is included.

The only statistical information which my corpus query tool provides is the frequency of the queried feature per one million words. In order to calculate the frequency per one million words, the number of hits and the number of words which were queried has to be known. This is accomplished with the help of two different PHP-functions. The first function counts all sentence numbers in the corpus which contain an instance which matches a query. If there are two or more hits in the same sentence, the sentence number is counted repeatedly. It would be wrong to estimate the frequency of the number of hits based on the number of words in the corpus because the corpus may be delimited by several scope delimiters. A second PHP-function therefore counts all words over which the query was conducted. The frequency per one million words is

calculated based on the formula of one million divided by the number of words times the number of hits. The resulting number is rounded to the next integer.

The HTML page displaying the results is supplied with a number of hidden input fields which contain the values of the queried element and the query options. Triggered by the pressing of a button, the information from the hidden input fields is transferred to another HTML page. There it can be processed to display the next ten sentences or the same sentences in a different output format. The statistical information mentioned in the above paragraph is also submitted in the same manner. Each time when a user reaches a new output page, a PHP-functions checks if the frequency per million words is already available. A large corpus is therefore only scanned once for the number of all results so that subsequent result display pages take less time for loading because the information is already known.

In the representation of the sentence the queried element is highlighted by a red-colored font. The highlighting is performed by a pattern-matching algorithm which compares the queried item to every word and in case of a match changes the font color to red. If the queried object is a word, it is often the case that the letters of the query term do not exactly match all instances retrieved because the pattern-matchers operates case-sensitively. This occurs most commonly if the word is capitalized because it occurs at the beginning of a sentence. The queried word is therefore transformed into three different typeface forms, namely first letter uppercase, all letters upper case, and all letters lower case form. This makes the recognition and highlighting of all queried elements possible.

The disadvantage of the pattern-matching strategy to highlight queried elements is that it is highly over-generating. If, for example, the query asks for a certain word directly followed by another one, and the word occurs twice in the same sentence but only once followed by the second word, the pattern matcher regardlessly marks the word twice. A more complex marking algorithm would improve this flaw.

4.6.2 Complex Query Output: Table with Syntactic and Semantic Structure

For queries which involve several elements or syntactic structures, the output is displayed as a table which contains a verticalized sentence and its lemmas, part-of-speech tags, syntactic annotation, and semantic information. This table looks as displayed in figure 4.15.¹¹

The format of the table displayed in figure 4.15 resembles the NeGra Export format but is due to the use of table frames and colors easier to read. In the first column, each word of a sentence is displayed on a new line. The second column shows the part-of-speech tag for each word. If the corpus has been annotated with lemmas, a third column with lemmas is added; else this column is omitted.

¹¹The format of this table was originally designed by Simon Clematide for the presentation of annotated sentences in the NeGra Export format.

Wortform	STTS-Tag	Syntax				
Nach	APPR					
einer	ART					
Befragung	NN					
unter	APPR					
den	ART					
100	CARD	AP				
größten	ADJA		CAP	NP		
werbetreibenden	ADJA					
Unternehmen	NN					
,	\$.					
Agenturen	NN					
sowie	KON					
Zeitschriften	NN					
und	KON	CNP				
Sendeanstalten	NN					
wird	VAFIN					
dem	ART					
Thema	NN					
"	\$(
neue	ADJA	NP				
Medien	NN					
"	\$(
auch	ADV					
in	APPR	PP				
Zukunft	NN					
ein	ART					
großer	ADJA	NP				
Stellenwert	NN					
eingräumt	VVPP					
.	\$.					

Figure 4.15: Screen shot: table with syntactic and semantic sentence structure

The next block of columns varies in its breadth depending on the levels of syntax displayed. Syntactic constituents are represented as vertical bars labelled with their function name. The scope of a constituent can be seen in its vertical length. The leaves of the syntactic tree structure are the words on its left hand side; the root node is the constituent bar on the right hand side of this table. Parent nodes can be found if the table is read horizontally from left to right. Syntactic constituents are dominated by the constituent on the same line on their right hand side. In figure 4.15, the root node is constituent *S*.

In order to distinguish the constituents with the same labelling, each constituent is colored in a different shade. This is not only of importance if constituents with the same name occur, but also if a constituent has discontinuous elements. The different parts of a discontinuous constituent are always placed in the same column but also marked in the same color so that they can be distinguished from other constituents. In figure 4.15, constituent *VP* in the second column from the right has three discontinuous elements, at the end even interrupted by constituent *NP*. In this representation, a user can easily see which parts belong together.

Recalling figure 4.8 on page 68, we note that the syntactic constituents were not vertically aligned in columns but distributed in their horizontal order per line. The length of a line in relation *Syntax* therefore varies according to the number of constituents dominating this word. In figure 4.15, syntactic constituents vary in their horizontal depth so that they can be displayed column-wise. The following list shows the four steps taken to transform relation *Syntax* into the HTML-table with aligned syntactic constituents. Detailed descriptions of these steps and figures illustrating them will be supplied in the subsequent paragraphs.

1. create PHP-array with syntactical constituents;
2. calculate column number for each constituent and insert it into array;
3. calculate horizontal width for each constituent and insert it into array;
4. transform the PHP-array into HTML-code, observing empty and occupied table fields.

In a first step, an associative PHP-array¹² of the following format is created:

⟨sentence number⟩⟨constituent number⟩ {name of constituent, start, end, column, width}

The entry indexed with the sentence number refers to associative arrays indexed with their constituent number. The constituent number is the same number as in the NeGra Output format starting at 500. Each of these constituent arrays contains the information needed to position a syntactical constituent, namely its label, its starting position, its ending position, its column, and its width. In the first step, only the framework of this complex array listing all constituents under their respective sentence number is created and the starting and ending position of each element

¹²Note that in PHP associative arrays take up the function of what is in other programming languages called hash tables.

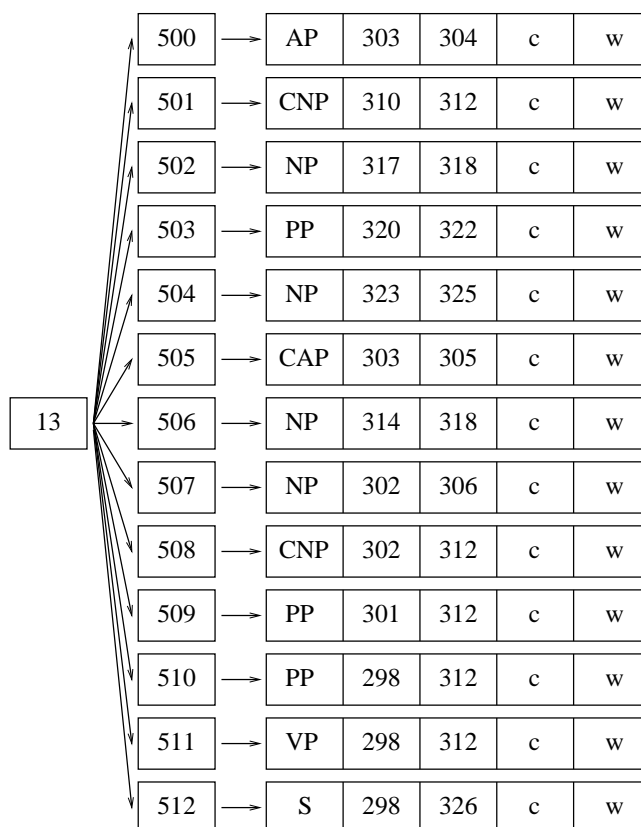
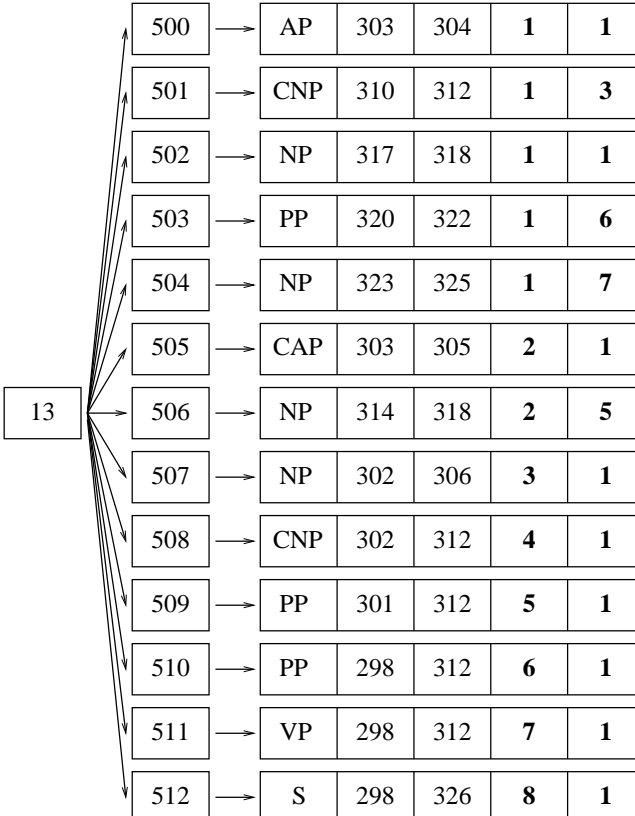


Figure 4.16: Syntax array after completion of step 1

filled in. The starting and ending position are found with the help of a continuous index which numbers each word. If a syntactic constituent occurs for the first time, the index is inserted in the array position of the starting and ending number. Each time the constituent appears again, its former ending position is replaced with the current one, thus finding the end regardless of constituent length. The other entries of the array are so far left open. At the end of step one, the array is sorted so that the syntactic constituents are in numerical order for each sentence. For sentence number 13 from the ComputerZeitung manually annotated corpus the array looks as displayed in figure 4.16. An excerpt from the corresponding MySQL table `Syntax` has been shown in table 4.8 on page 68. Column number and horizontal width are marked with *c* and *w* respectively.

In the second step, a column number is assigned to each constituent. The left-most column is column number one. The first constituent with the lowest constituent number is placed in column one. If the starting and ending positions of the second constituent do not overlap with the first one, it is also placed in column one; otherwise, the column number is increased by one. This procedure is applied to all constituents and the final number of columns stored.

In the third step, the horizontal width of each syntactic constituent is calculated. For each constituent, the column number of the constituent on the left of it is searched. This number is compared to the column number of the constituent. The horizontal width is the difference between these two numbers. The maximal width is stored in the array. After completion of steps two and three, the syntax array displayed in figure 4.16 looks as in figure 4.17. Changes are marked in bold-face print.



500	→	AP	303	304	1	1		
501	→	CNP	310	312	1	3		
502	→	NP	317	318	1	1		
503	→	PP	320	322	1	6		
504	→	NP	323	325	1	7		
505	→	CAP	303	305	2	1		
13	→	506	→	NP	314	318	2	5
	→	507	→	NP	302	306	3	1
	→	508	→	CNP	302	312	4	1
	→	509	→	PP	301	312	5	1
	→	510	→	PP	298	312	6	1
	→	511	→	VP	298	312	7	1
	→	512	→	S	298	326	8	1

Figure 4.17: Syntax array after completion of steps 2 and 3

A special problem pose discontinuous constituents as for example verbal phrase VP in the seventh syntax column in figure 4.15. Since they have more than one starting and ending point, additional arrays starting with different constituent numbers are created. In order to recognize the different parts of a discontinuous constituent, numbers based on its original constituent number increased in steps of 100 are chosen. Constituent number 511 is therefore supplemented with two additional constituents numbered 611 and 711. The column number and horizontal width are taken over from the first part of the constituent. Figure 4.17 is completed with the information displayed in figure 4.18.

The fourth step is concerned with the transformation of this array into HTML-code. A constituent is triggered if its starting position corresponds to the position of the word. Its vertical length and horizontal width are directly inserted into the HTML table field commands *rowspan*

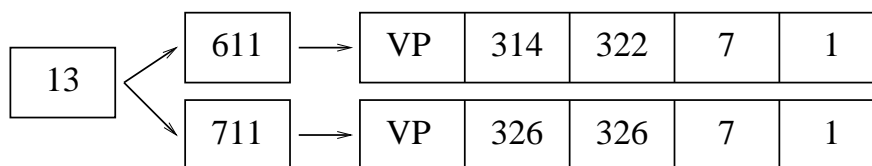


Figure 4.18: Discontinuous elements in syntax array

and *colspan*. For each row, the number of table fields which are not occupied by a constituent has to be calculated so that the rows can be filled. This is achieved by a number of conditions which are checked for all columns. Another function assigns a different background color to each constituent number. In the end, the syntactical output looks as displayed in figure 4.15.

If the corpus is semantically annotated, the semantic constituents are displayed in the last section of the table for complex outputs. The procedure to calculate the positions of semantic constituents is similar to the one described for the syntactical structure, although much less complex. Since there are two types of semantical constituents, namely *named entities* and *prepositional phrase types*, the maximal number of columns is two and the constituent width is always one. The only information needed is therefore the semantic constituent's label and its starting and ending position.

Conclusion In this chapter, I have presented my corpus query tool. All PHP-programs are included in Appendix D on a separate CD-ROM. The next chapter is concerned with an analysis of the system. Since I have presented two different database formats, I will evaluate both and draw a conclusion based on these results.

Chapter 5

Analysis and Evaluation

A running version of the corpus query tool described in the previous chapter is available on a server at the Department of Computer Science at the University of Zurich. It is, however, important to conduct a thorough analysis to estimate the strengths and weaknesses of my corpus query tool. There are several aspects which need to be considered. Obviously, it needs to be checked whether the prerequisites for corpus query established in section 2.2 have been observed. Additionally, the influence of the database format needs to be evaluated by comparing the performance of the two versions of a database format presented in the previous chapter. If one database format yields a better performance at reasonable cost, it will be preferred in the final implementation. Another series of considerations will have to consider the influence of corpus size and estimate the possibilities and limits of my corpus query tool in this respect. The goal of the analysis of all these aspects is to evaluate the status quo of the corpus query system but also to produce suggestions for improvement which may eventually be included at a later stage if the project is continued.

5.1 Interface Evaluation

When comparing the query interface of my corpus query tool with query interfaces of other corpus query tools, the prominent difference is that my corpus query tool provides a number of query types instead of an input field in which allows all sorts of queries. Only a limited set of queries can be posed, and each query needs a particular sequence of input fields which can be translated into SQL query statements. The corpus query systems presented in section 2.2.2 make use of a corpus query language or other formal constructs so that several types of queries can be posed. A possibility to directly formulate SQL query statements to retrieve linguistic data, however, is undesirable for linguists doing research with corpora.

The constrained set of query type is an advantage as well as a disadvantage of my corpus query tool. The advantage is that query types are assessable and therefore rather simple to handle.

Beginners of corpus linguistics are confronted only with a limited set of query types which are easy to understand because the query title already explains what can be retrieved with each query type. On the other hand, an experienced corpus user is restricted to these query types. For an advanced research project in corpus linguistics, new query types will certainly have to be added. It is, however, possible to formulate many types of linguistic queries in SQL and program the respective query interface.

It is not possible at the current stage of the project to give suggestions about the improvement of the interface without confusing the original strategy of the corpus query tool. As a far-fetched suggestion, a query language which could be translated into SQL-statements would certainly bring more flexibility. This hypothetical language would include restricted natural-language elements (e.g. *find all prepositions which are followed by a conjunction*) resembling SQL-query statements. This is, however, subject of a different area of research.

5.2 Output Evaluation

Two of the prerequisites of corpus query established in section 2.2 address result display, namely *result reproduction* and *reconstructability*. *Result reproduction* refers to the necessity of displaying the query results in a meaningful way to the user and offer basic statistical information about the number of hits and the corpus size. *Reconstructability* requires that the location of the result must be clearly discernible. Location refers to the corpus name, sentence number, or other convenient information of the result's origin so that the original context can unambiguously be reconstructed. The fulfillment of these two tasks is accomplished by the output presentation of my corpus query tool.

Result Reproduction As I have presented in section 4.6, there are three different types of result reproduction in my corpus query tool. All of them serve a different purpose: the first output format displays the results in the KWIC format so that a broad overview of concordances can be gained quickly, the second output format lists each sentence with its part-of-speech tags glossed below it so that the sentence's part-of-speech structure can be discerned easily, and the third output format shows the complete annotation information including part-of-speech tags, lemmas, and the syntactical and semantic annotation of the sentence. The third output version suits a query type which is interested in detailed syntactic or semantic structures, its disadvantage being that it is too elaborate for a fast analysis of a large number of sentences. The type of result reproduction therefore depends on the user's interest and the query type.

For queries which ask for simplified part-of-speech tags or single items, my corpus query tool allows a choice between the KWIC output format and the output with part-of-speech tags. For queries comprising two queried items at variable distance or syntactical constituents, the output is presented in the table displaying all annotation information. This distribution of output formats based on the query type seeks to display the results in a meaningful way to the user.

Additionally, each output format is supplemented with an information sentence which tells a user about the query and the number of hits which have been retrieved. It also includes a statement about the frequency of the queried item per one million words.

The above statements about output formats and the information sentence show that the prerequisite of *result reproduction* is fulfilled in my corpus query tool. There are, however, suggestions for improvement. One improvement of my corpus query tool with regards to output reproduction is that a user should be able to choose among all three output formats, allowing the definition of “meaningful” output format to the persons using the corpus query tool themselves. Additionally, a graphical output would complete the choice of output formats. This is, however, beyond the scope of the original project.

Reconstructability Since the corpora are stored in database relations which include an identifying word number as well as a sentence number, *reconstructability* is of no difficulty for my corpus query tool. In the information sentence which appears at the top of each result reproduction page, the corpus from which a sentence was taken is indicated. At the beginning of each sentence, its number is displayed. The origin of the sentence is therefore unambiguously reconstructible.

5.3 Database Performance

The database system is the core of my corpus query tool. Its evaluation is therefore an important matter because the efficiency of the whole tool depends on the format of the data which is stored in the database system. There are several aspects which will be analyzed in this section. In a first analysis, I will test the difference between the two versions of the corpus database format which I have presented in section 4.2. Subsequently, I will try to predict the performance of my corpus query system for the intended operation with large automatically annotated corpora. In a concluding step, I will check whether the database of my corpus query systems fulfills the prerequisites set for a corpus query system in section 2.2 concerning the retrieval of data from the corpus database.

5.3.1 Comparison of Database Formats

In section 4.2 I have presented two different database formats for my corpus query tool. The first version is based on the textbook approach which observes the four steps of database modelling and results in an entity-relationship model which can be transformed into a relational database. The second version is an adaptation and extension of Oliver Plaehn’s database model described in [Plaehn 1998] designed to store corpora in the NeGra export format for the ANNOTATE project. The first version of the database format stores the information from the corpus files in three tables `Sentence`, `Word`, and `Parent`. The second model makes use of an N-table structure which

reduces redundancy by assigning an identifying number to all annotation data and storing only the number in place of a text.¹

Reflecting on these facts, one may assume that the first version of the database format possibly results in shorter data retrieval times because a smaller number of joins between tables has to be executed in order to gather the information required by the output formats. There is, however, a trade-off between shorter retrieval times and storage space. By which factor can the corpus query system be accelerated through the database format, and how does this improvement affect storage space? In order to answer these questions, I have conducted a series of measuring experiments with both database formats.

For the experiments measuring data retrieval times, the time which is needed to browse a corpus database and retrieve the instances matching a query is observed. The timing experiments were set up in the following way: a query was posed from the query interface and resulted - instead of displaying sentences - in outputting the time which was needed to retrieve the requested data. The SQL-query statements were supplied with a time stamp function which measured the time before beginning and after completing the correspondence with the MySQL database. The PHP-function `microtime()` which outputs the current time stamp in the format of seconds and microseconds accomplished this. The period of time which was needed to retrieve a result from the MySQL database was calculated by subtracting the start time from the end time. Additionally, the SQL-query statement was slightly modified. Since we are interested in knowing the retrieval time of all hits, the LIMIT-clause which constrains the number of hits was removed and the query results not passed on to the outputting function.

The time for each query was measured three times in order to prevent experimental inaccuracies or irregularities in the operation of the network system. Since the MySQL corpus database runs on a server at the Department of Computer Science which is also used for other purposes, some deviations in retrieval times could be made out. Additionally, a database management system is able to copy several data blocks from the hard disk into the cache of the main memory (Random Access Memory). If the same query is posed twice in a row, the retrieval time is much shorter for the second query. When conducting the experiments measuring data retrieval times, queries were alternated to handle the buffering of results. The final time was calculated by averaging the three results and is presented in seconds.

Simple Query Tables 5.1, 5.2, and 5.3 show a comparison of retrieval times of simple queries in both database format versions. The query types evaluated are *query for word* and *query for a word directly followed by another word*. For the experiments with the *query for word*, five words were chosen which occur in the corpora with different frequencies. Depending on the corpus, the first word *Mensch* occurs between 32 and 57 times per one million words²; the most frequent word, the full stop (*.*), occurs between 45'831 and 46'619 times per one million words.

¹A complete list of the relations used in both database formats can be found in Appendix B.

²There are several reasons for the large difference in the number of occurrences of the word *Mensch* in the corpora observed: Since the corpora are relatively small, a word occurring seldom has more impact which is not levelled out as in a random distribution of large corpora. Additionally, different text genres (such as the difference between

Table 5.1: Database Format Comparison of the TagesAnzeiger Corpus: Simple Query

Queried word	Number of hits	TagesAnzeiger (database version 1)	TagesAnzeiger (database version 2)	Factor DB2:DB1
Mensch	18	0.14 s	0.37 s	2.64
hat	1091	1.26 s	3.31 s	2.63
von	2653	2.95 s	7.85 s	2.66
die	9022	7.74 s	19.68 s	2.54
.	45878	13.66 s	34.79 s	2.55
er hat	23	0.29 s	0.44 s	1.52
, dass	967	1.61 s	1.83 s	1.14
, die	1253	1.79 s	1.80 s	1.01

Table 5.2: Database Format Comparison of the ComputerZeitung Automatically Annotated Corpus: Simple Query

Queried word	Number of hits	ComputerZeitung autom. annotated (database version 1)	ComputerZeitung autom. annotated (database version 2)	Factor DB2:DB1
Mensch	2	0.08 s	0.30 s	3.75
hat	139	0.21 s	0.72 s	3.43
von	446	0.53 s	1.89 s	3.57
die	1328	1.16 s	3.14 s	2.71
.	2012	2.03 s	5.21 s	2.57
er hat	0	0.07 s	0.31 s	4.43
, dass	0	0.08 s	0.31 s	3.88
, die	190	0.38 s	0.49 s	1.29

Table 5.3: Database Format Comparison of the ComputerZeitung Manually Annotated Corpus: Simple Query

Queried word	Number of hits	ComputerZeitung manually annotated (database version 1)	ComputerZeitung manually annotated (database version 2)	Factor DB2:DB1
Mensch	2	0.10 s	0.30 s	3.00
hat	205	0.36 s	0.96 s	2.67
von	968	1.12 s	2.91 s	2.60
die	2210	1.92 s	5.34 s	2.78
.	2859	3.14 s	8.54 s	2.72
er hat	1	0.09 s	0.34 s	3.78
, dass	0	0.07 s	0.31 s	4.43
, die	334	0.48 s	0.60 s	1.25

The remaining three queries close the gap between these two at more or less regular intervals. The same procedure was chosen for the *query for word followed by word*. An infrequent pair of words (*er hat* which occurs 0 to 73 times per one million words) is contrasted by the most frequent combination of two words, which is a comma followed by the relative pronoun *die*. It is the most frequent because *die* is a relative pronoun following feminine nominative singular nouns as well as all plural nouns. The combination of , *die* occurs between 3990 and 5354 times per one million words.

As can be observed in tables 5.1, 5.2, and 5.3, the first version of the database format always results in shorter data retrieval times for simple queries. The last column shows the factor by which the first version works faster than the second one. The factor varies between 1.01 (which is hardly an improvement at all) and 4.43 (which occurs twice in a query which had no instances to retrieve). The vast majority of simple queries is improved by a factor between 2.5 and 2.7. These results show that the first version of the database format is definitely preferable with regards to data retrieval times.

The factor between the first and the second version of the database format differs between the two types of query compared. A *query for a word* is for the most part improved by a factor between 2.5 and 2.7 with the exception of queries which retrieve a small number of matching instances. If only a few matching instances are found, a query for a word may be improved by a factor between three and four. This differs, however, from the *query for word followed by word*. When neglecting queries which retrieve only a few matching instances, a *query for a word followed by a word* is only improved by a factor between 1.01 and 1.29. As table 5.4 shows, the reason for this divergence can possibly be found in the number of joins which have to be executed.

Table 5.4: Comparison of the Number of JOINS for Simple Queries

	Number of JOINS in database version 1	Number of JOINS in database version 2
Query for word	1	3
Query for word followed by word	2	4

The SQL query statement for a *query for word* requires the join between a temporary table and table `Word` in the first version of the database format and three joins in the second version. The joins which are added occur between tables `Word` and `Tag` and tables `Tag` and `TagSimple`. The number of joins for the *query for word* can thus be reduced by a factor of three. For the *query of a word followed by a word*, two joins are necessary in the first and four joins in the second version of the database format. The joins necessary in both database versions occur between tables `Word` and `Word` (table `Word` is joined with itself to allow the calculation of the distance between two different word identification numbers) and a temporary table and table `Word`. The additional joins in the second version of the database format occur between tables

the `ComputerZeitung` treating technical subjects and the `TagesAnzeiger` as a daily newspaper) make use of different words.

Word and Tag and tables Tag and TagSimple. The number of joins can therefore only be reduced by a factor of two. A larger factor of join reduction is certainly a reason for improved data retrieval times with the first version of the database format.

Table 5.5: Database Format Comparison of the TagesAnzeiger Corpus: Complex Query

Query Type	Queried Item	Number of hits	Tages-Anzeiger (version 1)	Tages-Anzeiger (version 2)	Factor DB2:DB1
Query for pos followed by pos with distance exactly 1	AJDA NN	13264	20.41 s	40.15 s	1.97
Query for pos followed by pos with distance at most 5	AJDA NN	21950	27.84 s	49.32 s	1.77
Query for pos followed by pos with distance at least 1	ADJA NN	31085	27.63 s	46.84 s	1.70
Query for syntactical constituent	AP	2418	4.94 s	9.63 s	1.95
Query for syntactical constituent	NP	27653	26.65 s	51.54 s	1.93
Query for sentence	1	1	0.0045 s	0.0072 s	1.60
Query for sentence	3000	1	0.0028 s	0.0041 s	1.64

Complex Query Tables 5.5, 5.6, and 5.7 show the data retrieval times for complex queries. A complex query is distinguished from a simple query by the output format. A complex query is displayed in a table which contains the verticalized sentence including its syntactic as well as semantic annotation. To make these linguistic information available, the information has to be retrieved from the respective tables. The complex queries chosen for timing experiments are *query for part-of-speech tag followed by part-of-speech tag in variable distance*, *query for syntactical constituent*, and *query for sentence number*. A *query of a word followed by a word* has already been evaluated in the simple query experiments, so the influence of the distance operator between part-of-speech tags (*exactly*, *at most*, *at least*) will be noteworthy. The *query for a syntactical constituent* is insofar important as the queried item is for once not searched in table Word but in table Parent. The query for one particular sentence (also called “corpus-browsing”) intuitively stands out with extremely fast result presentation, so it is interesting to see if the database format affects and improves such a well-running query type as well.

For the evaluation of the *query for a part-of-speech tag followed by another part-of-speech tag*, the combination of part-of-speech tags chosen was always the same, namely an attributive adjective (ADJA) followed by a regular noun (NN). Since both of these part-of-speech tags occur

Table 5.6: Database Format Comparison of the ComputerZeitung Automatically Annotated Corpus: Complex Query

Query Type	Queried Item	Number of hits	CZ autom. annotated (version 1)	CZ autom. annotated (version 2)	Factor DB2:DB1
Query for pos followed by pos with distance exactly 1	AJDA NN	1986	2.98 s	5.56 s	1.87
Query for pos followed by pos with distance at most 5	AJDA NN	3381	3.92 s	6.49 s	1.66
Query for pos followed by pos with distance at least 1	ADJA NN	4848	4.05 s	6.53 s	1.61
Query for syntactical constituent	AP	249	0.57 s	1.09 s	1.91
Query for syntactical constituent	NP	4025	3.76 s	7.28 s	1.94
Query for sentence	1	1	0.0039 s	0.0072 s	1.85
Query for sentence	3000	1	0.0046 s	0.0058 s	1.26

frequently, large numbers of matching instances can be found in a corpus. The ADJA-NN combination was tested with three different distance operators. The first mode requested a distance between ADJA and NN of exactly 1 and resulted – depending on the corpus – in a frequency between 42'235 and 51'072 instances per one million words³; the second mode queried for a distance of at most five and resulted in a frequency between 69'894 and 89'128 instances per one million words, and the last mode queried for a distance of at least one, resulting in frequencies between 98'981 and 130'999 instances per one million words. The number of matching instances is thus increased with each query mode so that the influence of large numbers of results can be observed.

The data retrieval times of the *query for a part-of-speech tag followed by another part-of-speech tag* show that the first version of the database format results in better retrieval times than the second one by a factor between 1.37 and 1.97. Interestingly enough, a difference between

³The large divergence in occurrences of adjectives and nouns between automatically and the manually annotated corpora is due to the fact that the sentences in the manually annotated corpus were selected because they contained at least one full verb and one sequence of a noun followed by a preposition. This selection criterium requires them to be of a certain complexity, whereas sentences in the automatically annotated corpora vary in their size, possibly even being very short. Adjectives therefore compellingly occur more often in the sentences which were selected. Cf. section 4.3 for more information about the corpora.

Table 5.7: Database Format Comparison of the ComputerZeitung Manually Annotated Corpus: Complex Query

Query Type	Queried Item	Number of hits	CZ manu. annotated (version 1)	CZ manu. annotated (version 2)	Factor DB2:DB1
Query for pos followed by pos with distance exactly 1	AJDA NN	3186	4.93 s	7.15 s	1.45
Query for pos followed by pos with distance at most 5	AJDA NN	5560	6.60 s	9.08 s	1.38
Query for pos followed by pos with distance at least 1	ADJA NN	8172	6.56 s	9.00 s	1.37
Query for syntactical constituent	AP	874	1.78 s	2.65 s	1.49
Query for syntactical constituent	NP	7275	6.25 s	9.27 s	1.48
Query for sentence	1	1	0.0051 s	0.0055 s	1.08
Query for sentence	3000	1	0.0051 s	0.0065 s	1.27

the manually annotated and the automatically annotated corpora can be made out. The Tages-Anzeiger and the ComputerZeitung automatically annotated corpus are improved by a factor of always greater than 1.61; with the ComputerZeitung manually annotated corpus it is always less than 1.40. These numbers indicate that the frequency of the queried item plays an important role in the time of retrieval. I will comment further on this idea in section 5.3.2.

Since the automatically annotated corpora were syntactically only annotated with adjective, prepositional and noun phrases, the syntactic constituents queried in all corpora were AP and NP. The data retrieval times reflect the insights gained with the query for part-of-speech tags. Adjective phrases as well as noun phrases occur with a higher frequency in the manually annotated corpus because as mentioned in section 4.3, this corpus is composed of selected sentences which include at least one full verb and a sequence of a noun followed by a preposition, thus being of a certain complexity which include more adjective and noun phrases. Likewise, the improvement by database format version 1 results in a factor around 1.5 for the manually annotated corpus but in a factor around 1.9 for the automatically annotated corpora. The assumption that data retrieval times are not only influenced by the database scheme but also by the frequency of the queried item is confirmed.

The *query for a sentence* always resulted in data retrieval times below 72 milliseconds. It is

hard to judge whether times of this order are representative. The general trend of an improvement by the first version of the database format can still be made out, as can be seen in the difference between the manually and the automatically annotated corpora.

Again, a difference between query types can be discerned. In all corpora, the query for a syntactical constituent can be improved by using the first version of the database format by a larger factor. The following analysis of the number of joins used in both SQL query statements shows how this aspect influences the outcome.

Table 5.8: Comparison of the Number of JOINS for Complex Queries

	Number of JOINS in database version 1	Number of JOINS in database version 2
Query for pos followed by pos in variable distance	3	7
Query for syntactical constituent	2	6

Table 5.8 shows that the number of joins for a *query for a part-of-speech tag followed by another part-of-speech tag* can be improved in the first version of the database format by a factor of two; the number of joins for a *query for a syntactical constituent* can be improved by a factor of three. Based on the previous assumptions about the reduction of joins, these numbers explain why the improvement factors for the *query for syntactical constituent* are slightly higher than the improvement factors for the *query for part-of-speech tag followed by part-of-speech tag*.

I have shown that a selection of query types standing for all query possibilities can be improved by using the first version of the database format. The improvement is caused to a large extent by the reduced number of joins. The negative influence of the number of joins can also be seen in a more detailed analysis of the improvement factors which show that the more joins have been omitted, the faster can data be retrieved from a database. There is, however, a trade-off with storage room. The reduction of joins inevitably results in more redundancy and the corpus database therefore requires more storage space on a server. Table 5.9 shows a comparison of the storage room which each corpus database occupies in the first and the second version of the database format.

Table 5.9: Comparison of Storage Space between Database Format Versions

	Storage Space Version 1 (Megabytes)	Storage Space Version 2 (Megabytes)	Factor DB1:DB2
TagesAnzeiger	60.50	57.15	5.86%
ComputerZeitung autom. annotated	8.28	7.95	4.15%
ComputerZeitung manually annotated	12.41	11.59	7.08%

Table 5.9 shows that a corpus stored in the first version of the database format occupies more storage space than a corpus stored in the second version. The amount of additional disk space

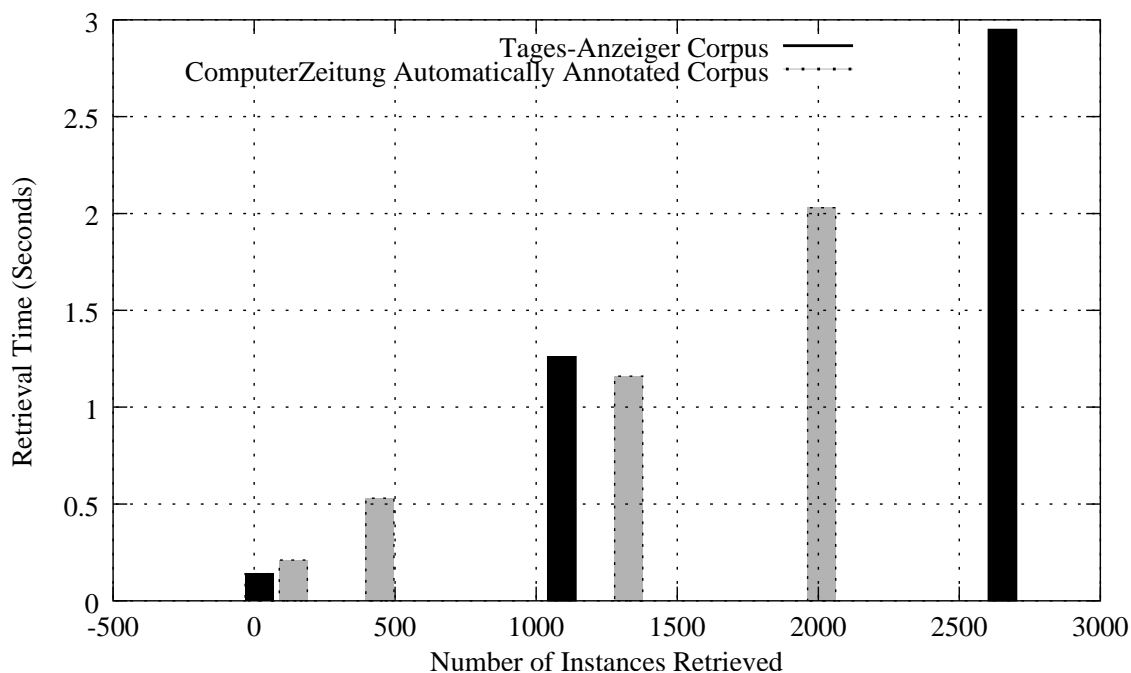


Figure 5.1: Data Retrieval Times per Matching Instances in the ComputerZeitung Manually Annotated Corpus and the Tages-Anzeiger Corpus

varies between four and seven per cent. Compared with the average improvement of 122 per cent for data retrieval times from all queries, the additional storage space is more than compensated.

5.3.2 Database Performance

In a final stage, the corpus query tool is supposed to be used for large automatically annotated corpora. The Tages-Anzeiger corpus contains 314'049 words; a large corpus most likely exceeds one million words. In this section I would like to analyze how the size of a corpus influences retrieval times. Although I have not focused on optimization strategies, a corpus query system which confronts a user with unbearably long loading times is not acceptable. The analysis of the influence of the corpus size on data retrieval times will be based on the results from the timing experiments presented in the previous section.

In a first analysis, the time interval needed to retrieve a certain number of matching instances in two different corpora is compared. The corpora chosen are the ComputerZeitung automatically annotated corpus and the Tages-Anzeiger corpus because their annotation is comparable. The ComputerZeitung corpus contains 3000 sentences; the Tages-Anzeiger contains corpus 20'000 sentences. If a query retrieving the same amount of matching instances takes the same time in

both corpora, the size of the corpus is of no importance. Figure 5.1 shows that this is almost but not exactly the case.

None of the queries retrieve exactly the same amount of matching instances. When focussing on the two bars in the center of figure 5.1 which show the query for word *hat* in the Tages-Anzeiger corpus which retrieves 1091 hits and on the query for word *die* in the ComputerZeitung corpus which retrieves 1328 hits, a general trend can be made out. The corpus query system needs slightly more time (3.31 seconds compared to 3.14 seconds) to retrieve less instances in the larger Tages-Anzeiger corpus. A slight influence of the corpus size can therefore be made out. It is, however, impossible to name an exact number which stands for the difference in data retrieval times depending on corpus size based on the experiments conducted on the three testing corpora.

Although the direct influence of corpus size cannot be determined, its indirect effects on the number of instances retrieved can be examined. When trying to find the exact amount of the influence of the corpus size on data retrieval times, the general trend of retrieval times per number of instances retrieved may give an indication about the development with larger corpora. Figure 5.2 shows the connection between the number of instances which have to be retrieved and the amount of time needed to do so in the Tages-Anzeiger corpus.

Figure 5.2 shows that data retrieval times of queries for a word in the Tages-Anzeiger corpus progress in a linear way depending on the number of hits which have to be retrieved.⁴ An equation approximating this line can therefore be calculated. Equations 5.1 and 5.2 show how to find the best fit straight line from a set of measured values. In this case, the number of values n equals five because five different queries for word have been measured. Variable a corresponds to the y-axis intercept, and variable b matches the slope of the line. Equation 5.1 shows how the approximate value of the arithmetic mean of a random sample with n values is calculated, and equation 5.2 shows the formula of a regression line including the calculation of variables a and b . Both formulas can be found on page 91 in [Verein Schweizerischer Mathematik- und Physiklehrer 1992].

$$(5.1) \quad \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

$$(5.2) \quad y = a + bx, \quad a = \frac{\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}}{\sum_{i=1}^n x_i^2 - n \bar{x}^2}, \quad b = \bar{y} - a \bar{x}$$

If applied to the results of the Tages-Anzeiger corpus, equations 5.1 and 5.2 result in the following equation which approximates the line in figure 5.2.

$$(5.3) \quad y = 0.2042 + 0.0009x$$

⁴The data retrieval times of both ComputerZeitung corpora progress in exactly the same way but are for reasons of legibility not included.

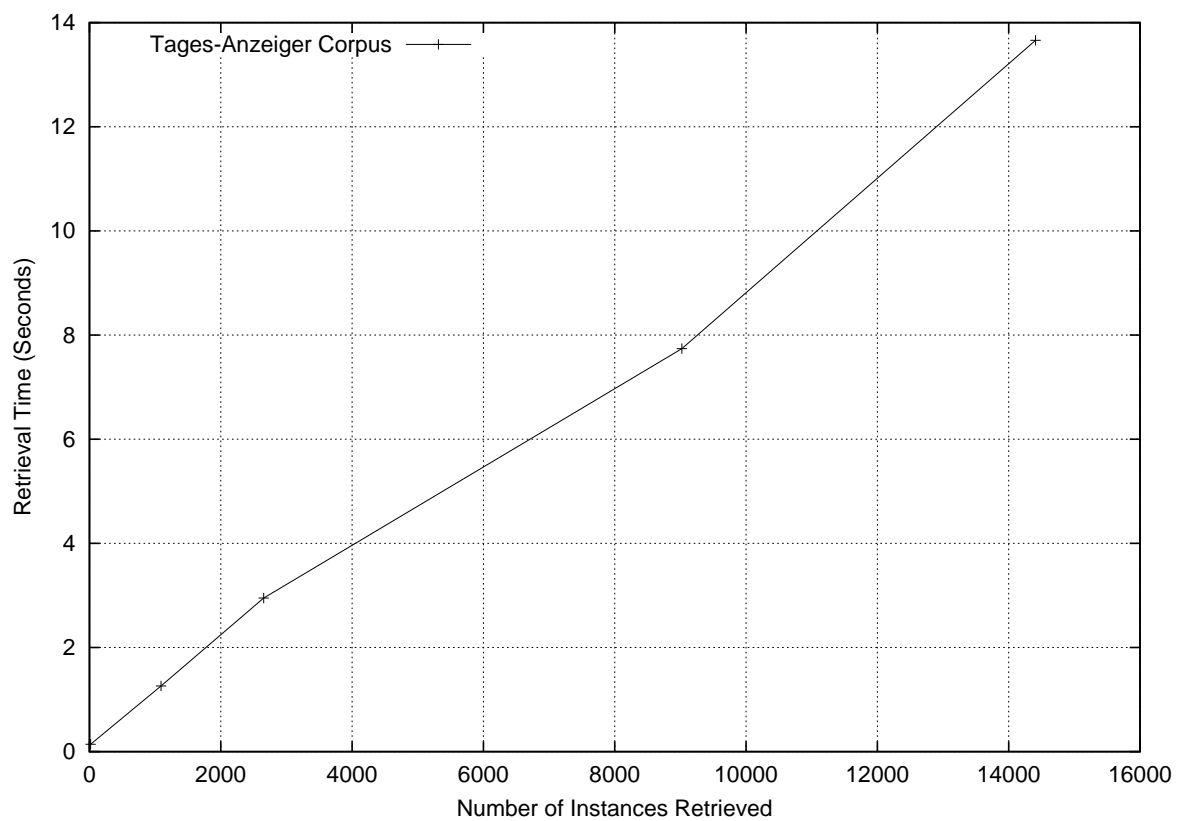


Figure 5.2: Comparison of Data Retrieval Times per Matching Instances in the Tages-Anzeiger Corpus

Equation 5.3 shows that the data retrieval times of the Tages-Anzeiger progress in a linear way with the slope of 0.0009. The line does not exactly intersect the x-axis at zero but crosses it at the x-value of 0.2042. In order to find out how correct such an approximative equation for a line is, a correlation coefficient can be calculated. This coefficient is based on the standard deviation and measures the linear cohesion between independent values.⁵ For equation 5.3, the correlation coefficient r is displayed in equation 5.4.

$$(5.4) \quad r = 0.989674784$$

A perfect correlation between an equation and a line corresponds to the coefficient of one. The correlation coefficient displayed in 5.4 shows that equation 5.3 comes very near to the actual data.⁶ Assuming that the distribution of data retrieval times progress as they have 20'000 sentences, guesses about retrieval times with larger corpora can be made. This is, however, a risky business because the hypothesis of equation 5.3 could only be proven by two corpora of 3000 sentences.

If equation 5.3 is applied, one has to be aware of the fact that the influence of corpus size is neglected. The factor considered is the number of instances which have to be retrieved. Although it is risky to predict a general trend based on the data from only two corpora, it seems that corpus size does have an indirect influence on retrieval times, but it is nowhere as important as the influence of the number of instances which match a query. There is, however, a connection between the two because a linguistic entity will occur with the same frequency regardless of corpus size.

5.3.3 Database Evaluation

Among the four criteria which are compelling prerequisites for any corpus query system, two concern the structure of the corpus database. The first prerequisite is *completeness*, meaning that every occurrence which matches a query is found in a corpus. When working with a database system, this prerequisite is met if the SQL query statement is correct and the database management system works properly. The correctness of the SQL query statement can be tested by working with a small number of sentences. The results from a query statement can be checked manually. If the SQL query statement outputs the correct instances from an assessable amount of sentences, it will also do so on a larger scale. On the other hand, a database management system is definitely reliable because it has been used often and experiences made in these applications guarantee a correct operation. The criteria of *completeness* is therefore fulfilled.

The second prerequisite for corpus query tools being connected with the corpus database is *efficiency*. It states that the time of retrieval must be limited to a reasonable interval. In

⁵The detailed formula can be found in [Verein Schweizerischer Mathematik- und Physiklehrer 1992] on page 85.

⁶Another regression line with the correlation coefficient $r=0.9897$ is possible. Its equation is $y = a * x^b$ with $a=0.0171$ and $b=0.6679$. Although the linear progression matches the data better, this equation shows that the development is less than of linear order because b is less than 1. I will, however, prefer the linear equation.

the previous subsection, I have presented data retrieval times for my corpus query tool. When comparing the two database formats, it is obvious that the second version of the database format does not result in acceptable data retrieval times. Depending on the query, I have been able to reduce the time span between fifty to three hundred percent.

I have also been able to show that retrieval times chiefly depend on the number of hits which have to be retrieved from a corpus. Queries matching a large number of instances will, however, match an even greater number of instances in a larger corpus, and retrieval times will grow proportionally. With 20'000 sentences, a query retrieving 4848 instances (0.015 instances per corpus) takes approximately four seconds. The final time, however, will be somewhat different because only ten results are retrieved at a time, but a HTML page also needs time for loading. In my opinion, this is still a reasonable time interval. The prerequisite of efficiency, however, remains a matter of worry to my corpus query system.

5.3.4 Suggestions for Improvement

Some suggestions for improving my corpus query tool have already been mentioned in the previous sections of this chapter and will not be repeated here. The main concern of my corpus query system, however, is its performance. Although I have implemented the most obvious query optimization strategies such as indexing, reducing the number of joins, and reducing the number of sentences retrieved, there are some further ideas about improving the performance of my corpus query tool.

For general information about performance tuning of MySQL database system, the fifth chapter in [Reese et al. 2002] devoted to this topic gives insights about approaches which can be taken to improve the performance of my corpus query system. On page 81, Reese et al. (2002) list five points which are to be considered.

1. Application tuning
2. SQL query tuning
3. Database server tuning
4. Operating system
5. Hardware

The order of the list corresponds to the effects of the measures. Application tuning therefore yields better results than hardware updating. For further work being done on my corpus query tool, this list will be a helpful support.

The first two issues include host application and SQL query tuning. Most suggestions for SQL query tuning have already been implemented (the strategies for SQL query tuning have

been described in section 4.4). The effect of some other strategies mainly in the section of host application tuning (such as data caching and connection pooling) remain to be tested. So far I have not optimized the PHP-scripts so that an improvement in this area is certainly possible.

Another idea about SQL query tuning was suggested by Dr. Bernhard Ruef during a presentation of my corpus query system within a lecture about corpus linguistics with and for computational linguistics. He suggested that the instances of a query retrieving more than one item should be tested so that the scope of the query can be restricted as soon as possible by using the least frequent item to be queried first. For more complex queries, this strategy would certainly improve retrieval times. I am convinced that this strategy and others in a similar style as well could result in better retrieval times.

The last three issues of the list of MySQL query performance tuning concern matters which were out of my sphere of influence because the database server on which my corpus query tool is running belongs to the Institut für Informatik at the University of Zurich. The database server named “ludwig”, however, is set up in the file structure mode and other applications run on it. A different server could possibly influence the corpus query system positively.

It seems that there is a good chance of improving the performance of my corpus query tool if all components including hardware and host language scripts are reexamined and optimized based on their operational purpose. The improvements remain, however, the task of another programmer.

5.4 Conclusion

The goal of the project which I have carried out in the context of my Lizentiatsarbeit was to implement a corpus query system which stores syntactically annotated corpora in a relational database system. The design of this tool was planned to be web-based so that users can access the database through a HTML query interface. The intended users were beginners in corpus linguistics.

In this thesis I have presented an overview of theoretical issues concerning corpus query and included a description of a selection of other corpus query systems. The focus of the thesis, however, was set on the documentation of my own corpus query system. I have presented its different components, namely the database format, the corpora which were used for testing the corpus query tool, the SQL query statements, the interface functions, and the design of the result output. In a last chapter, I have analyzed my corpus query tool and have been able to show that the format of the database is a crucial factor for the performance of my corpus query tool.

The use of relational database systems to store large amounts of linguistic data (such as syntactically annotated corpora) has so far been neglected. The only project testing the collaboration of database technologies with corpus linguistics is the corpus query system called San Remo by Thomas Knneth ([Knneth98]). Its limitation, however, is that San Remo was restricted to the use with the British National Corpus which is only part-of-speech tagged. Otherwise, current corpus query systems make use of file systems to store and access corpora. With the implementation of my corpus query tool, I have been able to show that the advantages of database systems can be used for the storage of syntactically annotated corpora as well.

As a next step, a comparison between a corpus query system based on a file system and a corpus query system making use of a database system testing query retrieval times of exactly the same corpus would be interesting. Based on the results of the evaluation of my corpus query tool, I estimate at the moment a file system to be faster than my corpus query tool, especially for large corpora. I am convinced, however, that there are a number of improvements - especially involving database technologies which I have not explored yet - which can make a corpus query system based on a database system faster as well as easier to administrate than one based on a file system. The database system, for example, need not necessarily be a relational one. Additionally, the database format as well as retrieval strategies can be optimized for the retrieval of sentences. An intensified effort between the fields of database technology and corpus linguistics would in my opinion result in profound improvements of any type of linguistic data storage.

Another focus of my corpus query tool is the implementation of an intuitive and user-friendly interface which can be accessed through the internet. Without having conducted any experiments with test persons, a conclusive statement about the user-friendliness of the corpus query tool cannot be made. There are, however, a number of components which make the interface easy to handle. All query inputs involving a fixed set of choices (such as part-of-speech tags or syntactical categories) can be selected from a list which includes their name as well as a short description of their function. Tedious consultations of help files and false inputs can thus be

avoided. Additionally, a number of warning messages are displayed if the user inputs senseless query statements. The corpus query tool provides only a selected range of query types, but within these queries, many errors which could occur to an inexperienced corpus linguist have been anticipated.

A test version of my corpus query tool including three corpora described in this thesis is running on a server at the Department of Computer Science at the University of Zurich. Instead of claiming completeness, the current version shows that it is possible to implement a corpus query tool for syntactically annotated corpora based on a relational database system. Further efforts would certainly improve retrieval times and expand its functionality and thus create a corpus query tool which could compete with other ones.

Bibliography

- [Biber et al. 2000] Douglas Biber, Susan Conrad, and Randi Reppen. 2000. *Corpus Linguistics: Investigating Language Structure and Use*. Cambridge University Press, 2nd edition.
- [Brants 1997] Thorsten Brants. 1997. The NeGra Export Format for Annotated Corpora (Version 3). Technical report, Universität des Saarlandes.
- [Burnard 1996] Lou Burnard. 1996. Introducing SARA: An SGML-Aware Retrieval Application for the British National Corpus. *Papers Presented at the Second Conference on Teaching and Language Corpora*.
- [Bußmann 1990] Hadumod Bußmann. 1990. *Lexikon der Sprachwissenschaft*. Alfred Kröner Verlag, Stuttgart, 2nd edition.
- [Carstensen et al. 2001] Kai-Uwe Carstensen, Christian Ebert, Cornelia Endriss, Susanne Jekat, Ralf Klabunde, and Hagen Langer, editors. 2001. *Computerlinguistik und Sprachtechnologie: Eine Einführung*. Spektrum Akademischer Verlag, Heidelberg, Berlin.
- [Corley et al. 2001] Stefan Corley, Martin Corley, Frank Keller, Matthew W. Crocker, and Shari Trewin. 2001. Finding Syntactic Structure in Unparsed Corpora: The GSearch Query System. *Computers and Humanities*, 35(2):81–94.
- [Elmasri and Navathe 2000] Ramez Elmasri and Shamkant B. Navathe. 2000. *Fundamentals of Database Systems*. Addison Wesley Longman, Reading, Massachusetts, 3rd edition.
- [Finegan 1999] Edward Finegan. 1999. *Language: Its Structure and Use*. Harcourt Brace College Publishers, Fort Worth, 3rd edition.
- [Garside et al. 1997] Roger Garside, Geoffrey Leech, and Anthony McEnery. 1997. *Corpus Annotation: Linguistic Annotation from Computer Text Corpora*. Longman, London and New York.
- [Jurafsky and Martin 2000] Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing*. Prentice Hall, Upper Saddle River.
- [Künneht 1998] Thomas Künneht. 1998. Datenbankgestützte Speicherung von Korpora. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg.

- [Lezius and König 2000] Wolfgang Lezius and Esther König. 2000. Towards a Search Engine for Syntactically Annotated Corpora. In Schukat-Talamazzini, Ernst G. and Zühlke, Werner, editors, *Proceedings der 4. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS-2000)*, pages 113–116, Ilmenau. VDE-Verlag.
- [Linke et al. 1996] Angelika Linke, Markus Nussbaumer, and Paul R. Portmann. 1996. *Studienbuch Linguistik*. Max Niemeyer Verlag, Tübingen, 3rd edition.
- [McEnery and Wilson 1996] Tony McEnery and Andrew Wilson. 1996. *Corpus Linguistics*. Edinburgh Textbooks in Empirical Linguistics, Edinburgh.
- [Meier 2002] Charles F. Meier. 2002. *English Corpus Linguistics: An Introduction*. Cambridge University Press, Cambridge.
- [Mengel and Lezius 2000] Andreas Mengel and Wolfgang Lezius. 2000. An XML-based Representation Format for Syntactically Annotated Corpora. In *Proceedings of the Second International Conference on Language Resources and Engineering (LREC 2000)*, volume 1, pages 121–126.
- [Plaehn 1998] Oliver Plaehn. 1998. ANNOTATE Datenbank-Dokumentation. Technical report, Universität des Saarlandes.
- [Rechenberg and Pomberger 1999] Peter Rechenberg and Gustav Pomberger, editors. 1999. *Informatik-Handbuch*. Carl Hanser Verlag, München, Wien, 2nd edition.
- [Reese et al. 2002] George Reese, Randy Jay Yarger, and Tim King. 2002. *Managing and Using MySQL*. O'Reilly, 2nd edition.
- [Schiller et al. 1999] Anne Schiller, Christine Stöckert, Simone Teufel, and Christine Thielen. 1999. Guidelines für das Tagging Deutscher Textkorpora mit STTS. Technical report, Institut für Maschinelle Sprachverarbeitung der Universität Stuttgart und Seminar für Sprachwissenschaft der Universität Tübingen.
- [Schmid and Kempe 1996] Helmut Schmid and A. Kempe. 1996. Tagging von Korpora mit HMM, Entscheidungsbäumen und Neuronalen Netzen. *Wiederverwendbare Methoden und Ressourcen zur linguistischen Erschliessung des Deutschen*, pages 231–244.
- [Stoll and Leierer 2000] Rolf D. Stoll and Gudrun Anna Leierer. 2000. *PHP4 and MySQL*. Data Becker, Paderborn, 2nd edition.
- [Verein Schweizerischer Mathematik- und Physiklehrer 1992] Verein Schweizerischer Mathematik- und Physiklehrer. 1992. *Formeln und Tafeln: Mathematik - Physik*. Orell Füessli, 5th edition.
- [Volk and Schneider 1998] Martin Volk and Gerold Schneider. 1998. Comparing a Statistical and a Rule-Based Tagger for German. In *Proceedings der 2. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS-1998)*, Bonn.

- [Volk 2001] Martin Volk. 2001. *The Automatic Resolution of Prepositional Phrase Attachment Ambiguities in German*. Universität Zürich, Habilitationsschrift.
- [Voormann and Lezius 2002] Holger Voormann and Wolfgang Lezius. 2002. TIGERin – Grafische Eingabe von Benutzeranfragen für ein Baumbank-Anfragewerkzeug. In Stephan Busemann, editor, *Proceedings der 6. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS-2002)*, pages 231–234, Saarbrücken.
- [Zierl 1998] Marco Zierl. 1998. Entwicklung und Implementierung eines Datenbankssystems zur Speicherung und Verarbeitung von Textkorpora. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg.

Glossary

Annotation (i) the practice of adding explicit additional information to machine-readable text; (ii) the physical representation of such information. ([McEnery and Wilson 1996]:177)

Competence Noam Chomsky distinguishes between a speaker's competence and PERFORMANCE. Competence is the ability to produce and assign meaning to any kind of grammatical utterance. ([Bußmann 1990]:396)

Corpus (i)(loosely) any body of text; (ii)(most commonly) a body of machine-readable text; (iii)(more strictly) a finite collection of machine-readable text, sampled to be maximally representative of a language or variety. ([McEnery and Wilson 1996]:177)

Corpus Linguistics The activities involved in compiling and using a CORPUS to investigate natural language use. ([Finegan 1999]:588)

Database/Database System The combination of a collection of logically coherent data with some inherent meaning and a DATABASE MANAGEMENT SYSTEM. ([Rechenberg and Pomberger 1999]:876)

Database Management System (DBMS) A collection of programs that enables users to create and maintain a database; i.e. a general-purpose software systems that facilitates the process of defining, constructing, and manipulating databases for various applications. ([Elmasri and Navathe 2000]:5)

Descriptive Adequacy Evaluation criterion for grammatical theories of natural language established by Noam Chomsky. A grammar is of descriptive adequacy if it describes a speaker's COMPETENCE to determine the correctness of linguistic expressions. ([Bußmann 1990]:46)

Entity-Relationship Model A data model used for the conceptual design of database applications. The ER-model describes data as entities, relationships, and attributes. ([Elmasri and Navathe 2000]:41-72)

Explanatory Adequacy Evaluation criterion for grammatical theories of natural language established by Noam Chomsky. A linguistic theory is of explanatory adequacy if it does not only determine the correctness of linguistic expressions but additionally explains how a speaker acquires this knowledge. ([Linke et al. 1996]:104-105 and [Bußmann 1990]:46)

Join Operation in relational database systems combining related tuples from two relations into single tuples, resulting in one combined relation.
([Elmasri and Navathe 2000]:219)

KWIC Acronym for KeyWord-In-Context, describing a type of sentence representation in which a keyword is centralized and context preceding or following the keyword is displayed on its left and right respectively.

Lemma The headword form of an entry in a dictionary (German: Grundform), e.g. the inflected verb form *ran* belongs to the lemma *run*.

Parsing (Automatic) grammatical analysis with the goal of finding syntactic functions and constituents of a text. An automatic parser has an accuracy rate of 70-80 per cent.

Performance Noam Chomsky distinguishes between a speaker's COMPETENCE and performance. Performance is the actual output of a speaker's competence, i.e. the utterances produced. ([Bußmann 1990]:396)

Register A language variety associated with a particular situation of use. Examples: baby talk, scholarly writing. ([Finegan 1999]:594)

Relational Data Model Data model implemented in relational database systems representing the database as a collection of relations. A relation is a mathematical set of values resembling a table. ([Elmasri and Navathe 2000]:196-198)

SQL (Structured Query Language) Standardized data definition and data manipulation language for relational database systems.

Tagging (also part-of-speech tagging) (Automatic) grammatical analysis with the goal to find part-of-speech tags. Taggers have become highly accurate; a tagger can automatically analyze a corpus at accuracy rates exceeding 95 per cent. ([Meier 2002]:91)

Tag-Set A group of symbols representing various parts-of-speech. ([Meier 2002]:86)

Tokenization The segmentation of a text into entities most closely resembling words.

Trebank A special kind of corpus consisting of a collection of syntactically annotated sentences. The encoding of the sentence structure resembles a tree.
([Carstensen et al. 2001]:377-385)

Appendix A

SGML-Tags in the ComputerZeitung and Tages-Anzeiger Corpora

Table A.1 shows which SGML-Tags were inserted into the ComputerZeitung corpus in order to preserve the formatting.

Table A.1: SGML-Tags in the ComputerZeitung Corpus

<AUTHOR>	author
<AU_ABBR>	author abbreviation
<CITY>	document anchor city
<DOC>	document begin
<H2>	header
	list item
<LOC>	location of the document within the newspaper (volume and page number)
<NUM>	volume number of the newspaper
<P>	paragraph

Additionally, a sequence of dots ('...') was replaced by the tag <Dots> to prevent a mix-up with dots at the end of a sentence.

The SGML-Tags in table A.2 were already extant in the Tages-Anzeiger corpus.

Table A.2: SGML-Tags in the TagesAnzeiger Corpus

<AN>	article number
<AU>	author
<DA>	date of article writing
<DD>	document
<HT>	main title
<KA>	??
<LA>	language of the article
<LD>	lead
<LG>	caption
<NT>	??
<P>	paragraph
<PG>	page
<RU>	rubric
<SE>	??
<SM>	??
<SO>	source
<TI>	title
<TX>	article text
<UR>	??
<UT>	subtitle
<ZT>	intermediate title

During the annotation, tags <DA> and <PG> were removed. The date of the article can be reconstructed from the article number <AN>.

Additionally, a sequence of dots ('...') was replaced by the tag <Dots> to prevent a mix-up with dots at the end of a sentence.

Appendix B

List of Database Relations

The following tables describe the detailed format of the relations used in the database system implemented for my own corpus query tool. Column **Name** lists the set of attributes (columns) which are defined for a relation, column **Type** indicates the MySQL data type which is used in this column¹, and column **Description** gives additional information about the purpose of the attribute. Column **P** (Primary) indicates if an attribute functions as primary key, column **I** (Index) is checked if the attribute is indexed, and column **U** (Unique) is checked if each attribute value occurs only once. If a single field is marked as primary key, MySQL takes it automatically as indexed and unique.² Each corpus is saved in a new database.

¹The following MySQL data types were used: char (character; the size is indicated in bytes), varchar (variable-length character; the size is indicated in bytes; storage room is the length of value + 1), int (integer; a basic whole number with a range of -2¹⁴⁷483⁶⁴⁸ to 2¹⁴⁷483⁶⁴⁷), smallint (a basic whole number with a range of -32⁷⁶⁸ to 32⁷⁶⁷), and tinyint (a basic whole number with a range of -128 to 127).

²This is only true for single primary keys. Note the behavior of a compound primary key in table `Parent`.

B.1 Database Format Version 1

The first version of the corpus database format is based on an entity-relationship model. Besides a table called *Corpus* which contains meta-information about the corpus which is saved in the database, the first version comprises four tables named *Sentence*, *Word*, *Parent*, and *Syntax*. Table *Word* is the pivotal table which stores linguistic information to each word. Table *Parent* and *Syntax* do so accordingly for syntactic structures.

Table Corpus This table contains the information about the corpus which is stored in the database.

Table B.1: Table Corpus

Name	Type	Description	P	I	U
Name	varchar(100)	name of the corpus	√		
DBName	varchar(50)	database in which the corpus is stored			
MaxSentNr	int	number of the last sentence			
Language	varchar(50)	language of the corpus			

Table Sentence This table stores the number of each sentence. The last number is equivalent to the *MaxSentNr* in table *Corpus*.

Table B.2: Table Sentence

Name	Type	Description	P	I	U
Id	int	identification number of the sentence	√		

Table word This table contains the information to a word. Each word is provided with an identification number. By means of the foreign key `SentenceId`, a word can be identified with one particular sentence. All other attributes of table `Word` are parts of the linguistic annotation.

Since a corpus may contain a large number of words and sentences, datatype `int` is assigned to the word and sentence identification numbers. Attribute `Txt` is text of variable length depending on the word. The corresponding datatype is `int`. The lengthiest part-of-speech tag in the current annotation is seven letters long, so datatype `char(7)` is sufficient. Simplified part-of-speech tags take up room up to fifteen letters, so datatype `char(15)` is assigned. `Edglabels` consists of four letters. Semantical name tags and semantical type tags are of datatype `char(14)` because the opening and the closing brackets have to be converted into their HTML-codes which take up 4 letters each. It is a tendency of lemmas to take up more room than the words, so datatype `varchar(120)` is assigned. Morphological tags were not available in the tested corpora but space of length `char(30)` is provided.

Table B.3: Table `Word`

Name	Type	Description	P	I	U
<code>Id</code>	<code>int</code>	key of the word	√		
<code>SentenceId</code>	<code>int</code>	sentence key		√	
<code>Txt</code>	<code>varchar(100)</code>	word text (token)		√	
<code>Tag</code>	<code>char(7)</code>	part-of-speech tag		√	
<code>TagSimple</code>	<code>char(15)</code>	simplified part-of-speech tag		√	
<code>ParentId</code>	<code>smallint</code>	foreign key to <code>Parent</code> ; the parent is a non-terminal node one of whose children is the word, or 0 if the word is not bound in a syntactic structure.		√	
<code>Edglabel</code>	<code>char(4)</code>	edglabel (description of the edge from the word to the node)		√	
<code>SemName</code>	<code>char(14)</code>	the semantical name tag labels a named entity		√	
<code>SemType</code>	<code>char(14)</code>	the semantical type tag labels temporal or local prepositional phrases		√	
<code>Lemma</code>	<code>varchar(120)</code>	the text of the lemma as analyzed by GERTWOL		√	
<code>MorphTag</code>	<code>char(30)</code>	morphological information of the word		√	

Table Parent This table contains the information about the syntactic annotation of a word. In order to place each parent node with its corresponding sentence, the key to `SentenceId` is included. In syntactic annotations, each parent node may also be a child node of another parent; this is realized by referring recursively from `parentId` to the same table. Parent identification numbers range from 500 to 1000, so data type `smallint` is of sufficient size to number the parent nodes. Since parent identification numbers start at 500 within each sentence and therefore are not unambiguous, the primary key of table `Parent` is composed of the parent and the sentence identification number.

Table B.4: Table Parent

Name	Type	Description	P	I	U
Id	smallint	number of the nonterminal node parent	√		
SentenceId	int	foreign key to <code>Sentence</code>	√		
Edgelabel	char(4)	edgelabel (description of the edge from one node to another)		√	
Parentlabel	char(4)	description of the node		√	
ParentId	smallint	foreign key to <code>Parent</code> ; the parent is a non-terminal node one of whose children is the node, or 0 if the node is not bound in another syntactic structure.		√	

Table Syntax This table is the result of several joins of table `parent` and table `parent-label`. It makes the recursive structure of the syntactic annotation explicit.

Table B.5: Table Syntax

Name	Type	Description	P	I	U
Id	int	foreign key to table <code>Word</code>	√		
SentenceId	int	foreign key to table <code>Sentence</code>		√	
NodeTxt1	tinyint	label of the syntactic node on the first level		√	
NodeTxt2	tinyint	label of the syntactic node on the second level		√	
NodeTxt3	tinyint	label of the syntactic node on the third level		√	
NodeTxt4	tinyint	label of the syntactic node on the fourth level		√	
ParentId	smallint	identification number of the parent node on the fifth level		√	

The number of database tables in the second version of the database format is larger than the one presented above. Some tables of the second database format are only slightly modified; others are added.

B.2 Database Format Version 2

Table Corpus This table contains the information about the corpus which is stored in the database.

Table B.6: Table Corpus

Name	Type	Description	P	I	U
Name	varchar(100)	name of the corpus	√		
DBName	varchar(50)	database in which the corpus is stored			
MaxSentNr	int	number of the last sentence			
Language	varchar(50)	language of the corpus			

Table Sentence This table stores the number of each sentence. The last number is equivalent to the MaxSentNr in table Corpus.

Table B.7: Table Sentence

Name	Type	Description	P	I	U
Id	int	identification number of the sentence	√		

Table Text This table contains the texts of each word occurring in the corpus; in other words, each word type (as opposed to token) is stored. Each text is provided with an identification number of data type integer.

Table B.8: Table Text

Name	Type	Description	P	I	U
Id	int	key of the text-string (type)	√		
Text	varchar(100)	word text		√	

Table word This table contains the information to a word. Each word is provided with an identification number. By means of the foreign key `SentenceId`, a word can be identified with one particular sentence. All other attributes of table `Word` are foreign keys which refer to other tables which contain information about the linguistic annotation of the word.

Table B.9: Table `Word`

Name	Type	Description	P	I	U
<code>Id</code>	int	key of the word	√		
<code>SentenceId</code>	int	sentence key		√	
<code>TextId</code>	int	foreign key to <code>Text</code>		√	
<code>TagId</code>	tinyint	foreign key to <code>Tag</code> ; if the value is 0, no tag is assigned to the word		√	
<code>ParentId</code>	smallint	foreign key to <code>Parent</code> ; the parent is a non-terminal node one of whose children is the word, or 0 if the word is not bound in a syntactic structure.		√	
<code>EdgelabelId</code>	tinyint	foreign key to <code>EdgeLabel</code> ; the edge label is the description of the edge from the word to the node or 0 if the word is not bound in a syntactic structure		√	
<code>SemNameId</code>	tinyint	foreign key to <code>SemName</code> ; the semantical name tag labels a named entity, or its value is 0 if the word does not belong to a semantical name class.		√	
<code>SemTypeId</code>	tinyint	foreign key to <code>SemType</code> ; the semantical type tag labels temporal or local prepositional phrases, or its value is 0 if the word does not belong to a semantical type class.		√	
<code>LemmaId</code>	int	foreign key to <code>Lemma</code> ; if the value is 0, no lemma is analyzed for the word.		√	
<code>MorphTagId</code>	smallint	foreign key to <code>Morph</code> ; if the number is 0, no morphological information is assigned to the word		√	

Table Tag The information to the part-of-speech annotation is stored in this table. In the current annotation, there are 60 part-of-speech tags, so the data type of sufficient size to number the tags is tinyint. Part-of-speech tags are currently no longer than seven letters.

Table B.10: Table Tag

Name	Type	Description	P	I	U
Id	tinyint	identification number of the part-of-speech tag	√		
Txt	char(7)	short text which will be displayed in the graphical representation		√	√
Description	varchar(80)	description of the part-of-speech tag			
TagSimpleId	tinyint	foreign key to TagSimple; the simple tag groups the detailed categories of the STTS into groups which are more intuitive for beginners in corpus linguistics		√	

Table TagSimple Each part-of-speech tag from the Stuttgart-Thübingen tagset is mapped to a broader category in order to have a simplified tagset for beginners in corpus linguistics. A detailed list of the simplified tagset can be found in Appendix C. There are currently eleven simplified part-of-speech tags, so the data type of sufficient size to number the tags is tinyint. The texts of the tags are no longer than fifteen letters.

Table B.11: Table TagSimple

Name	Type	Description	P	I	U
Id	tinyint	identification number of the part-of-speech category	√		
Txt	char(15)	short text which will be displayed in the graphical representation		√	√
Description	varchar(80)	description of the part-of-speech category			

Table Parent This table contains the information about the syntactic annotation of a word. In order to place each parent node with its corresponding sentence, the key to `SentenceId` is included. In syntactic annotations, each parent node may also be a child node of another parent; this is realized by referring recursively from `parentId` to the same table. Parent identification numbers range from 500 to 1000, so data type `smallint` is of sufficient size to number the parent nodes. Since parent identification numbers start at 500 within each sentence and therefore are not unambiguous, the primary key of table `Parent` is composed of the parent and the sentence identification number.

Table B.12: Table `Parent`

Name	Type	Description	P	I	U
<code>Id</code>	<code>smallint</code>	number of the nonterminal node parent	✓		
<code>SentenceId</code>	<code>int</code>	foreign key to <code>Sentence</code>	✓		
<code>EdgelabelId</code>	<code>tinyint</code>	foreign key to <code>Edgelabel</code> ; the edge label is the description of the edge from one node to another or 0 if the word or node is not bound in a syntactic structure		✓	
<code>ParentlabelId</code>	<code>tinyint</code>	foreign key to the description of the node in table <code>Parentlabel</code> ; if the number is 0, no description is assigned to the node		✓	
<code>ParentId</code>	<code>smallint</code>	foreign key to <code>Parent</code> ; the parent is a non-terminal node one of whose children is the node, or 0 if the node is not bound in another syntactic structure.		✓	

Table Edgelabel The labels for the edges in the syntactic annotation are stored here. Since there are forty-four tags for edgelabels in the current annotation, data type `tinyint` is of sufficient size to number the tags. The texts are no longer than four letters.

Table B.13: Table `Edgelabel`

Name	Type	Description	P	I	U
<code>Id</code>	<code>tinyint</code>	identification number of the edgelabel	✓		
<code>Txt</code>	<code>char(4)</code>	short text which will be displayed in the graphical representation		✓	✓
<code>Description</code>	<code>varchar(80)</code>	description of the edgelabel			

Table Parentlabel The labels for parent nodes in the syntactic annotation are stored here. Since there are twenty-six tag for parentlabels in the current annotation, data type tinyint is of sufficient size to number the tags. The texts are not longer than four letters.

Table B.14: Table Parentlabel

Name	Type	Description	P	I	U
Id	tinyint	identification number of the parentlabels	√		
NodeTxt	char(4)	short text which will be displayed in the graphical representation		√	√
Description	varchar(80)	description of the parent label			

Table SemName This table contains the tags used for the semantical annotation of named entities. The categories used are person name (<PERS>), geographical name (<GEO>), company name (<FA>), and product name (<PROD>). Since all of them are numbered to indicate components of larger named entity expressions, the number of semantical name tags totals to forty-one. Data type tinyint is therefore sufficient to describe these values. Since the pointed brackets have to be transformed into their HTML-code which takes up four letters, the texts are of a maximum length of fourteen letters.

Table B.15: Table SemName

Name	Type	Description	P	I	U
Id	tinyint	identification number of the semantic name tag	√		
Txt	char(14)	short text which will be displayed in the graphical representation		√	√
Description	varchar(80)	description of the semantic name tag SemType			

Table SemType This table contains the tags used for the semantical annotation of prepositional phrases. The categories used are temporal prepositional phrase (<temp>), and local prepositional phrase (<loc>). Since all of them are numbered to indicate components of larger named entity expressions, the number of semantical type tags totals to twenty-one. Data type tinyint is therefore sufficient to describe these values. Since the pointed brackets have to be transformed into their HTML-code which takes up four letters, the texts are of a maximum length of fourteen letters.

Table B.16: Table SemType

Name	Type	Description	P	I	U
Id	tinyint	identification number of the semantic type tag	√		
Txt	char(14)	short text which will be displayed in the graphical representation		√	√
Description	varchar(80)	description of the semantic type tag SemType			

Table MorphTag The morphological tags are stored in this table.

Table B.17: Table MorphTag

Name	Type	Description	P	I	U
Id	smallint	identification number of the morphological tag	√		
Txt	char(30)	short text which will be displayed in the graphical representation		√	√
Description	varchar(80)	description of the morphological information			

Table Lemma The lemmas are stored in this table. Each lemma is only stored once and referred to by its primary key. Since lemmas tend to be longer than words, space for 120 letters is provided.

Table B.18: Table Lemma

Name	Type	Description	P	I	U
Id	int	identification number of the morphological tag	√		
Txt	varchar(120)	text of the lemma		√	√

Table Syntax This table is the result of several joins of table `parent` and table `parent-label`. It makes the recursive structure of the syntactic annotation explicit.

Table B.19: Table Syntax

Name	Type	Description	P	I	U
Id	int	foreign key to table <code>Word</code>	√		
SentenceId	int	foreign key to table <code>Sentence</code>		√	
NodeTxt1	tinyint	label of the syntactic node on the first level		√	
NodeTxt2	tinyint	label of the syntactic node on the second level		√	
NodeTxt3	tinyint	label of the syntactic node on the third level		√	
NodeTxt4	tinyint	label of the syntactic node on the fourth level		√	
ParentId	smallint	identification number of the parent node on the fifth level		√	

Final Implementation for Size Comparisons In tables `Tag`, `TagSimple`, `Edgelabel`, `Parentlabel`, `SemName`, `SemType`, and `MorphTag`, a description about the linguistic tags used is included. In the first version of the corpus database, this description is missing. When comparing the size of the two databases, the afore-mentioned tables are therefore also included in the first version of the database format so that both versions contain the same information.

Appendix C

Simplified STTS Tagset

Table C.1 shows a simplified tagset based on the Stuttgart-Tübingen tagset. The STTS-tags in the second column were mapped onto the broader categories displayed in the first column.

Table C.1: Simplified STTS Tagset

Simplified Tag	STTS-Tags
Nomen	NN, NE
Verb	VVFIN, VVIMP, VVINF, VVIZU, VVPP, VAFIN, VAIMP, VAINF, VAPP, VMFIN, VMINF, VMPP
Adjektiv	ADJA, ADJD
Adverb	ADV
Präposition	APPR, APPRART, APPO, APZR
Konjunktion	KOUI, KOUS, KON, KOKOM
Pronomen	PDS, PDAT, PIS, PIAT, PIDAT, PPER, PPOSS, PPOSAT, PRELS, PRELAT, PRF, PWS, PWAT, PWAV, PAV
Artikel	ART
Partikel	PTKZU PTKNEG, PTKVZ, PTKANT, PTKA
Zahl	CARD, ORD
Interpunktion	\$. \$. \$(

Appendix D

PHP-programs on a Separate CD-ROM

Curriculum Vitae

Personal Information

Name Charlotte Merz
Date of Birth April 8, 1977

Education

1997-2003 University of Zurich
English Linguistics and Literature (major)
Computational Linguistics (first subsidiary subject)
Computer Science (second subsidiary subject)

2001 Exchange semester at University of Georgia

1994-1995 High school year at Coffee High, Douglas, GA
Coffee High School Honor Graduate

1993-1997 Kantonschule Zofingen
Matura Typus B

1989-1993 Bezirksschule Zofingen

1984-1989 Primarschule Zofingen

Work Experience

2002-2003 student assistant at the Institute of Computational Linguistics

2000-2001 student assistant for Dr. M. Volk
at the Institute of Computational Linguistics

summer 2000 annotation of linguistic corpora
at the Institute of Computational Linguistics

1997-2000 part-time secretary for Mr. C.-L. Raaflaub
at Raaflaub Attorneys at Law, Zurich