DEREKO

(DEutsches REferenzKOrpus)
German Reference Corpus

Final Report (Part I)

Stefanie Dipper, Hannah Kermes,
Dr. Esther König-Baumer, Wolfgang Lezius (IMS)
Frank H. Müller, Tylman Ule (SfS)

Project leaders: Prof. Dr. Erhard Hinrichs (SfS), Prof. Dr. Christian Rohrer (IMS)
Duration: May 1, 1999 – January 31, 2002

IMS: Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart
SfS: Seminar für Sprachwissenschaft, Universität Tübingen

February 25, 2002

Contents

Chapter 1

Introduction

1.1 Overview

The DEREKO project (Deutsches Referenzkorpus) has been a joint project of the Institut für deutsche Sprache (IDS) in Mannheim, the Institut für Maschinelle Sprachverarbeitung (IMS) in Stuttgart, and the Seminar für Sprachwissenschaft (SfS) in Tübingen. The project has been funded from May 1999 through January 2002 (IDS: March 2002) by the Ministry of Science, Research and the Arts of the State of Baden-Württemberg.

The project was set up in order to improve the infrastructure for text-based linguistic research and development by building a huge, automatically annotated German text corpus and the corresponding tools for corpus annotation and exploitation. This raised the following issues:

1. corpus acquisition
2. corpus preparation
3. corpus exploitation

The task of *corpus acquisition* consisted of marketing activities and contract negotiations in order to convince publishing houses and individuals to grant research licenses for their texts. (Responsibility: IDS)

Corpus preparation involved several steps of 'text enrichment'. The meta-information (author, date of publication, etc.) of a text has to be encoded in normalised markup. The text has to be segmented (i.e. the surface structure of the text has to be detected and marked up, including paragraphs, sentences, and word forms). Furthermore, in order to make the texts more valuable for researchers interested in a wide range of linguistic phenomena, a partial syntactic analysis has been carried out, in addition to POS tagging and lemmatisation, and all additional information was added via a customised markup scheme. (Responsibilities: IDS for the markup of meta-information and sentence and paragraph segmentation; SfS for linguistic annotation, with some input on the lexical level from the IMS)

In order to make use of the linguistically annotated text corpus, powerful specialised tools for

corpus exploitation are needed. The basic tool developed for this purpose is a query engine ('TIGERSearch'), which can access structural text annotation in an efficient manner. On this basis, query collections were built which help answer the questions which lexicographers and linguists may have, for example in which lexical and syntactic contexts the word "streichen" (paint; erase) occurs and how often it occurs in these contexts. (Responsibility: IMS)

The tools and resources developed for the DEREKO project provide an important repository for the Kompetenzzentrum für Text- und Informationstechnologie, where they will be developed further and made available to the language technology community and to academic researchers in the field of linguistics and computational linguistics.

This document and additional documentation for DEREKO can be found at the project web site:

`http://www.sfs.uni-tuebingen.de/dereko/`

1.2 Cooperations

The work for the DEREKO project has been carried out in close interaction with the following projects:

- TIGER (funded by DFG)

  Partners: Computational Linguistics, Universität des Saarlandes; IMS; Institut für Germanistik, Universität Potsdam

- Kompetenzzentrum für Text- und Informationstechnologie (funded by MWK Baden-Württemberg)

  Partners: IMS and SfS

- SFB 441: Linguistische Datenstrukturen (funded by DFG)

  Partner: SfS

- TMR – Learning Computational Grammars (funded by EC)

  Partners: University of Groningen, The Netherlands; University of Antwerp, Belgium; SRI Cambridge, United Kingdom; University College Dublin, Ireland; University of Geneva, Switzerland; SfS; XRCE Grenoble, France

The TIGER project shared the development of the specialised query engine TIGERSearch. The DEREKO and TIGER corpora have the 'STTS' tagset [26] in common for the part-of-speech annotation. A certain amount of exchange took place with respect to issues of syntactic annotation.

The Kompetenzzentrum took the role of the prototypical user. Tasks from the lexicon development at the IMS itself ('IMSLex') and from joint projects with dictionary publishers gave rise to the construction of specific query collections.

Both SFB 441 and TMR-LCG developed annotation standards and tools jointly with the DEREKO project and shared the resulting resources.

Chapter 2

Corpus Acquisition and Document Markup

Due to delays caused by negotiations with publishers, the full DEREKO corpus will be available only when the project ends for the project partner IDS Mannheim on March 31, 2002. The other two project partners, IMS Stuttgart and SfS Tübingen, have agreed on using the taz newspaper corpus (September 1986 to May 1999), which both partners have licensed, as an interim solution. The taz corpus contains more than 200 million words, which is approximately 20% the size of the final DEREKO corpus. The taz corpus can therefore be considered to be a reasonable approximation of the final corpus for testing the annotation and query tools.

The IDS Mannheim has currently prepared a sample corpus of about ten million words in the final DEREKO markup. The taz corpus has also been cast into this format, and all SfS and IMS tools were adapted accordingly, so that once the DEREKO corpus has been prepared by the IDS, it can be annotated and queried subsequently by the tools prepared by SfS and IMS.

Chapter 3

Linguistic Annotation

The main goal of the DEREKO corpus is to provide a large general purpose resource for the German language. A linguist using such a resource will expect detailed yet reliable information. The state of the art in syntactic annotation, however, shows that beyond the syntactic level of chunks, automatic syntactic annotation has to deal with rapidly increasing ambiguity, and consequently, the quality of automatic annotation declines.

For DEREKO, a finite-state approach to parsing was adopted, solving both the problems of speed and accuracy outlined above. Finite-state grammars can be applied efficiently, so that huge volumes of text can be processed quickly. Second, the phenomena that can be described by finite-state grammars coincide with those syntactic phenomena that are only moderately ambiguous. Annotation using finite-state grammars is very useful for linguistic research and language technology applications, as the overall syntactic ambiguity is reduced, and further annotation can take direct advantage of it, as is exemplified e.g. by the query collections presented in section 4.3. In the following sections the linguistic markup is presented in more detail (sect. 3.1 and 3.2), and the robust and efficient DEREKO linguistic annotation system is presented (sect. 3.3).

3.1 Morphosyntactic Annotation

Part of speech (POS) annotation is a robust standard technique used as a basis for further annotation in the DEREKO annotation system. In order to provide linguists with more fine-grained information, baseforms and morphological information have also been added for each word form token in the corpus.

Parts of Speech

The STTS tag set developed by IMS and SfS [26] defines the set of POS labels used in the DEREKO linguistic annotation. In order to train POS taggers appropriately, a resource of approx. 500.000 manually annotated word form tokens has been set up, consisting of texts from newspapers (approx. 315.000 tokens), novels (approx. 150.000 tokens) and other text types (ap-

prox. 30.000 tokens). The POS taggers are trained with either news texts, texts from novels, or all texts, and their most probable analyses are recorded in the linguistic markup. Currently the tnt and Brill taggers are used [4, 5]. The output of all taggers is combined to achieve a single solution by majority voting [30], and POSs are revised using input from syntactic annotation [22]. All information added by the tagging, correction, and voting steps is recorded in the markup so that researchers using DEREKO may access it easily to inspect or improve results.

Morphology and Baseform

Reducing full forms to baseforms provides a useful abstraction for query purposes, so that a corpus user may search for, e.g., all examples of a verb without explicitly listing all the verb's full forms. Consequently, baseforms are added in the DEREKO linguistic annotation using the DMOR (Deutsche Morphologie, [25]) tool from the IMS. Information from DMOR is also used to add all possible morphological analyses for each (word form, POS) tuple. A mapping from the STTS POS tags to the DMOR analyses has been devised keeping only information which is most useful for further syntactic annotation, including case, number, gender, person, and inflection type [29].

POS information is available for each word form in the corpus. Morphological and base form information is avaliable for all word forms covered by DMOR making the morphosyntactic annotation provided in DEREKO a reliable starting point for further annotation.[1]

3.2 Syntactic Annotation

The DEREKO linguistic annotation follows the model of topological fields standardly assumed by syntactic descriptions for German [14, 8, 12, 10]. In addition to topological fields, clauses and chunks are annotated in order to prepare unrestricted language data for global sentence analysis. The annotation task has been split to deal first with those structures that can be processed with finite-state techniques, and that take advantage of syntactic restrictions only. Verb argument structure, e.g., can be added later using more powerful formalisms (see [22]).

3.2.1 Chunks

Chunks are defined as non-recursive continuous kernels of phrases [2]. This means that chunks may contain chunks of other categories but that they may not contain chunks of the same category. A typical example to illustrate this is the treatment of PP-attachment in chunks. The DEREKO linguistic annotation does not resolve this issue because of the non-recursive nature of chunks (see figure 3.1: *a farmer from Bayrischzell*). As a consequence, some phrases may be split up, because in German, prepositional phrases may also modify adjectival phrases. The

[1] In the taz corpus, there are 3.168.413 word form types and 204.698.328 word form tokens, of which 8.591.257 are not analysed by DMOR, corresponding to 886.625 word form types. Of these, the eight most frequent punctuation forms sum up to 1.657.056 word form tokens, and 557.656 types occur only once.

prepositional chunk (PC) *durch jahrelange Fehlentscheidungen* in the example in figure 3.2[2] is part of a complex adjective phrase with the adjective *hochverschuldeten* as its head. Figure 3.2 shows two separate chunks, however, because an adjective chunk cannot contain another adjective chunk (like the one contained in the PC). As a noun chunk (NC) cannot contain another NC for the same reason, the PC in figure 3.2 is not part of the following NC, either. As a consequence, the article *der* cannot be in one chunk with *hochverschuldeten Bahn* as the structure is discontinuous now, because the PC is a constituent separating the article and the rest of the NC. Thus, constituents like articles or prepositions which cannot be attached to their respective chunks are left 'stranded'.

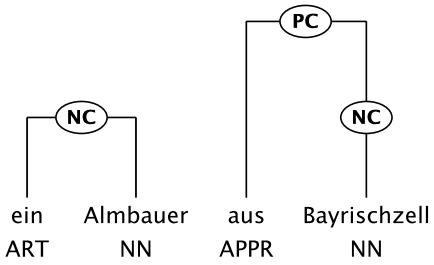

Figure 3.1: Postmodifying prepositional chunk

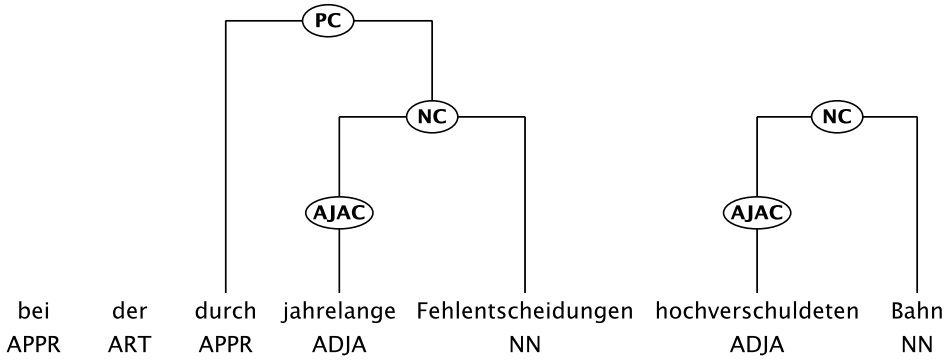

Figure 3.2: Chunk structure of a complex prepositional phrase

There are five major types of chunks. Verb chunks (VC_)[3], noun chunks (NC), adjective chunks (AJ_C), adverb chunks (AVC) and prepositional chunks (PC). Prepositional chunks always contain a noun chunk, and noun chunks may contain adverb chunks and adjective chunks. Adjective chunks may contain adverb chunks. Verb chunks are contained in no other chunk and they also cannot contain any other chunk.

[2] *at the by years-lasting misjudgements highly-indebted railways*; *at the railways, which was highly indebted due to misjudgements lasting years*

[3] The '_' stands for one letter if it occurs within a chunk name and for one or more letters if it occurs at the beginning or end of a chunk name.

As the annotation of verb chunks is quite complex, it is outlined only in general terms here (please cf. [21] for a detailed description). Verb chunks play a special role in the chunk structure, as they are both chunks and part of the clausal frame (and, thus, topological fields). This fact and the fact that verb chunks (and their combination) already contain a lot of information about the structure and type of the sentence they occur in has lead to a very fine-grained distinction among the different verb chunks, as opposed to the other kinds of chunks. Verb chunks are categorised on the basis of their syntactic distribution and their inner structure. As regards syntactic distribution, there are three main types of verb chunks: VCL-, VCR- and VCF-. VC[L]-chunks are chunks constituting the left part of the clausal frame and VC[R]-chunks constitute the right part of the clausal frame. VC[F]-chunks are chunks which in the basic word order would be the right part of the clausal frame but which are fronted and, thus, topicalised.

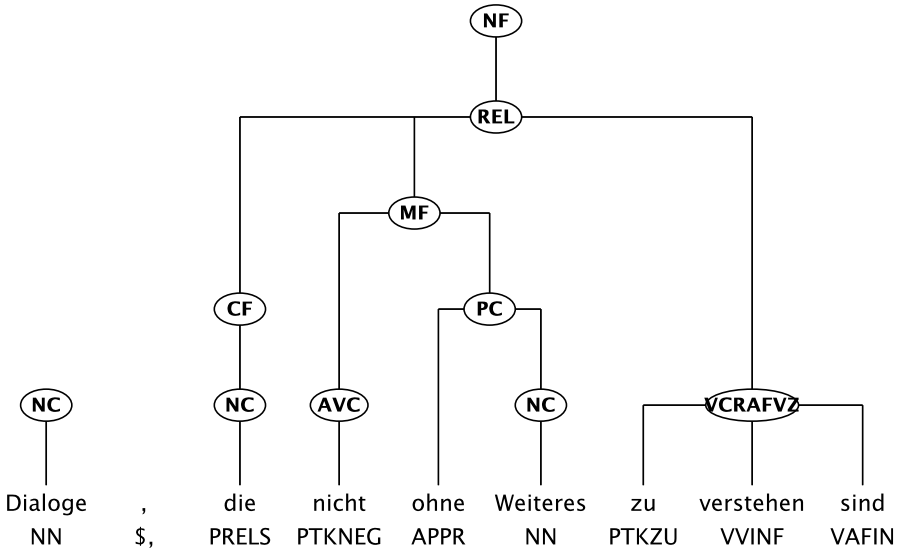

Figure 3.3: Verb chunk in relative clause

The following letters in the name of verb chunks correspond to the second and third letters in the verb's tag, which denote verb type (V=lexical, A=auxiliary, M=modal) and finiteness (F=finite, I=infinitive, P=perfect participle)[4]. The sequence of the verbs in the chunk type name corresponds to the syntactic dependence, which is the opposite of the sequence of the occurrence of the verbs in the sentence. Thus, in the sequence *Dialoge, die nicht ohne Weiteres zu verstehen sind* the verbal complex *zu verstehen sind* is assigned the chunk type VC[R] for right part of clausal frame, [AF] for auxiliary finite, [VZ] for lexical verb/infinitive with 'zu' (see figure 3.3: *dialogues, that not without further [explanation] to understand are*; *dialogues you cannot understand offhand*).

[4]An exception is being made in the treatment of the infinitive with 'zu', where 'I' is replaced with 'Z' in the chunk type name (see figure 3.3), and with the imperative, where a 'B' is assigned.

Table 3.1: Overview of the Chunk Labels

Chunk Label	Definition
AJAC	attributive adjective chunk
AJACTRUNC	AJAC with truncated adjective
AJACC	at least two coordinated AJACs
AJVC	predicative adjective chunk/adverb chunk
AJVCTRUNC	AJVC with truncated adjective/adverb
AJVCC	at least two coordinated AJVCs
AVC	adverb chunk
AVCC	coordinated AVC
NC	noun chunk
NCTRUNC	NC with truncated noun
NCC	two NCs coordinated by coordinator
NCell	elliptical noun chunk (i.e. without head noun)
PC	prepositional chunk
VC_	verb chunk
VCTRUNC	verb chunk with truncated verb
VCL_	verb chunk as left part of clausal frame
VCR_	verb chunk as right part of clausal frame
VCF_	verb chunk in topicalised (fronted) position

3.2.2 Topological Fields

Topological fields describe sections in the German sentence with respect to the distributional properties of the verb. In German, the verb complex is divided into two parts in the affirmative sentence with the finite verb coming first and the rest of the verb complex following later on to the end of the sentence. This construction is called the *Satzklammer* (clausal frame), the finite verb being the left part of the clausal frame (Linke Klammer, LK) and the non-finite verb complex being its right part (Rechte Klammer, RK). Thus, in the affirmative sentence, the sections preceding the finite verb may be called the *Vorfeld* (front field, VF), the section in the clausal frame the *Mittelfeld* (middle field, MF) and the section following the non-finite verb complex the *Nachfeld* (post field, NF). If clauses are coordinated, there may also be a KOORD-Feld (coordinator field, KOORDF). Sentences in which the verb is fronted (e.g. imperatives and yes/no-questions) are like affirmative sentences except that they lack a VF. Subordinated sentences are different in that the finite verb goes together with the verb complex (being the last element in it). They also lack a VF, and their LK is occupied by the complementiser field (CF).

The model of topological fields describes the distribution of constituents relative to the clausal frame. It is therefore primarily a distributional model, not giving any account of the verb-argument structure and not revealing the relation between the constituents within the topological fields, either. In fact, the very structure of constituents within topological fields is left open. It is, however, important to point out that a lot of constituent order phenomena can be described relative to topological fields.

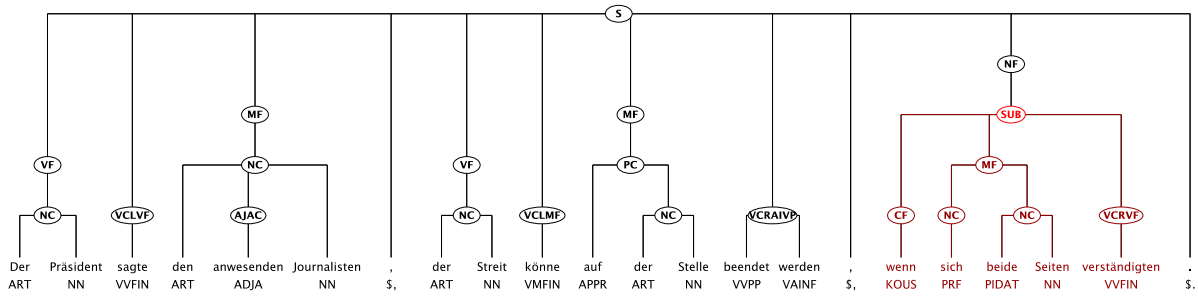

Figure 3.4: Complex Sentence

The advantages of the annotation of topological fields as a basis for the annotation of verb argument structure can best be illustrated by an example: Figure 3.4 shows a sentence containing five verbs in three verb complexes.[5] As the arguments may appear on both sides of the verb, it is by no means clear, where the respective arguments of the verb are to be found. After the annotation of the topological fields, however, the scope is reduced: The arguments of *sagte* – as long as they are single phrases – may only occur in the preceding VF and the following MF. The same is true of the following clause, where the field structure shows that *könne . . . beendet werden* is one verb complex and then it is clear that phrasal arguments may only occur in the VF and the MF, because the NF contains a clause. In that subclause it is again clear that the arguments of the verb must be in the MF.

Figure 3.4 illustrates how the scope of potential arguments is reduced by annotating field structure. It should be taken into account, however, that the annotation of field and clause structure is a shallow annotation in the same sense as the annotation of chunks is shallow. It is, thus, not always clear to which field or clause a subclause should be attached. In figure 3.4 it is left open whether the NF containing the adverbial clause is the NF of the preceding subclause or of the main clause (i.e. the NF of the whole sentence). The annotation also leaves it open whether the subclause *der Streit könne auf der Stelle beigelegt werden* is a subclause at all. The motivation for this approach is that without taking into account valency information, it is not clear whether a sequence of VF, LK and MF is a separate main clause in an asyndetic construction (like in *Das Parlament debattiert, der Präsident handelt*), or a subclause (like in figure 3.4). Still, the annotation of topological fields prestructures a sentence and serves as a solid base for further annotation.

3.2.3 Clauses

There are three different kinds of clauses annotated by the DEREKO linguistic annotation system. General subclauses (SUB), relative clauses (REL) and non-finite clauses (INF). A clause is defined as having one head verb (with the exception of coordinated head verbs). Main clauses are not annotated by the system, because this would be beyond the scope of shallow annotation. As a consequence, the relation between subclauses is not made explicit (the implications being

[5] *The president said the attending journalists, the quarrel could on the spot finished be, if themselves both sides agree. – The president said to the attending journalists, that the quarrel could be finished immediately, if both sides come to an agreement.*

discussed above). Generally, the attachment of subclauses is left open in shallow clause annotation just like the attachment of a prepositional chunk is left open in chunk annotation. This means for example that the reference noun of a relative clause is not specified.

The category REL subsumes all relative clauses introduced by relative pronouns. Clauses introduced by adverbial relative pronouns (i.e. PWAV) are annotated as SUB, because from the perspective of shallow annotation, it is not clear in which cases a PWAV is a relative pronoun at all. The category INF subsumes all non-finite clauses containing an infinitive (not those containing a participle), both introduced by a complementiser and non-introduced. The category SUB subsumes all other introduced subclauses, i.e. mainly adverbial clauses. Recursiveness is handled by applying the annotation cascade for subclauses twice. As the annotation of clauses is shallow, this does not affect sequential subclauses (because they are not attached at all), but only cases where embedding occurs in the MF. Thus, in the DEREKO linguistic annotation, a subclause never contains a subclause in its MF which again contains another subclause in its MF. These cases are rare, however, because centre-embedding is limited due to cognitive limitations.[6]

3.3 Implementing Robust Large-Scale Linguistic Annotation

The DEREKO linguistic annotation system has to annotate huge volumes of unrestricted text, integrating several types of linguistic information. Its implementation therefore focuses on speed and robustness, taking advantage of existing tools and markup standards.

Extending the CES for More Verbose Linguistic Annotation

XML is widely used as an encoding format and has proved to be very suitable for encoding linguistic phenomena due to its inherent tree structure. The DEREKO linguistic markup is therefore encoded in XML on output, and also for processing purposes.

The DEREKO linguistic annotation scheme extends the Corpus Encoding Standard (CES) [15] below the sentence level to cover, e.g. the annotation of chunks and topological fields, or information for POS tag majority voting. It therefore complements the IDS extensions applying to sentences and larger units.

The CES is extended using in-line linguistic annotation. The original proposal of using a standoff annotation scheme was not followed mainly for the following reasons:

- The original unannotated corpus can be reproduced from the annotated corpus, because e.g. all formatting information is preserved in the markup.
- A separate version of the unannotated corpus is mainly advisable when several orthogonal annotations are added later. The annotation scheme used in the DEREKO project,

[6] A counterexample is a sentence like: *Wenn Dein Freund, der, wenn er den Mund aufmacht, lügt, kommt, geh ich. (When your friend, who, whenever he the mouth opens, lies, comes, I leave; I leave when your friend comes who lies whenever he opens his mouth.)*

however, is considered to be a starting point for further annotation itself.

- It is assumed that the DEREKO linguistic annotation will be useful for most applications. Processing links from the annotation to the original corpus (i.e. using stand-off annotation) may outweigh the benefit of having a smaller source, when the links have to be followed for the vast majority of applications.
- Many current XML processing tools support efficient processing of a stream of XML data, and standards for linking XML documents are not yet as widely supported.

The DEREKO linguistic annotation system covers sentences and smaller textual units:

Tokens are the basic units of all subsequent annotation. They are detected with a finite-state grammar recognising also more complex combinations of tokens appearing, e.g., in dates, or in numbers followed by measurement units (i.e. a basic named entity recognition is performed). Phenomena recognised in the tokenisation step include clitics (`<t i=" f='gib'/><t i='CLITIC' f="'s"/> <t i=" f='mir'/>`) and urls (`<t i='URL' f='http://www.sfs.uni-tuebingen.de//>`). A conservative approach was followed, preferring to group less characters to form a token when a decision is uncertain, so that, e.g. "`17. Juni`" is recognised as `<t i='DAY' f='17.'/> <t i='MONTH' f='Juni'/>`, but, e.g. "`das 100. Mal`" is tokenised as `<t i=" f='das'/> <t i='NUM' f='100'/><t i='PUNCT' f='.'/> <t i=" f='Mal'/>`, because the full stop may either belong to the preceding token or not.

Each token is assigned one or more POS tags, and each (token, POS) tuple is assigned one or more lemmas and morphological ambiguity classes. Figure 3.5 shows how the XML tree structure is used to encode the possibly ambiguous analyses. In figure 3.5, the word form "`der`" has two POS analyses (article or relative pronoun) and the article is the preferred reading (`<P t='ART' r='1'/><P t='PRELS' r='2'/>`). Each POS in turn may have more than one baseform, that again is connected to one or more morphological analyses ("`der`" as an article may either be nominative, and the corresponding baseform is "`der`", or dative or genitive, and the corresponding baseform is "`die`"). All information may come from more than one source, and each source may assign a certainty to its analysis (`<j n='TaggerA/News' c='0.3'/>`). Including verbose information for POS tagging, lemma and morphology, offers researchers the opportunity to check second-best POS analyses, or disambiguating morphology, which in the present system is not yet accomplished.

All whitespace in the original document is recorded between the token (`<t/>`) elements, so that the annotation is non-destructive with respect to the source text. Tokenisation and the corresponding markup is described in more detail in [29].

Chunks, Topological Fields and Clauses are all encoded as separate elements wrapping the structure they dominate (as `<ch/>`, `<fd/>` and `<cl/>` elements, respectively). They all have an attribute (`c`) encoding the specific kind of chunk, field, or clause, as in e.g. `<ch c='NC'><t f='Dialoge'/></ch>` (omitting sub-token information). The categories and annotation strategies are described in more detail in [29].

```
<t f='der' i=''>
  <P t='PRELS' r='2' c='0.2'>
    <j n='TaggerA/News' c='0.3'/> <j n='TaggerB/News' c='0.1'/>
    <b f='der'> <j n='MorphA' c='1'/> <m d='nsm'/> </b>
    <b f='die'> <j n='MorphA' c='1'/> <m d='dsf'/> </b>
  </P>
  <P t='ART' r='1' c='0.8'>
    <j n='TaggerA/News' c='0.7'/> <j n='TaggerB/News' c='0.9'/>
    <b f='der'>
      <j n='MorphA' c='1'/> <m d='nsm'/> </b>
    <b f='die'>
      <j n='MorphA' c='1'/> <m d='dsf'/><m d='gp0'/><m d='gsf'/>
    </b>
  </P>
</t>
```

Figure 3.5: Encoding Alternative POS, Lemmas and Morphological Analyses

The output produced by the DEREKO linguistic annotation system can be imported straightforwardly into the query tool described in chapter 4.

Architecture of the DEREKO Linguistic Annotation System

The annotation system is implemented as a cascade of tools all adding certain information to a common XML data structure, which can be considered to be the *blackboard* of the DEREKO linguistic annotation system. Each tool relies on information that has been added in previous steps, and the information it adds is in turn used by tools later in the cascade. The parts of the cascade are grouped hierarchically, because groups of tools can be identified solving a major task like segmenting text, or adding morphosyntactic and syntactic information to the text (see figure 3.6, left hand side). Each group, again, has sub-groups that solve subtasks of the major task. The major task of morphosyntactic annotation, e.g., starts with the subtask of adding POS information. In the POS tagging subtask, several taggers trained with different data add their information one after the other. After all taggers have finished, the POS tag is determined via majority voting, finishing the POS subtask (see figure 3.6, right hand side).

Separating linguistic annotation into clear-cut tasks, sub-tasks, and tools, has several advantages:

- Hypothesis Testing

  All related tools in a subtask may be easily regrouped and executed on the same input data. For the noun chunk annotation, e.g., several hypotheses could be quickly evaluated, and the best tool and subtask sequence was found rapidly, resulting in a top-down bottom-up approach.

- Extensibility

  New processing steps may be introduced anywhere without changing the rest of the system. Obvious places for new steps include the input end, where filters are already avail-

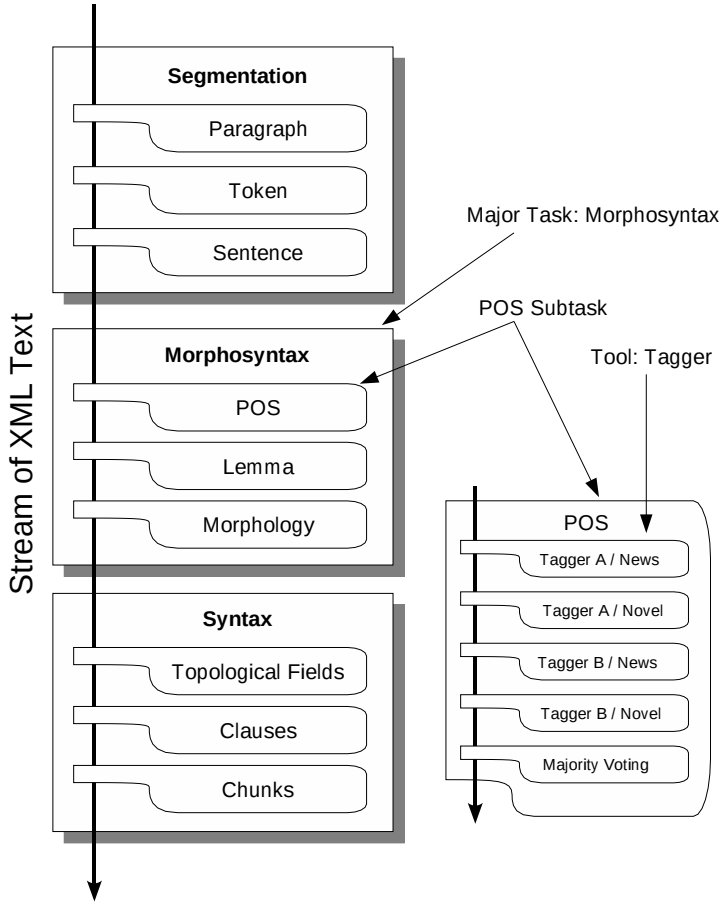

Figure 3.6: Architecture of the Linguistic Annotation System

able that process, e.g. plain text or HTML in addition to the native DEREKO format[7], or the output end, where more refined annotation steps may follow. However, the system may also be extended anywhere mid-stream. In order to improve a well-known POS tagging problem, e.g., a tag correction system has been applied immediately after the clausal frame is annotated without changing the rest of the cascade [22].

- Robustness

  Subsequent annotation steps may take advantage of previous steps, e.g. chunks are usually annotated only within topological fields. However, if any annotation steps fail, subsequent steps may still annotate the input text ignoring the failed steps. In case, e.g., a verb frame and the corresponding topological fields cannot be detected correctly due to POS tagging errors, chunks will still be annotated, using the syntactically unannotated sequence of tokens in a sentence. The resulting sentence contains less detailed but still useful syntactic annotation.

[7]When processing the native DEREKO format prepared by the IDS, the paragraph and sentence segmenting subtasks are dropped.

- Scalability

  Each major task, sub-task, or tool may be executed on a different computer, or on a different processor for multi processor systems. The tasks may be executed parallely, where a single data stream connects all tasks on all computers. Computers may also be chosen that excel in certain tasks (e.g. with either a fast processor or quick disk access). Only the slowest tool on the fastest processor limits overall throughput.

- Debugging

  An error may be spotted quickly by executing all tasks, subtasks, or tools individually, only stopping when the error occurs. All annotation is given in the common input and output format, and tools have been devised for browsing it, that can be used to inspect the output of each single tool.

- Reusing Existing Tools and Libraries

  Using XML as the only connection between the tools lets you incorporate a growing number of general-purpose XML tools or tools supporting rapid prototyping (e.g. `xmlperl` or `fsgmatch` [11]). If present tools lack sufficient speed, a specialised tool can be implemented quickly using high performance XML libraries which are available as open source software (e.g. the majority voting tool has been implemented along these lines).

The tools developed for the annotation of the DEREKO corpus have wide applicability beyond the tasks for which they were developed. Because of their modular design and implementation, modules can be reused for a wide variety of language technology applications, including as preprocessors for deeper linguistic analysis, term extraction, and other information-retrieval tasks, as well as for the construction of language models in speech recognition applications. The annotated corpora themselves can be used as training material for statistical parsers, as gold standards for the evaluation of NLP systems, and as data repositories for linguistic research.

Chapter 4

Corpus Exploitation

In order to make use of a linguistically annotated text corpus, tools for corpus exploitation are needed. We have developed a treebank query language (Sect. 4.1) and implemented this language, i.e. the treebank query engine 'TIGERSearch' (Sect. 4.2). Furthermore, we have built query collections which address specific lexicographic tasks (Sect. 4.3). For efficiency reasons, a certain level of the query results is also annotated to the corpus in order to complement the corpus annotation, which has been described in the previous sections.

4.1 A query language for structurally annotated corpora

In cooperation with the TIGER project, a specialized corpus query language has been designed. This language has been described in full detail in [18] resp. [19]. Subsequently, we will give an overview of the major features of the language.

A syntactic derivation consists of nodes and relations among nodes. Nodes can be queried by Boolean expressions over feature-value pairs. The text corpora in DEREKO have been annotated with the `word` and the `pos` (part-of-speech) feature on the level of words, and with the `cat` (syntactic category) feature on the level of derived nodes. A possible query would be:

```
[word="Abend" & pos="NN"]
```

Relations among nodes are expressed in terms of the two elementary operators `.` (direct precedence) for the horizontal ordering and `>` (direct dominance) for the vertical ordering. There is a set of derived node relations like `>*` (dominance) and `.*` (precedence) etc. Sample query:

```
[cat="PC"] > [pos="APPRART"]
```

Relational expressions can be combined by (restricted) Boolean expressions (negation excluded). Example:

```
([cat="NC"] >* [pos=/AD.*/]) & ([cat="NC"] > [pos="NE"])
```

The example above shows that regular expressions can be used to describe feature values (`/AD.*/`).

The two node descriptions `([cat="NC"]` can refer to different nodes. The identity of the two `NC` nodes can be enforced by inserting a variable:

```
(#x1:[cat="NC"] >* [pos=/AD.*/)
& (#x1:[cat="NC"] > [pos="NE"])
```

In addition, the user can define a type hierarchy. Types are very useful to formulate queries in a more concise way. For example, one could define a type `nominal` which stands for the possibly less intuitive regular expression `"N.*"`:

```
[pos=nominal] .* [pos="VVFIN"]
```

For the convenience of the users, a series of predicates has been added. For example, if more complex annotation were added to the DEREKO corpus, one could impose that a structure be discontinuous like in the case of an NP which is modified by an extraposed relative clause:

```
#np:[cat="NP"] & (#np >RC []) & discontinuous(#np)
```

4.2 The treebank search engine TIGERSearch

The query language has been realized by a specialised search engine, TIGERSearch. In order to create a really useful tool, the following modules have been added:

- sophisticated graphical user interface (TIGERGraphViewer) for browsing the query results
- graphical registry tool (TIGERRegistry) for easy corpus administration
- XML-import of corpora

  Import filters are available for the DEREKO corpus format and for a range of other well-known corpora (PennTreebank [23], NEGRA [28], TIGER [7], Susanne and Christine [24], Penn-Helsinki Parsed Corpus of Middle English [1], VerbMobil [13])
- XML- and SVG-animation-export of query results

  Sample XSLT-stylesheets for the creation of some other formats (e.g. plain text) have been included.

The different modules work together as shown in Fig. 4.1. The TIGERRegistry tool provides import filters for the transformation of various treebank formats into the TIGER-XML format. On this basis, the TIGERSearch indexer can be called in order to create a corpus index. A query is processed by parsing it, transforming it into a Java object. This query object is then evaluated against the corpus index. Query result objects can be either transformed into TIGER-XML for further processing, or can be viewed by the GraphViewer.

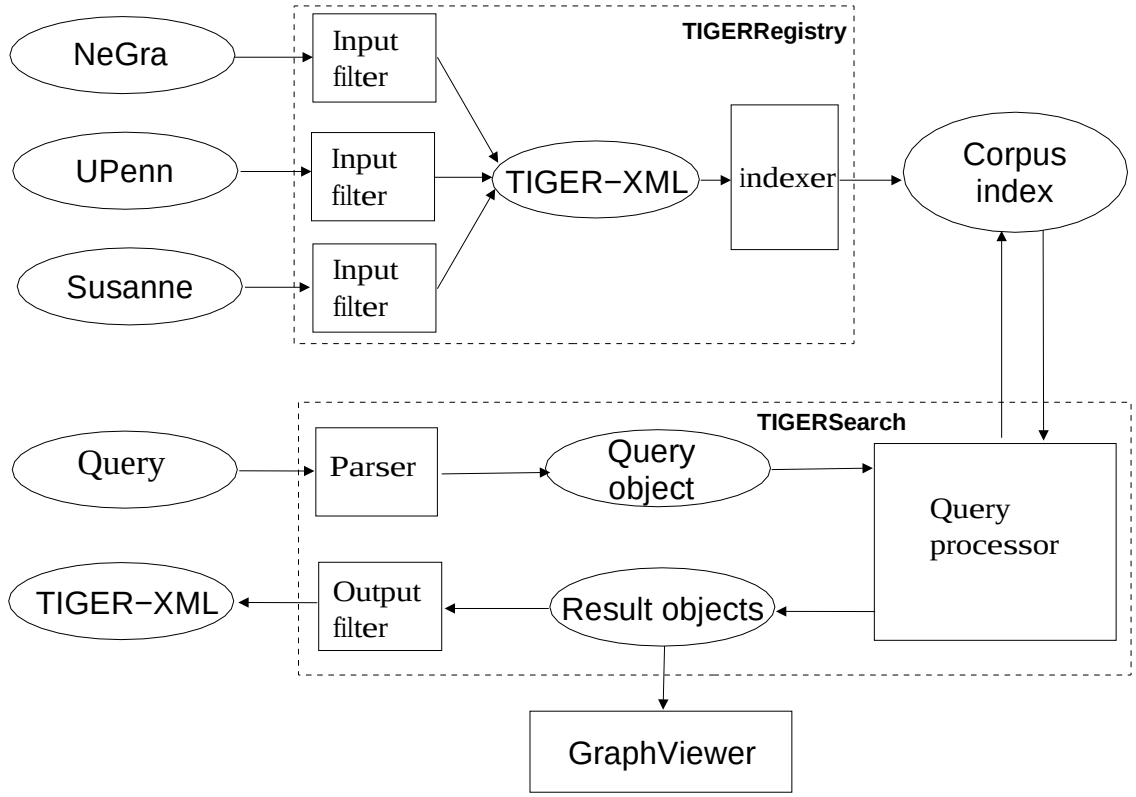

Figure 4.1: Architecture of the TIGERSearch software

The TIGERSearch software has been implemented in Java for several reasons:

- platform-independence

  TIGERSearch is available for all major platforms which support Java 1.3: Microsoft Windows, Solaris, Linux, and Mac OS X.

- realization of software engineering standards

  Java as an object-oriented language makes it much easier to write conceptually well-defined programs. Various error sources, which occur in other programming languages, are excluded by the rigid type checking by the Java compiler.

- support of client-server applications and multi-threads.

  Since Java provides built-in functionality for distributed computing, the TIGERSearch implementation can be easily adapted to future requirements (web-based applications, parallel computation on large corpora).

- Unicode support

  Due to the Java built-in Unicode support, TIGERSearch can be used with corpora in non-Western scripts. (At the current stage, e.g. a Japanese corpus could be imported into TIGERSearch, but there is not yet a convenient manner to key in the Japanese characters for querying the corpus.)

Manuals and download information for TIGERSearch can be obtained from the TIGERSearch website:

`http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERSearch/`

Subsequently, we give a little more detail wrt. the search engine and the graph viewer.

4.2.1 Search engine

When we started to think about the search engine, we had to decide whether to build a new, dedicated engine from scratch, or whether to use an existing database engine (c.f. [16]). We decided for the first alternative of building a new engine for the following reasons:

- Genuine search machines for XML-based (i.e. hierarchically structured) data were hardly available when we started the project in 1999.
- The linguistically motivated corpus representation format we had chosen included one (or several) complications in addition to the hierarchical information: crossing edges and so-called 'secondary' edges [27]. It was not clear, whether so-called XML-based databases would efficiently support the access to 'near-trees' which violate the strict embedding conditions of proper trees.
- Efficient database engines, in the way they are required to process large corpora, usually are very expensive commercial software - which cannot be afforded by the intended TIGERSearch users (researchers in the humanities). The reason why commercial database software is expensive is not only the amount of work which went into efficient search algorithms but also the cost for modules like transaction management, database recovery mechanisms etc. However, for corpus applications, only the (efficient) search module is needed.
- In our own implementation, the indexing mechanisms, query optimization, and query filtering can be tuned more easily to the properties of the kind of data and queries which occur in linguistic applications.

In order to ensure efficient query processing we have chosen an index-based approach. Data structures are built over the text representation of the original corpus and transformed into a binary corpus representation. In addition, many partial searches are performed during indexing in order to save processing time during query processing.

The query evaluation algorithm is a straight-forward implementation of chronological backtracking. A query is serialized as a list of tree nodes and their respective constraints. This list

is traversed from left to right. A query node is matched with a node from the corpus, and node relations are checked against the corpus definition by accessing the index. Whenever a conflict occurs, the algorithm discards the current choice, and chooses another matching node from the corpus. Once no more alternatives are available, the choice for the previous query node is reconsidered. Instead of this 'depth-first' search, which builts one result tree at a time, a breadth-first search based on 'join' operations could have been adopted in the way it is used in conventional database management systems. However, considering the implementation of a rather general concept of logical variables, the backtracking approach seemed more straight-forward. In addition, the depth-first approach is less space-consuming. The drawback is that, in its naive version, all nodes of the corpus have to be inspected, one after the other. For this reason, the base algorithm has been improved in two respects:

- search space filters

  If possible, the query is logically weakened to one of its subformulas, an atomic constraint (feature-value equation), which matches only a very limited number of corpus nodes, e.g. `[word="Haus"]`. It is sufficient to evaluate the remainder of the query formula on this small corpus extract (cf. [3]).

- query optimization

  Search space filters make only sense, if they can be determined at low cost and produce a drastic reduction of the search space. Therefore, additional efficiency can be gained by reordering the (atomic) subformulas of a query according to their prospective 'evaluation cost' in terms of number of matching corpus nodes, cf. [20]. Since the 'cost' can be read off from the index, the subformula reordering operation is fast, and helps to cut the evaluation time even in those cases, where a search space filter cannot reduce the search space in a reasonable manner.

There is ample space for further work on treebank-specific search space filters and query optimization. The problem is to find filters and optimization strategies which reduce the search effort drastically while being identifiable at low cost.

4.2.2 Graph viewer

The GraphViewer allows for the graphical display of the results of a TIGERSearch query. Its major features are:

- navigation through the list of result sentences
- navigation through matching subgraphs
- focussing on match
- imploding and exploding of subgraphs
- tooltip function (node features appear when mouse points to the node)

- printing and export of various graphics formats (jpeg, png, tiff, animated SVG images)
- customisability: users can set colors and other display parameter according to their own requirements

Since there is a clear-cut interface between the TIGERSearch engine and the GraphViewer, the GraphViewer can be seen as a stand-alone tool to be used with other applications as well.

4.3 Query collections

One application of a linguistically annotated corpus is to give evidence for certain lexical properties of words, cf. [9]. When deciding about the content of a lexical entry, one has to take into account at least the two following criteria:

- 'collocational' variation

  In which syntactic and collocational contexts does a word occur?
- frequency considerations

  How frequent are these occurrences? Are they worth being listed?

Let's consider the collocation pattern

(Support)Verb *. . . sicher, dass* (4.1)

The rather comprehensive *Duden Universalwörterbuch der Deutschen Sprache* [17] lists only one possible support verb (*sein*) for this kind of pattern:

Es ist sicher, dass er zustimmt.
Ich bin sicher, dass er noch kommt. (4.2)

Corpus investigation shows that, of course, *sein* is the most frequent verb in co-occurrence with *sicher, dass*, but that there are others, which are more or less common:

verb		freq.	example
sein	*sicher, dass*	*113*	
gelten	*sicher, dass*	*11*	*Es **gilt** als sicher, dass George Bush sein Veto einlegt.*
stellen	*sicher, dass*	*8*	*Ein Gesetz **stellt** sicher, dass von den kassierten Mitteln zehn Prozent für Verwaltungsaufwand abgezogen werden dürfen.*
wissen	*sicher, dass*	*2*	*. . . **wusste** ich sicher, dass ich jeden Tag wiederkommen kann.*

The frequency figures (for *dass*-clauses in the Nachfeld) have been extracted from a newspaper corpus with 40 million tokens.

In order to extract such corpus evidence in a reliable manner, a rather detailed syntactic analysis is required. The syntactic analysis must cover complement clauses (*dass*-clauses) and word order variation of verbs in German. As argued for in Sect. 3.2, we use finite-state based shallow syntactic analysis methods, and, in our case, the finite-state based corpus query language of the IMS Corpus Workbench (CQP) [6]. Since the development of the search engine for structurally annotated corpora (TIGERSearch) ran in parallel to the implementation of the query collections, the latter have been realized with CQP. The query collections can be easily ported to TIGERSearch, once it supports mechanisms for query abbreviations (templates) and corpus transformation.

Our CQP queries have been grouped in two layers:

- The first layer of CQP queries builts recursive syntactic structures of restricted depth of embedding which can be discovered in a reliable manner, i.e. which are usually not ambiguous with other possible analyses (as it would be case with e.g. PP-attachment).
- The second layer of CQP queries extract corpus evidence for specific lexicographic tasks.

For more efficient CQP-internal processing, the query results of the first layer are added to the corpus annotation, so that the basic syntactic analyses of the first layer do not have to be reconstructed for each query of the second layer.

On top of the chunk annotation, which has been described in Sect. 3.2, the first layer of queries identifies the following syntactic phrases:

- pre-head recursive embedding, e.g., PPs modifying APs, which are left stranded by the analysis in Sect. 3.2, are assembled to form complex APs, as the tree diagram in Fig. 4.2 exemplifies. This leads to the recursive embedding of the NP *die Köpfe der Apostel* (the heads [of] the apostles) in the NP *kleinen, über die Köpfe der Apostel gesetzten Flammen* (small, above the heads of the apostles set flames).
- post-head recursive embedding of
  - genitive NPs. In Fig. 4.2 the genitive NP *der Apostel* ([of] the apostles) is embedded in the NP *die Köpfe der Apostel* (the heads [of] the apostles) as a modifier.
  - named entities. In the example *einer Fahrt des Shuttle "Endeavour"* (a trip of the shuttle "Endeavour") the NP *"Endeavour"* is embedded as a modifier in the larger NP
- PP-attachment in some specific cases, e.g., in NP constructions including parenthesis. In *eine Wegbeschreibung durch die Hölle* ([a] directions through [the] hell), the PP *durch die Hölle* is attached to the preceding head noun, the whole construction being surrounded by double quotes.
- chunks with specific internal structures, e.g., news agency markers (*LONDON (adp)*) or measure adjective phrases (*10 Hektar größer* (10 acres larger)).

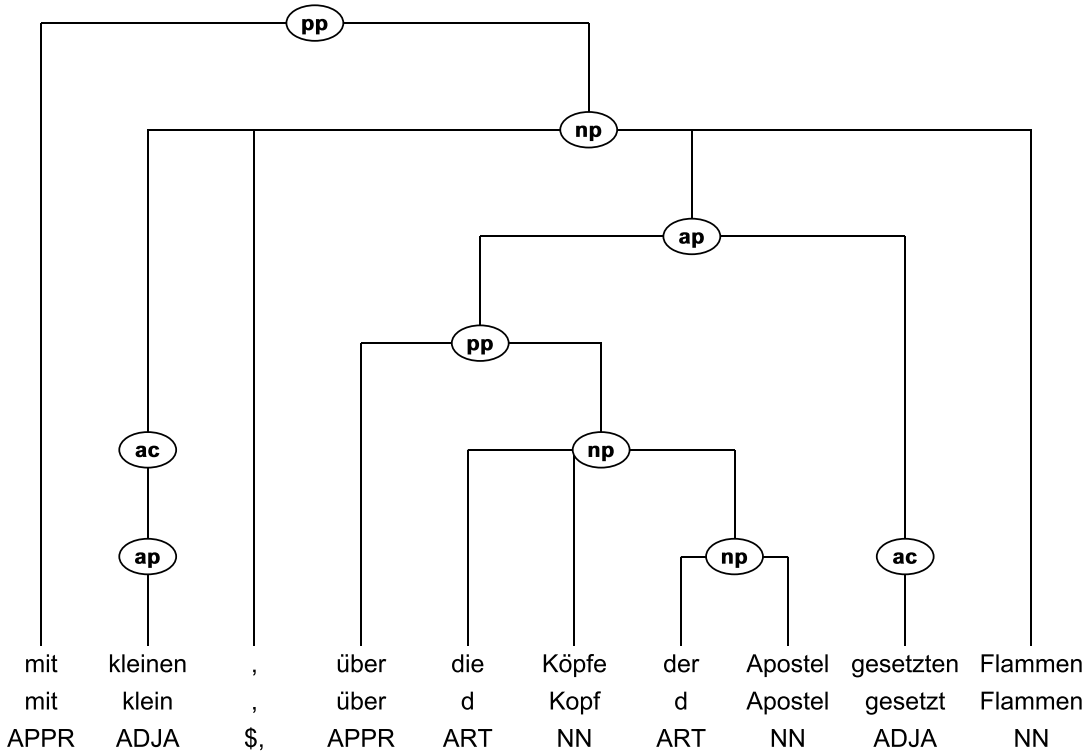

Figure 4.2: Full hierarchical structure of complex AP

Besides the additional syntactic analysis, certain lexical properties of terminal nodes are added. They are projected from the head to the chunks where necessary. The lexical information is used during the parsing process to specify restrictions for parsing rules. The rule that builts APs, e.g., allows only NCs with the feature attribute *year* to modify the adjective in pre-head position. The result are APs such as $[_{NP}$ *die* $[_{AP}$ $[_{NC}$ *1993*] *erbaute*] *Brücke*] (the 1993 built bridge; the bridge built in 1993).

The lexical properties of chunks are also used during extraction. The query can either look for chunks with specific lexical properties or it can exclude chunks with a specific property. When looking for subcategorization frames, e.g., one might want to exclude NPs with temporal heads, whereas, in the case of collocation extraction, named entities are not of interest. As these properties are annotated along with the chunks, it is easy to exclude them from the extraction results.

The chunks provided by the annotation described in Sect. 3.2 are used according to their category (e.g., NC, AC, or PC) and included in the CQP-Analysis to build up larger chunks where necessary and possible, otherwise they are left as they are.

Once all the query results of the first query layer have been annotated, the queries of the second layer which are to extract the corpus evidence can make use of the annotated chunks. As a big part of the analysis has already been performed off-line, the extraction query can leave this part of the analysis aside and can itself be kept simpler. Instead of rebuilding the rules for chunks such as NP, AP, PP anew each time, extraction queries can simply refer to the annotated structures.

Due to the incremental layers of queries, we have a better control on the way in which syntactic analysis proceeds. For example, in the off-line structure building process, one can allow for overgeneralizations (by relieving some syntactic restrictions), which can be corrected on the next level of analysis. Consider, for example, the rule which builds an AP with an embedded PP. In an intermediate step any adjective is allowed to take any number of preceding PPs as complement, knowing that quite a bit of noise is produced. At the end of the off-line work, these 'insecure' structures are checked for correctness. The structures are accepted if they prove to be in a secure context, otherwise they are rejected. In order to be accepted, APs embedding PPs have to be inside an NP with at least one 'secure' element preceding it. In (4.3), the AP *wegen des Regens nasse* is rejected and replaced by the smaller AP *nasse*, the same for the insecure NP, which is replaced by an NP to comprise only *nasse Füße* instead of the rejected insecure AP and the head noun. In (4.4), however, the AP *wegen des Regens nassen* is not rejected as the article *die* ensures the correctness of the AP.

Lisa hat [NP [AP [PP wegen des Regens] nasse] Füße] bekommen.
Lisa has because of the rain wet feet got. (4.3)
Lisa has got wet feet because of the rain

Lisa hat [NP die [AP [PP wegen des Regens] nassen] Füße] abgetrocknet.
Lisa has the because of the rain wet feet dried. (4.4)
Lisa has dried the feet that were wet because of the rain.

Note that we have implemented dedicated grammatical descriptions which have certain limitations. For example, the query which extracts the examples for the *sicher, dass* contexts also extracts false positives like the verb *aufregen* in *Bei der letzten war ich kein bißchen aufgeregt und mir sicher, dass der Ball reingeht*. The false positive is a result of the fact that the query is still very general and not restricted enough. It does not identify the elliptical construction of the example. This problem could be solved by restricting the query to a greater extent, i.e. by specifying the elements preceding the adjective in more detail. In addition, examples with a dative object would have to be filtered out.

The corpus query for the pattern (4.1) has been coded in such a manner that it can be used to extract the same type of syntactic pattern for other adjectives as well. In addition, the query is parametrisable with respect to the position of the *dass*-clause. In total, we have implemented parametrisable corpus queries for the following lexical and syntactic phenomena:

- adjectives in control verb constructions taking other subordinate clauses
- adjectives in control verb constructions taking infinitival clauses
- verb noun collocations
- verb noun collocation for a specific noun
- verb PP collocations with the preposition *an*
- adjective PP pairs extracted from complex APs, resulting in:

- adjective PO pairs
- verb PO pairs (PP occurring with deverbal adjective)

In joint projects with publishing houses, our query collections proved to be of great help for the lexicographers' work.

Bibliography

[1] Penn-Helsinki parsed corpus of Middle English, 2000.

[2] Steven Abney. Partial Parsing via Finite-State Cascades. In *Proceedings of the ESSLLI-96 Workshop on "Robust Parsing"*, Prague, 1996.

[3] Hans Argenton. *Indexierung und Retrieval von Feature-Bäumen am Beispiel der linguistischen Analyse von Textkorpora*, volume 40. infix Verlag, Sankt Augustin, 1998.

[4] Thorsten Brants. TnT – a statistical part-of-speech tagger. In *Proceedings of the 6th Conference on Applied Natural Language Processing (ANLP-2000), April*, Seattle, WA, 2000. find.

[5] Eric Brill. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing*, pages 152–155, Trento, IT, 1992.

[6] Oliver Christ, Bruno M. Schulze, and Esther König. *Corpus Query Processor (CQP). User's Manual*. Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Stuttgart, Germany, 1999.

[7] Stefanie Dipper, Thorsten Brants, Wolfgang Lezius, Oliver Plaehn, and George Smith. The TIGER treebank. Presented at the Third Wokshop on Linguistically Interpreted Corpora (LINC), 2001.

[8] Erich Drach. *Grundgedanken der Deutschen Satzlehre*. Diesterweg, Frankfurt/Main, 1937.

[9] Judith Eckle and Ulrich Heid. Extracting raw material for a German subcategorization lexicon from newspaper text. In *Proceedings of the 4th International Conference on Computational Lexicography, COMPLEX'96*, Budapest, Hungary, 1996.

[10] Oskar Erdmann. *Grundzüge der deutschen Syntax nach ihrer geschichtlichen Entwicklung*, volume Erste Abteilung. Cotta, Stuttgart, 1886.

[11] Claire Grover, Colin Matheson, Andrei Mikheevy, and Marc Moens. Lt ttt - a flexible tokenisation tool. In *2nd International Conference on Language Resources & Evaluation (LREC 2000), 31 MAY - 2 JUNE 2000*, Athens, Greece, 2000.

[12] S. H. A. Herling. über die Topik der deutschen Sprache. In *Abhandlungen des frankfurterischen Gelehrtenvereins für deutsche Sprache*, volume Drittes Stück, pages 296–362, 394. 1821.

[13] Erhard W. Hinrichs, Juli Bartels, Yasuhiro Kawata, Valia Kordoni, and Heike Telljohann. The Verbmobil treebanks. In Schukat-Talamazzini Ernst G. and Werner Zühlke, editors, *KONVENS-2000 Sprachkommunikation*, pages 107–112. VDE-Verlag, 2000.

[14] Tilman Höhle. Der Begriff 'Mittelfeld', Anmerkungen über die Theorie der topologischen Felder. In *Akten des Siebten Internationalen Germanistenkongresses*, pages 329–340, Göttingen, 1986.

[15] Nancy Ide, Patrice Bonhomme, and Laurent Romary. XCES: An XML-based encoding standard for linguistic corpora. In *2nd International Conference on Language Resources & Evaluation (LREC 2000), 31 May–2 June 2000*, Athens, Greece, 2000.

[16] Laura Kallmeyer. A query tool for syntactically annotated corpora. In *Proceedings of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, Hong Kong, October 2000.

[17] Annette Klosa, Kathrin Kunkel-Razum, Werner Scholze-Stubenrecht, and Matthias Wermke. *DUDEN Deutsches Universalwörterbuch*. Dudenverlag, Mannheim, 2001.

[18] Esther König and Wolfgang Lezius. The TIGER language - a description language for syntax graphs. Part 1: User's guidelines. Technical report, IMS, University of Stuttgart, 2001.

[19] Esther König and Wolfgang Lezius. The TIGER language - a description language for syntax graphs. Part 2: Formal definition. Technical report, IMS, University of Stuttgart, 2001.

[20] Jason McHugh. *Data Management and query processing for semistructured data*. PhD thesis, Department of Computer Science, Stanford University, March 2000.

[21] Frank Henrik Müller. Shallow parsing style-book for german. Technical report, Seminar für Sprachwissenschaft, Universität Tübingen, 2001.

[22] Frank Henrik Müller and Tylman Ule. Satzklammer annotieren und Tags korrigieren. Ein mehrstufiges Top-Down-Bottom-Up-System zur flachen, robusten Annotierung von Sätzen im Deutschen. In Henning Lobin, editor, *Proceedings der GLDV-Frühjahrstagung 2001*, pages 225–234, Gießen, 2001. Gesellschaft für Linguistische Datenverarbeitung.

[23] Penn Treebank Project, University of Pennsylvania. *Bracketing Guidelines for Treebank II Style*, 1995.

[24] Geoffrey Sampson. *English for the Computer: The SUSANNE Corpus and Analytic Scheme*. Clarendon Press (Oxford University Press), 1995.

[25] Anne Schiller. DMOR: Benutzerhandbuch. Technical report, IMS, University of Stuttgart, 1995.

[26] Anne Schiller, Simone Teufel, Christine Stöckert, and Christine Thielen. Vorläufige Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report, Universität Stuttgart, Institut für maschinelle Sprachverarbeitung, and Seminar für Sprachwissenschaft, Universität Tübingen, 1995.

[27] W. Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. An annotation scheme for free word order languages. In *ANLP, 1997*, 1997. http://www.coli.uni-sb.de/ thorsten/anlp97/.

[28] Wojciech Skut, Thorsten Brants, Brigitte Krenn, and Hans Uszkoreit. A linguistically interpreted corpus of German newspaper texts. In *ESSLLI Workshop on Recent Advances in Corpus Annotation*, Saarbrücken, 1998. University of Saarbrücken.

[29] Tylman Ule. DEREKO linguistic markup. Technical report, Seminar für Sprachwissenschaft, Universität Tübingen, 2001.

[30] Hans van Halteren, Jakub Zavrel, and Walter Daelemans. Improving data driven wordclass tagging by system combination. In *Proceedings of COLING-ACL '98, August*, Montreal, Canada, 1998. Association for Computational Linguistics.