# A Shallow Text Processing Core Engine

Günter Neumann, Jakub Piskorski

German Research Center for

Artificial Intelligence GmbH (DFKI), Saarbrücken

Address correspondence to Dr. Günter Neumann at DFKI, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany; e-mail: neumann@dfki.de

## Abstract

In this paper we present[1] SPPC, a high-performance system for intelligent extraction of structured data from free text documents. SPPC consists of a set of domain-adaptive shallow core components that are realized by means of cascaded weighted finite state machines and generic dynamic tries. The system has been fully implemented for German; it includes morphological and on-line compound analysis, efficient POS-filtering, high performance named entity recognition and chunk parsing based on a novel divide-and-conquer strategy. The whole approach proved to be very useful for processing free word order languages like German. SPPC has a good performance (more than 6000 words per second on standard PC environments) and achieves high linguistic coverage, especially for the divide-and-conquer parsing strategy, where we obtained an f-measure of 87.14% on unseen data.

*Key words*: natural language processing, shallow free text processing, German language, finite-state technology, information extraction, divide-and-conquer parsing

[1]The paper is based on previous work described in (Piskorski and Neumann, 2000) and (Neumann, Braun, and Piskorski, 2000), but presents substantial improvements and new results.

## 1 Introduction

In the majority of current large–scale information management approaches linguistic text analysis is restricted to be performed on the word level (e.g., tokenization, stemming, morphological analysis or part-of-speech tagging) which is then combined with different word occurrence statistics. Unfortunately such techniques are far from achieving optimal recall and precision simultaneously. Only in few research areas viz. information extraction (Grishman and Sundheim, 1996; Cowie and Lehnert, 1996) or extraction of ontologies from text documents some approaches (e.g., (Assadi, 1997)) already make use of partial parsing. However, the majority of current systems perform a partial parsing approach using only a very limited amount of general syntactic knowledge for the identification of nominal and prepositional phrases and verb groups. The combination of such units is then performed by means of domain-specific relations (either hand-coded or automatically acquired). The most advanced of today's systems are applied to English text, but there are now a number of competitive systems which process other languages as well (e.g., German (Neumann et al., 1997), French (Assadi, 1997), Japanese (Sekine and Nobata, 1998), or Italian (Ciravegna et al., 1999)).

**Why shallow processing?** Current large–scale information management systems IMS are concerned with the task of extracting relevant features from natural language (NL) text documents and searching for interesting relationships between the extracted entities (i.e., structured data objects), e.g., text mining, information extraction, semantics–oriented information retrieval or extraction of ontologies from NL texts, see also fig. 1. A challenging feature of such IMS is that the information is only implicitly encoded in an unstructured way from the perspective of a computational system. Thus, a major first step is to map the unstructured NL text to a structured internal representation (basically a set of data objects), which is then further processed by the application and domain–specific algorithms (e.g., in the case of text mining these are known data mining algorithms and in the case of information extraction they are domain–specific template filling and merging algorithms).

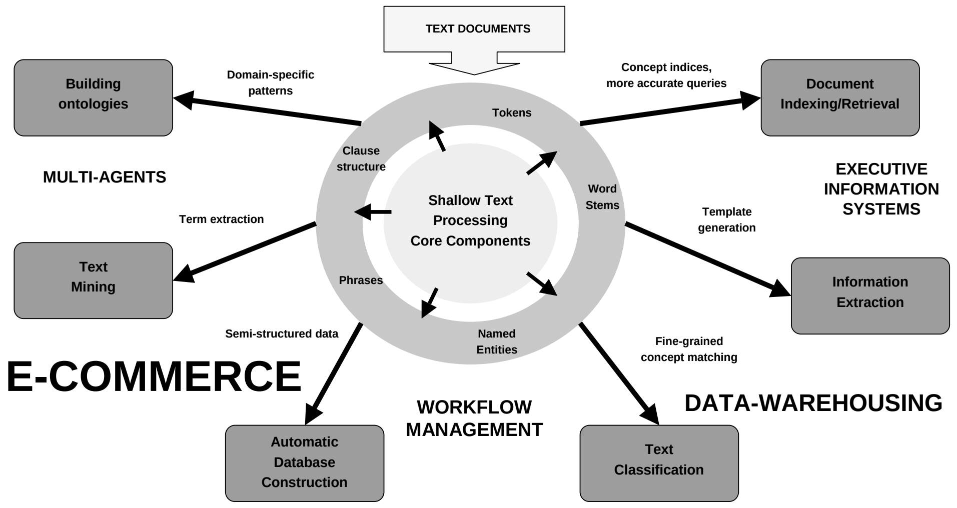


Figure 1: Applications of shallow text processing

It seems obvious that the more structure one can extract from the NL texts the better the application specific algorithms might perform. In principle, it would be possible to use an exhaustive and deep generic text understanding system, which would aim to accommodate the full complexities of a language and to make sense of the entire text. However, even if it were be possible to formalize and represent the complete lexicon and grammar of a natural language, the system would still need a very high degree of robustness and efficiency. It has been shown that, at least today, realizing such a system is impossible for large–scale NL processing. However, in order to fulfill the ever increasing demands for improved processing of real-world texts, NL researchers have started to relax the theoretical challenge to some more practical approaches which handle the requested robustness and efficiency. This has lead to so–called shallow NL processing approaches, where certain generic language regularities which are known to cause complexity problems are either not handled, e.g., instead of computing all possible readings only an underspecified structure is computed or handled very pragmatically, e.g., by restricting the depth of recursion on the basis of a corpus analysis or by making use of heuristic rules, like "longest matching substrings". This engineering view of language has lead to a renaissance and improvement of well-known efficient techniques, most notably finite state technology for robust parsing.

**NLP as normalization** We are interested in exploring and investigating large–scale re-usable and domain-adaptive language technology by viewing NLP as a step–by–step process of normalization from more general coarse-grained to more fine-grained information depending on the degree of structure and the naming (typing) of structural elements. For example, in the case of morphological processing, the determination of lexical stems (e.g., "Haus" (*house*)) can be seen as a normalization of the corresponding word forms (e.g., "Häusern" (*houses-PL-DAT*) and "Hauses" (*house-SG-GEN*). In the same way, named entity expressions or other special phrases (word groups) can be normalized to some canonical forms and treated as *paraphrases* of the underlying concept. For example, the two date expressions "18.12.98" and "Freitag, der achtzehnte Dezember 1998" could be normalized to the following structure:

$$\langle type = date, year = 1998, month = 12, day = 18, weekday = 5\rangle.$$

In the case of generic phrases or clause expressions, a dependence-based structure can be used for normalization. For example, the nominal phrase "für die Deutsche Wirtschaft" (*for the German economy*) can be represented as

$$\langle head = f\ddot{u}r, comp = \langle head = wirtschaft, quant = def, mod = deutsch\rangle\rangle.$$

One of the main advantages of following a dependence approach to syntactic representation is its use of syntactic relations to associate surface lexical items. Actually this property has lead to a recent renaissance of dependence approaches especially for its use in shallow text analysis (e.g., (Grinberg, Lafferty, and Sleato, 1995; Oflazer, 1999)). Following this view point, domain-specific IE templates also can be seen as normalizations because they only represent the relevant text fragments (or their normalizations) used to fill corresponding slots by skipping all other text expressions. Thus seen, two different text documents which yield the same template instance can be regarded as paraphrases because they "mean" the same.

**Robust parsing of unrestricted text** In this paper we will take this point of view as our main design criteria for the development of SPPC: a robust and efficient core engine for shallow text processing. SPPC consists of a set of advanced domain–independent shallow text processing tools which supports very flexible preprocessing of text wrt. the degree of depth of linguistic analysis. In contrast to the common approach of deep grammatical processing, where the goal is to find all possible readings of a syntactic expression, we provide a complete but underspecified representation by only computing a general coarse-grained syntactic structure which can be thought of as domain independent. This rough syntactic analysis can then be made more precise by taking into account domain-specific knowledge. Our parser recognizes basic syntactic units and grammatical relations (e.g., subject/object) robustly by using relatively underspecified feature structures, by postponing attachment decisions and by introducing a small number of heuristics.

SPPC is a very fast and robust, completed and functioning large–scale NLP system for German which possesses a high degree of modularity and domain–independence. Besides this important engineering strength of the system, the major scientific contribution is its novel two-phase robust parsing strategy. In contrast to standard bottom-up chunk parsing strategies, we present a divide-and-conquer strategy for robust parsing that only determines the topological structure of a sentence (i.e., verb groups, sub–clauses) in a first phase. In a second phase the phrasal grammars are applied to the contents of the different fields of the main and sub-clauses followed by a final step which determines the grammatical functions for the identified syntactic constituents. The whole approach proved to be very useful for processing free word order languages like German what concerns speed, robustness and coverage (f-measure of 87.14% of unseen data, see sec. 6). Although the basic underlying machinery of our robust parsing approach is based on state–of-the–art finite technology, the degree and richness of the syntactic structure (constituent structure, grammatical functions, agreement information) goes beyond most of the recent shallow parsing systems; especially for processing unrestricted German NL text, it seems to be the best.

SPPC has a high application potential and has already been used in different application areas ranging from processing email messages (Busemann et al., 1997), text classification

(Neumann and Schmeier, 1999), text routing in call centers, text data mining, extraction of business news information (these latter as part of industrial projects), to extraction of semantic nets on the basis of integrated domain ontologies (Staab et al., 1999). SPPC has been fully implemented for German with high coverage on the lexical and syntactic level, and with an excellent speed. We have also implemented first versions of SPPC for English and Japanese using the same core technology (see also sec. 8). In this paper we will, however, focus on processing German text documents.

The rest of the paper is organized as follows. In section 2 we give a complete overview of the whole system by describing briefly all relevant components using a running example to demonstrate some of the technical details. In the sections 3 to 5 we then describe in more detail the major novel aspects of our parsing approach. In section 3 we describe important aspects of our finite state technology, in section 4 we define and discuss a robust algorithm for performing online recognition of German compounds, and in section 5 the robust two–level parser is described. Evaluation results of major components are collected and summarized in section 6. In section 7, we relate our work basically to other methods which also treat unrestricted German texts, and conclude the paper in section 8 with a short description of some interesting future directions.

## 2 System overview

In this section we give a complete, but rough overview of the system. Details of major novel aspects of our approach will then be described in the following sections. The architecture of SPPC is shown in Figure 2. It consists of two major components, a) the Linguistic Knowledge Pool LKP and b) STP, the shallow text processor itself. The STP provides a partial analysis of NL-texts by exploiting the linguistic resources available in the LKP. Emphasis is placed on recognizing basic syntactic units without attempting to resolve attachment ambiguities or to recover missing information (such as traces resulting from the movement of constituents). The output of STP is a sequence of flat *underspecified (partial) dependency trees* (UDTs), where only upper bounds for attachment and the scoping of modifiers are expressed (see Section 5 for details). Besides the identification of

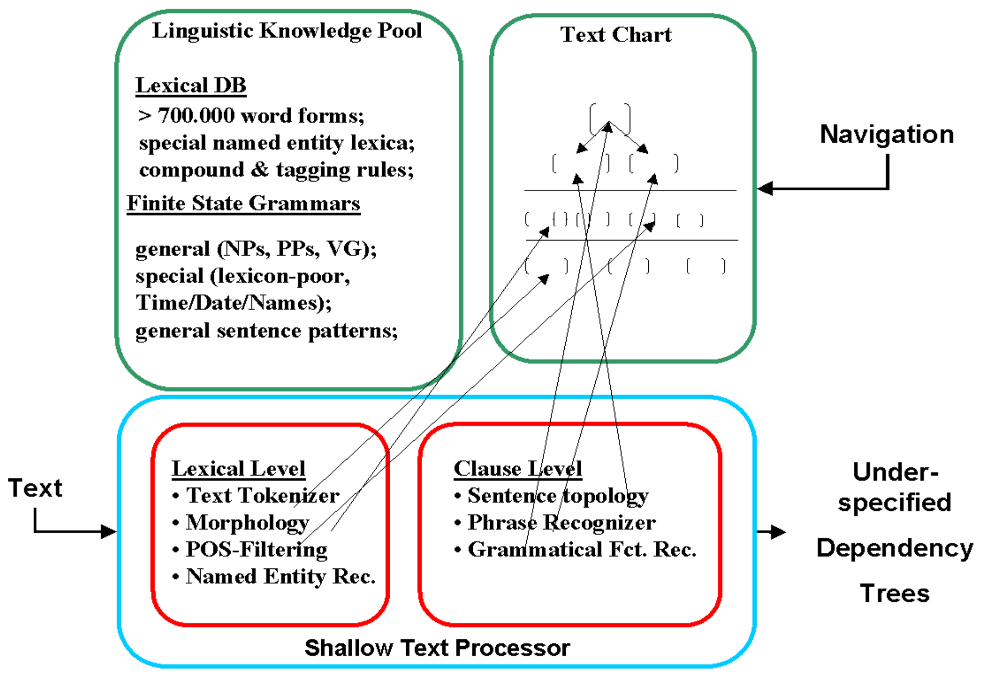


Figure 2: The blueprint of the system architecture.

shallow linguistic patterns (i.e., chunks) and the labeling of head-dependent relations, STP also assigns grammatical functions (e.g., subject, object) using a large subcategorisation lexicon.

We distinguish two primary levels of processing within in STP, the *lexical level* and the *clause level*. Both are subdivided into several components. In order to illustrate their functionality we will use the following sentence as a running example

Die Siemens GmbH hat 1988 einen Gewinn von 150 Millionen DM, weil die Aufträge im Vergleich zum Vorjahr um 13% gestiegen sind.

*Siemens Ltd made a profit of 150 million German Marks in 1988 due to a 13% increase in contracts in comparison to the previous year.*

## 2.1 Lexical Processing

**Tokenization** The first component on the lexical level is the TEXT TOKENIZER which maps sequences of characters into greater units, called tokens and identifies their type (e.g., lowercase words, abbreviations). We use a wide variety of about 50 token types which simplifies the processing of the subsequent modules. For instance, the token “GmbH” (*Ltd*) in our example would be classified as a *mixed word* since it consists of both lower and upper case letters, where such information determines a potential acronym.

**Morphological analysis** Each token which is identified as a potential word form is further processed by the MORPHOLOGICAL COMPONENT which includes inflectional analysis of word forms, and on-line recognition of compounds (e.g., “Kunststoffbranche” *synthetic materials industry*). The latter tasks is crucial when processing unrestricted NL texts since compounding is a very productive and creatively used property of the German language (see more details in section 4). Each recognized valid word form is associated with the list of its possible readings consisting of the stem, inflection information and the part-of-speech (POS–) category. For example, the token ”Gewinn” in our example sentence would be associated with the verb reading [POS: V FORM: IMP NUM: SG] (imperative singular of *to win*) and the noun reading [POS: N NUM: SG CASE: {Nom, Dat, Acc}] (*revenue* in singular nominative, dative or accusative). Our morphological component uses a full-form lexicon containing about 700,000 entries which were automatically created from 120,000 stem entries in the morphological component MORPHIX (Finkler and Neumann, 1988). We did not use MORPHIX in our system, simply because it is not available in C++ (the major programming language of the core system) with its full functionality. On the other hand, it is clear that making use of a full-form lexicon simplifies on-line morphological processing. Hence our combined approach—full-form lexicon with on-line compounding— also seems to be an interesting practical alternative.

**POS–tagging** Words which are ambiguous with respect to their POS–category are disambiguated using three types of manually constructed filtering rules: (a) case-sensitive rules, (b) contextual filtering rules based on POS-information (e.g., change tag of word from noun or verb to noun if the previous word is a determiner), and (c) rules for filtering out rare readings (e.g., “recht” - *right* vs. *rake*(3rd person, sg)). In order to achieve broader coverage we integrated (and manually checked) rules determined by Brill’s tagger

(Brill, 1993). The current version of the system contains about 120 filtering rules which are compiled into one single finite-state expression using the techniques described in section 3. As a simple illustrative example, consider again the word "Gewinn" (meaning either *to win* or *revenue*) where the POS–tagger would filter out the verb reading since verbs can only be used with an uppercase initial letter in the initial position of a sentence.

**Named–Entity recognition** In the final step of the lexical component, the Named Entity (NE) finder treats temporal expressions like time and date, several name expressions such as organizations, persons and location. Each type of NE is defined as a finite–state sub–grammar which takes into account the specific context an NE appears in (e.g., company designator, first name, morpho–syntactic properties of contextual elements). We are following a rule–based pattern–recognition approach similar to the approaches described in (Appelt et al., 1993; Grishman, 1995), such that NE's are identified basically by looking for contextual cues relying only on a small amount of NE–specific dictionaries (e.g., a list of the names of the 50 largest companies). The major reason for doing this is that the creation and use of NE's change dynamically over time, so that a pure dictionary approach is not realistic. We also decided to use a rule–based approach, because statistics based approaches (cf. (Borthwick, 1999) and (Bikel et al., 1997)) are too corpus and domain sensitive, and a rule–based approach is usually easier to maintain.

However, we observed that subsequent occurrences of already recognized NE's frequently appear in abbreviated form (e.g., "Siemens GmbH" and "Siemens"), often by making use only of a single word. Instead of defining specific recognition rules for these cases, we developed a method for the online creation of a dynamic NE lexicon. Once an NE has been recognized by means of the known rules (e.g., "Martin Marietta Corp."), we store all words (without the contextual cues) in a lexicon (e.g., separate, but connected entries for "Martin Marietta", "Martin" and "Marietta"). Then, for each unknown word sequence or common noun (e.g., the word "March" is usually used to refer to the month, but could also be part of an NE) which occurs in a certain distance to an recognized NE, we look it up in the dynamic lexicon. In this way, an NE–specific kind of co–reference resolution is performed. Continuing the annotation of our running example with extracted features, the NE–recognizer contributes as follows:

> Die [$_{company}$ Siemens GmbH] hat [$_{num}$ 1988] einen Gewinn von [$_{monetary}$ 150 Millionen DM], weil die Aufträge im Vergleich zum Vorjahr um [$_{percent}$ 13%] gestiegen sind.

Recognition of named entities could be postponed and integrated into the clausal level, but it seems to be more appropriate to perform it at this stage since it reduces the amount of potential sentence boundaries (many punctuation marks are included in named entities).

## 2.2 Clause Level

At the clause level, the hierarchical structure of the words of a sentence is constructed using our robust divide–and-conquer chunk parser. In section 5 the motivation as well as the technical details are discussed, so we will introduce it here only very briefly. In contrast to deep parsing strategies, where phrases and clauses as well as determination of their grammatical functions are interleaved, we separate these steps into three corresponding sub–components: 1) recognition of NPs, PPs, verb groups (VG), as well as named entity (NE) phrases; 2) recognition of topological clause structure; and 3) recognition of grammatical functions. Steps 1 and 2 are interleaved, whereas step 3 operates on the output of the topological structure recognition component. Steps 1 and 2 are realized by means of finite state grammars. Step 3 is realized through a specialized constraint solver which performs agreement checks between a verbal head and its dependents while taking into account the subcategorisation information which is available in the LKP (see Figure 2). The finite-state backbone of the whole system currently consists of 141 (mainly disjunctive) regular expressions plus a small number of lexical rules for passivization used in Step 3.

Again using our running example, in a first phase only the verb groups and the topological structure of a sentence according to the linguistic *field theory* (cf. (Engel, 1988)) are determined domain-independently. In our example, the recognition of verb groups (VG yields—note that we assume that NE–recognition as already taken place):

> Die [$_{company}$ Siemens GmbH] [$_{VG}$ hat] [$_{num}$ 1988] einen Gewinn von [$_{monetary}$ 150 Millionen DM], weil die Aufträge im Vergleich zum Vorjahr um [$_{percent}$

13%] [$_{VG}$ gestiegen sind].]]

and the determination of the topological structure results in the following bracketing sub–clause structures (where MAIN-CL and SUB-CL stand for main clause and sub clause respectively):

> [$_{MAIN-CL}$ Die [$_{company}$ Siemens GmbH] [$_{VG}$ hat] [$_{num}$ 1988] einen Gewinn von [$_{monetary}$ 150 Millionen DM], [$_{SUB-CL}$ weil die Aufträge im Vergleich zum Vorjahr um [$_{percent}$ 13%] [$_{VG}$ gestiegen sind].]]

In the next phase, general (as well as domain-specific) phrasal grammars (nominal and prepositional phrases) are applied to the contents of the different parts of the main and sub-clauses. The current result of the analysis of our example sentence is enriched with NP and PP bracketings (for nominal and prepositional constructs respectively):

> [$_{MAIN-CL}$ [$_{NP}$ Die [$_{company}$ Siemens GmbH]] [$_{VG}$ hat] [$_{num}$ 1988] [$_{NP}$ einen Gewinn] [$_{PP}$ von [$_{monetary}$ 150 Millionen DM]], [$_{SUB-CL}$ weil [$_{NP}$ die Aufträge] [$_{PP}$ im Vergleich] [$_{PP}$ zum Vorjahr] [$_{PP}$ um [$_{percent}$ 13%]] [$_{V}G$ gestiegen sind].]]

In the final step, the grammatical structure is computed using a large subcategorization lexicon for verb stems. It defines syntactic constraints for the arguments of the verb (usually nominal phrases) in order to assign grammatical functions to them once they have been identified (see details of this step in section 5). The final output of the parser for a sentence is then an underspecified dependence tree, where only upper bounds for attachment and scoping of modifiers are expressed (see Figure 3).

## 2.3 Information access

The system stores all partial results on each level of processing uniformly as feature value structures (together with their type and the corresponding start and end positions of the spanned input text) in a data structure called *text chart* (Piskorski and Neumann, 2000). The different kinds of index information computed by the individual components (e.g., text position, reduced word form, category, phrase) support an uniform flexible and efficient access to all extracted features. Therefore unnecessary re-computations can be avoided

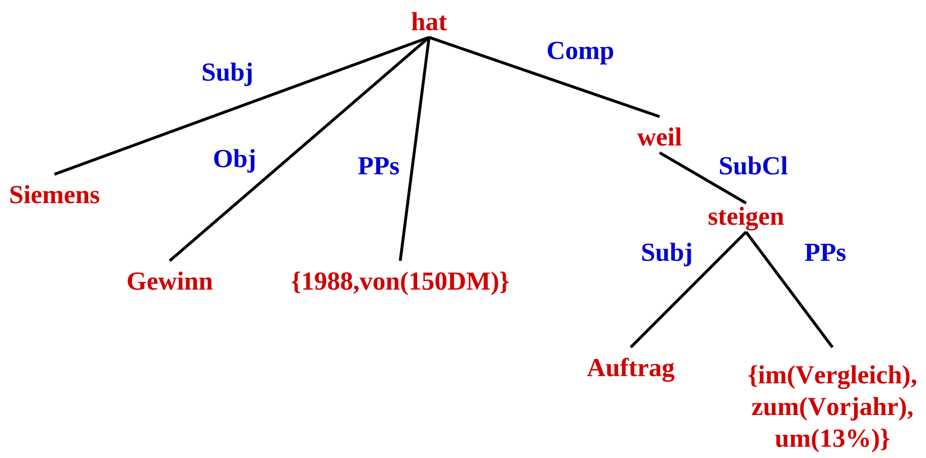


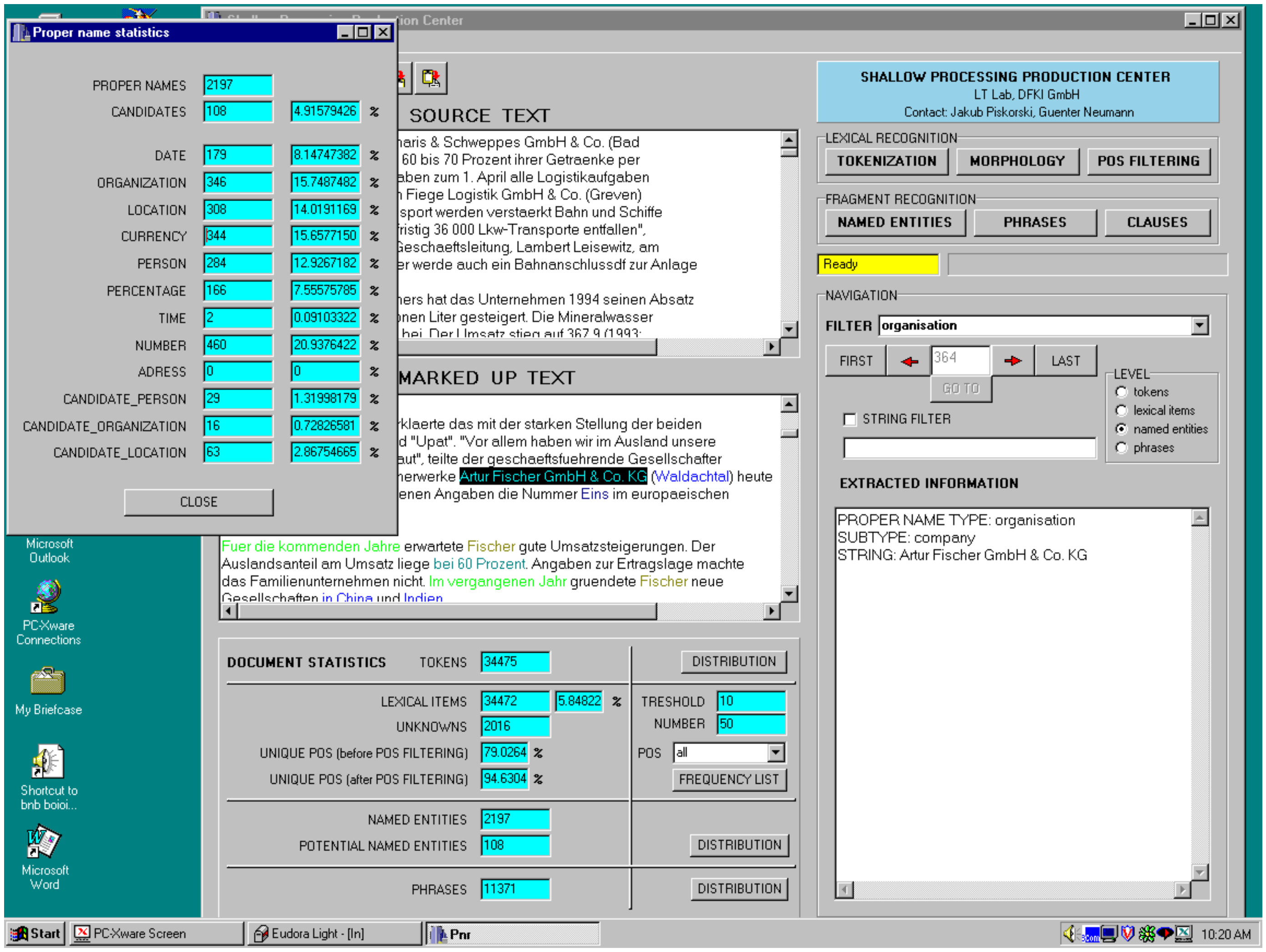
Figure 3: The underspecified dependence structure of the example sentence.

and rich contextual information in case of disambiguation or the handling of unknown constructions is provided. The system provides for a parameterizable XML interface, such that the user can select which sort of computed information should be considered for enriching the processed document with corresponding XML mark ups. In that way, the whole system can easily be configured for the demands of different applications. Figure 4 shows a screen dump of the current GUI of SPPC and demonstrates the navigation options.

### 2.4 Core Technology

The whole system has been realized on top of two major core technologies which are briefly described here (see section 3 for more details). For efficiency and expressivity reasons, one of the major design goals of SPPC is to model all levels of processing as finite-state (FS) devices[2]. Therefore, we developed a generic toolkit for building, combining and optimizing FS devices which provides all necessary functionalities relevant to the realization of the different processing levels (from tokenization to robust parsing) in a uniform way. Nevertheless, the toolkit contains all major state–of–the–art FS operations and is designed with the consideration of future enhancements and applications, as well as

[2]Computationally, FS devices are time and space efficient. From the linguistic point of view, local recognition patterns can be easily and intuitively expressed as FS devices



| | | | |
|---|---|---|---|
| PROPER NAMES | 2197 | | |
| CANDIDATES | 108 | 4.91579426 | % |
| DATE | 179 | 8.14747382 | % |
| ORGANIZATION | 346 | 15.7487482 | % |
| LOCATION | 308 | 14.0191169 | % |
| CURRENCY | 344 | 15.6577150 | % |
| PERSON | 284 | 12.9267182 | % |
| PERCENTAGE | 166 | 7.55575785 | % |
| TIME | 2 | 0.09103322 | % |
| NUMBER | 460 | 20.9376422 | % |
| ADRESS | 0 | 0 | % |
| CANDIDATE_PERSON | 29 | 1.31998179 | % |
| CANDIDATE_ORGANIZATION | 16 | 0.72826581 | % |
| CANDIDATE_LOCATION | 63 | 2.86754665 | % |

CLOSE

SHALLOW PROCESSING PRODUCTION CENTER
LT Lab, DFKI GmbH
Contact: Jakub Piskorski, Guenter Neumann

LEXICAL RECOGNITION: TOKENIZATION | MORPHOLOGY | POS FILTERING
FRAGMENT RECOGNITION: NAMED ENTITIES | PHRASES | CLAUSES

Ready

SOURCE TEXT

```
aris & Schweppes GmbH & Co. (Bad
 60 bis 70 Prozent ihrer Getraenke per
aben zum 1. April alle Logistikaufgaben
 Fiege Logistik GmbH & Co. (Greven)
sport werden verstaerkt Bahn und Schiffe
ristig 36 000 Lkw-Transporte entfallen",
Geschaeftsleitung, Lambert Leisewitz, am
er werde auch ein Bahnanschlussdf zur Anlage

hers hat das Unternehmen 1994 seinen Absatz
nen Liter gesteigert. Die Mineralwasser
 bei. Der Umsatz stieg auf 367,9 (1993:
```

MARKED UP TEXT

```
klaerte das mit der starken Stellung der beiden
d "Upat". "Vor allem haben wir im Ausland unsere
aut", teilte der geschaeftsfuehrende Gesellschafter
herwerke Artur Fischer GmbH & Co. KG (Waldachtal) heute
enen Angaben die Nummer Eins im europaeischen

Fuer die kommenden Jahre erwartete Fischer gute Umsatzsteigerungen. Der
Auslandsanteil am Umsatz liege bei 60 Prozent. Angaben zur Ertragslage machte
das Familienunternehmen nicht. Im vergangenen Jahr gruendete Fischer neue
Gesellschaften in China und Indien
```

NAVIGATION
FILTER: organisation
FIRST | 364 | LAST | GO TO
STRING FILTER
LEVEL: tokens, lexical items, [✓ named entities], phrases

EXTRACTED INFORMATION

```
PROPER NAME TYPE: organisation
SUBTYPE: company
STRING: Artur Fischer GmbH & Co. KG
```

DOCUMENT STATISTICS

| | | | |
|---|---|---|---|
| TOKENS | 34475 | | |
| LEXICAL ITEMS | 34472 | 5.84822 | % |
| UNKNOWNS | 2016 | | |
| UNIQUE POS (before POS FILTERING) | 79.0264 % | | |
| UNIQUE POS (after POS FILTERING) | 94.6304 % | | |
| NAMED ENTITIES | 2197 | | |
| POTENTIAL NAMED ENTITIES | 108 | | |
| PHRASES | 11371 | | |

DISTRIBUTION
TRESHOLD: 10
NUMBER: 50
POS: all
FREQUENCY LIST

Figure 4: A screen dump of the system in action. The user can choose which level to inspect, as well as the type of expressions (e.g., in case of named entities, she can select “all”, “organizaton”, “date” etc).

its use within other FS-based frameworks. Relevant details of our FS–toolkit are discussed in section 3.

Since in some cases FS devices are not an optimal storage device (e.g., maintenance of dynamic dictionaries), we have defined a generalized version of tries[3], which is a ubiquitous tree-based data structure in text processing (Cormen, Leiserson, and Rivest, 1992). We support storage of strings over an arbitrary alphabet, where each such string is associated

[3]A trie is a rooted tree with edges labeled by alphabet symbols, such that all outgoing edges of a node carry different labels. Furthermore, each node contains boolean value which indicates whether the path to the current node already constitutes a complete word.

with an arbitrary object. In addition to the standard time-efficient operations for insertion, searching, and deletion, we also included various operations for computing the longest and the shortest prefix/suffix of a given sequence in the trie, which are indispensable in the algorithm for compound decomposition (see section 4). Tries are also especially useful for implementing self organizing lexica which we we employ for the realization of the context-sensitive dynamic lexicon used in the process of the named-entity recognition mentioned earlier. We extended the standard trie so that all nodes representing complete words are connected, which allows efficient computation of statistical information to be performed over the set of extracted information.

## 3 Finite-State Technology

In order to cover all STP-relevant types of FS devices and to allow for a parametrizable weight interpretation we used the finite-state machine (FSM) as an underlying model for our toolkit. An FSM is a generalization of the more familiar finite-state automaton (FSA), finite-state transducer (FST) and their weighted counterparts (WFSA,WFST) (Mohri, 1997). FST's are automata for which each transition has an output label in addition to the input label. For instance, the FST in Figure 5 represents a contextual rule for part-of-speech disambiguation. Weighted FS devices allow for assigning weights to their transitions and states. In contrary to WFST's which are tailored to a specific semiring for weight interpretation (Mohri, 1997), the FSM's are more general in that they admit the use of arbitrary semirings.

More formally, we define a finite-state machine $M$ as a 9-tupple $M = (\Sigma_i, \Sigma_o, Q, i, c_i, F, C, E, (S, \bigoplus, \bigotimes, \overline{0}, \overline{1}))$, where: $\Sigma_i$ and $\Sigma_o$ are input and output alphabets, $Q$ is a finite set of states, $i$ is the initial state, $c_i$ is the initial weight, $F$ is the set of final states, $C : F \mapsto S$ is the final weight function, $E \subset Q \times (\Sigma_i \cup \epsilon) \times (\Sigma_o \cup \epsilon) \times S \times Q$ is the set of transitions and $(S, \bigoplus, \bigotimes, \overline{0}, \overline{1})$ is a semiring (Cormen, Leiserson, and Rivest, 1992), where $\overline{0}$ and $\overline{1}$ are the neutral elements of the summary operator $\bigoplus$ and the extension operator $\bigotimes$. The semiring determines the interpretation of weights, i.e. the weight of a path is computed by combining the weights of all arcs on this path using the extension

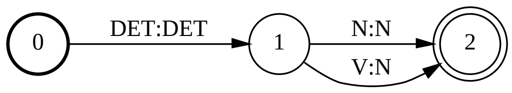


Figure 5: A simple FST representing a contextual POS-disambiguation rule: change tag of word form noun or verb to noun if the previous word is a determiner

operator, whereas the summary operator is used for combining the weights of all accepting paths for a given input string which yield the same output. Since different accepting paths might potentially produce different output, an output of an FSM applied to a given input string is defined as a set of pairs, each consisting of an output string and an associated weight computed as described above. Furthermore, only single alphabet symbols may be used as transition labels, since most FS operations require this feature and time consuming conversions may be avoided (see (Piskorski, 1999) for more formal details).

The architecture and functionality of our FSM Toolkit is mainly based on the tools developed by AT&T (Mohri, Pereira, and Riley, 1996). The operations provided are divided into: (a) rational and combination operations (e.g., composition, intersection), (b) equivalence transformations (e.g., determinization, minimization) and (c) converting operations (e.g., creating graphical representations). The realization of most of them is based on the recent approaches proposed in (Mohri, 1997), (Mohri, Pereira, and Riley, 1996), (Roche and Schabes, 1995), (Roche and Schabes, 1996) and they work with arbitrary real-valued semirings (only a computational representation of the semiring is needed). We used the *tropical semiring* $(R \cup \infty, min, +, \infty, 0)$ for FS-pattern prioritization, whereas the real semiring $(R, +, \cdot, 0, 1)$ is appropriate when dealing with FS probabilistic grammars. [4] For instance, the patterns for named-entity recognition are represented as WFSA's (see figure 6), where the weights indicate their priorities. These WFSA's are merged into an optimized (deterministic and minimal) single WFSA representing all NE-recognition patterns. Through the choice of using the *tropical semiring* all potential ambiguities can be

[4]If the weights represent probabilities, the weight assigned to a path should be the product of the weights of its transitions, while the weight assigned to a set of paths with a common source and target should be the sum of all path weights in the set.

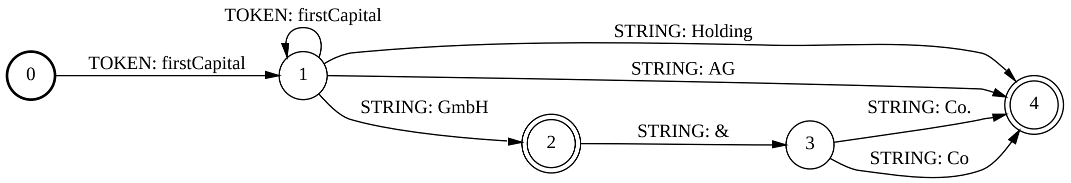


Figure 6: The automaton representing the pattern (here a simplified version) for recognition of company names.

resolved.

In contrast to the AT&T tools, we provide some new operations relevant to STP. For instance, the algorithm for local extension, which is crucial for merging part-of-speech filtering rules into a single FST has been realized and adapted for the case of WFST's. Local extension (Roche and Schabes, 1995) of an FSM which transforms $a$ into $b$ is an FSM that transforms string $u$ into $v$, where $v$ is built from the string $u$ by replacing each occurrence of $a$ by $b$ in such a way that each possible factorization of the input string $u$ is considered. The overall cost is computed by combining the weights of all transformations of $a$ into $b$ with the extension operator. More formally, a local extension of an FSM $M$ is an FSM $M_{locext}$, which for all $u \in \Sigma_i^*$ with $u = x_1 y_1 x_2 .....x_n y_n x_{n+1}$ and $\forall k \in 1, ..., n$: $y_k \in L(M)$ ($M$ accepts $y_k$), $M$ transforms $y_k$ into $\overline{y_k}$ with cost $c_k$ and $\forall l \in 1, ...., n+1$: $x_l \in \Sigma^* - (\Sigma^* \circ (L(M)) \circ \Sigma^*)$, transduces $u$ into $v$ where $v = x_1 \circ \overline{y_1} \circ x_2 \circ .... \circ \overline{y_n} \circ x_{n+1}$ with cost $c_1 \bigotimes c_2 \bigotimes .... \bigotimes c_n$. The merging of all POS-filtering rules is then done by computing the local extension of each rule represented as FST and combining the resulting FST's into a single optimized FST by applying composition, determinization and minimization.

We also improved the general algorithm for removing $\epsilon$-moves (Mohri, Pereira, and Riley, 1996), which is an essential operation in the process of determinization. We sketch here briefly the major modifications. The standard algorithm is divided into two phases. In the first phase, the input FSM $M$ is subdivided into $M_\epsilon$ containing only $\epsilon$-moves and

$\overline{M_\epsilon}$ containing all other arcs. Subsequently, $\widehat{M_\epsilon}$ representing the transitive closure of $M_\epsilon$ is computed. Finally, the new equivalent $\epsilon$-free FSM is constructed by iterating over the set of transitions of $\widehat{M_\epsilon}$ and modification of existing weights or introduction of new edges in $\overline{M_\epsilon}$. Since the computation of the transitive closure in the general case of arbitrary semirings has the complexity of $O(n^3)$ assuming that the computation of $\bigotimes$ and $\bigoplus$ can be performed in $O(1)$ (Cormen, Leiserson, and Rivest, 1992), we implemented some modifications.

Firstly, in the preprocessing step all of the *simple* $\epsilon$-moves are removed from the input FSM, where an $\epsilon$-move is considered as *simple* when its target state does not have any outgoing $\epsilon$ arcs. Removing such transitions introduces new transitions to the input FSM or minor weight modifications, and results in the appearance of new *simple* $\epsilon$-moves. Therefore, this process is repeated until no more *simple* $\epsilon$-moves exist[5]. Analogously to the standard algorithm, the resulting FSM is then subdivided into $M_\epsilon$ and $\overline{M_\epsilon}$. In the next step, the transitive closure of each connected component in $M_\epsilon$ is computed since one could expect them to be small in relation to the overall size of $M_\epsilon$. The remaining procedure is identical to that of the standard algorithm. Despite the fact that the modifications described here impair the worst-case complexity, they proved to speed up the removal of $\epsilon$-moves considerably in the process of optimizing the FS grammars used in SPPC. As a matter of fact, the second phase of the algorithm (computing the transitive closure etc.) turned out to be superfluous since there were no remaining $\epsilon$-moves.

The FSM Toolkit is divided in two levels: a user–program level consisting of a standalone application for manipulating FSM's by reading and writing to files and a C++-library level which implements the user-program level operations and allows for easy embedding of the toolkit into other applications.

## 4 Online compound analysis

In this section we describe the basic algorithm for performing compound recognition in German, because we feel that our approach is of particular interest in the context of

[5]The technique described here is more of a guideline, since depending on input data one could define *simple* $\epsilon$-moves differently and use various methods for removing them.

large–scale robust NL processing.[6] Unlike in English, German compounds[7] are in general orthographically single words (e.g., “Computerspiel” - computer game) and they are usually not lexicalized. Therefore, every token not recognized as a valid word form in a lexicon is a compound candidate and since nouns are written with a capitalized initial letter in standard German it is not a straightforward decision to exclude such words from being a compound (noun compounds are most frequent). Furthermore, German compounds frequently include so called linking morphemes (e.g., “s” in “Forschungsausgaben” - “Forschung” + “s” + “ausgaben” (*research expenses*). The syntactic head of a German compound is the rightmost constituent and all other constituents function as modifiers of the head.

The syntactic structure of a compound may be complex and ambiguous. For instance, the structure of the compound “Biergartenfest” (*beer garden party*) could be [beer [garden party]] (*garden party with beer*) or [[beer garden] party] (*party in the beer-pub*). In addition, more than one valid syntactic segmentation for a given compound might exist (e.g., “Weinsorten” could be decomposed into “Wein” + “sorten” (*wine types*) or “Wein” + “s” + “orten” (*wine places*). Semantically correct segmentation and computation of the internal structure of a German compound might require a great deal of knowledge, but since computing such complete information might be unnecessary for performing more low–level tasks (e.g., part-of-speech filtering or phrase recognition), we focus here on *shallow compound analysis* and present an algorithm which computes a single syntactically valid segmentation of a compound and determines its head while leaving internal bracketing unspecified.

The basic idea of the algorithm is to use a full-form lexicon to find the longest suffix and prefix of a compound candidate which are valid word forms and may function as compound morphemes, and to finally try to segment the remaining string by consecutively finding longest prefixes. Since the choice of the longest suffix and the longest prefix

[6]Actually we are not aware of any published technical details concerning large-scale German compound recognition.

[7]A compound is a consecutive sequence of at least two morphemes which functions as a valid word form.

might sometimes not be correct (e.g., “Autoradiozubehör” *car radio equipment*, can not be properly decomposed if we choose the longest prefix “Autor” (*author*) and the longest suffix “zubehör” (*equipment*) since the remaining part “adio” can not be further decomposed and is not a valid word form either), the algorithm iterates over all combinations of suffixes and prefixes starting with the longest ones.

```
(1) procedure $find\_segmentation(STRING w_1w_2.....w_n)$
(2) LIST $Infixes = \emptyset$
(3) $s \leftarrow longest\_valid\_suffix(w_1w_2.....w_n)$
(4) while($s \neq \epsilon$) do
(5)         $p \leftarrow longest\_valid\_prefix(w_1w_2.....w_{n-|s|})$
(6)         while($p \neq \epsilon$) do
(7)                 $r \leftarrow w_{|p|+1}w_2.....w_{n-|s|}$
(8)                 while($r \neq \epsilon$) do
(9)                         $t \leftarrow longest\_valid\_infix(r)$
(10)                        if($t = \epsilon$) $t = longest\_valid\_linking\_morpheme(r)$
(11)                        if($t \neq \epsilon$)
(12)                            $Indexes.add(t)$
(13)                            $r \leftarrow r_{|t|+1}.....r_{|r|}$
(14)                        else break;
(15)                if($r = \epsilon$) return $p + Infixes + s$
(16)                $p \leftarrow longest\_valid\_prefix(p_1p_2.....p_{|p|-1})$
(17)        $s \leftarrow longest\_valid\_suffix(s_2p_3.....s_{|s|})$
(18) return $\emptyset$
```

Since there are some compound morphemes (prefix or infix) which are not valid word forms we use a special list of morphemes which may function as compound prefix or infix (e.g., “multi”, “mega”, “macro”, “top” and some verb stems). In order to improve precision, we introduced simple constraints for validating compound morphemes (separately for prefixes, suffixes and infixes) which proved to pay off. For instance, coordinations are disallowed to be compound morphemes, whereas only verbs in imperative singular form are accepted as compound prefixes.

The decomposition algorithm is presented on page 20 in pseudo–code. The function $longest\_valid\_suffix$ returns the longest suffix of a given string which is a valid word form (or is included in the special morpheme list) and may function as a valid compound suffix. Analogously, the functions $longest\_valid\_prefix$ and $longest\_valid\_infix$ return the longest prefix of a given string which may function as a valid compound prefix or as a compound infix, respectively. After determination of the suffix and prefix, the algorithm tries to segment the remaining word sequence in the *while*-loop in line 8. If at some stage no appropriate prefix can be found we then look (line 10) for a valid linking morpheme (note that since we use a full-form lexicon the only morpheme which has to be considered is "s"). After successfully identifying all compound morphemes (line 15) we perform an additional handling of suffixes (not illustrated in the pseudocode) beginning with "s" since they introduce some ambiguities (e.g., "Wertschöpfungsteil" - *added value part* could be split into "Wert" + "schöpfung" + "steil" or "Wert" + "schöpfung" + "s" + "teil").[8]

The algorithm described in this section achieves surprisingly high recall and precision (see section 6).[9]

## 5 Clause level processing

In this section we describe the robust parsing strategy with a focus on the computation of the topological structure of German sentences and grammatical function recognition (for on overview of the whole parsing strategy, see section 2.2). Before discussing these details, we firstly motivate the approach.

**Problems with standard chunk parsers** Most of the well-known shallow text processing systems (cf. (Sundheim, 1995) and (SAIC, 1998)) use cascaded chunk parsers

[8]Note that it is relatively easy to extend this algorithm to compute all syntactically valid segmentations

[9]The algorithm is also used as a subroutine for resolving *coordinated compounds* like for instance, "Leder-, Glas-, Holz- und Kunststoffbranche" (*leather, glass, wood, plastic, and synthetic materials industry*) or 'An- und Verkauf" (*purchase and sale*). However, we will not discuss it here because of lack of space.

which perform clause recognition after fragment recognition following a bottom-up style as described in (Abney, 1996). We have also developed a similar bottom-up strategy for the processing of German texts, cf. (Neumann et al., 1997). However, the main problem we experienced using the bottom-up strategy was insufficient robustness: because the parser depends on the lower phrasal recognizers, its performance is heavily influenced by their respective performance. As a consequence, the parser frequently wasn't able to process structurally simple sentences, because they contained, for example, highly complex nominal phrases, as in the following example:

> "[$_{NP}$Die vom Bundesgerichtshof und den Wettbewerbshütern als Verstoß gegen das Kartellverbot gegeißelte zentrale TV-Vermarktung] ist gängige Praxis."
>
> *Central television marketing, censured by the German Federal High Court and the guards against unfair competition as an infringement of anti-cartel legislation, is common practice.*

During free text processing it might not be possible (or even desirable) to recognize such a phrase completely. However, if we assume that domain-specific templates are associated with certain verbs or verb groups which trigger template filling, then it will be very difficult to find the appropriate fillers without knowing the correct clause structure. Furthermore, in a sole bottom-up approach, some ambiguities – for example relative pronouns – can't be resolved without introducing much underspecification into the intermediate structures.

Therefore we propose the following *divide-and-conquer* parsing strategy: In a first phase only the verb groups and the topological structure of a sentence are determined domain-independently following roughly the theory of *topological fields* (Engel, 1988) (see Figure 7). In a second phase, general (as well as domain-specific) phrasal grammars (nominal and prepositional phrases) are applied to the contents of the different fields of the main and sub-clauses.

We call our parsing strategy *divide-and-conquer*, because we first identify a coarse–grained, top–down sub–clause bracketing for a sentence (divide), and then apply the phrasal grammars on each string of the identified sub–clauses independently (conquer), which realizes a bottom–up step. This is in contrast to the standard bottom–up chunk

"[$_{CoordS}$ [$_{core}$ Diese Angaben konnte der Bundesgrenzschutz aber nicht bestätigen], [$_{core}$ Kinkel sprach von Horrorzahlen, [$_{relcl}$ denen er keinen Glauben schenke]]]."

*[[This information could not be verified by the Border Police] [Kinkel spoke of horrific figures [which he did not believe.]]]*

Figure 7: An example of a topological structure. It consists of two core sub–clauses (where the second one has an embedded relative clause) which are combined in a simple coordinated structure. Note that the comma is obligatory in German, and hence can be used as a reliable cue for identifying possible sub-clauses.

parsers which would first compute all phrases before combining them to sub–clauses. The whole approach seems to be very useful for robust processing semi-free word order languages like German, in which there is, in principle, no strict order for the various phrases of a sentence, e.g., in German the two sentences "The student buys the book for his friend on the campus" and "For his friend the student buys on the campus the book" are well–formed (assuming for the moment, that German syntax would apply to English sentences). This free word order is a real challenge even in the case of chunk parsing, because the order of the phrases cannot be taken into account (in contrast to Languages like English, which have a relative fixed word order) when identifying grammatical functions (which is necessary for identifying possible slot fillers in case of information extraction, for instance). Our *divide-and-conquer* approach offers several advantages:

- improved robustness, because parsing of the sentence topology is based only on simple indicators like verb groups and conjunctions and their interplay,
- the resolution of some ambiguities, including relative pronouns vs. determiner (e.g., the German word "der" (*the*) can be used for both), and sentence coordination vs. NP coordination, and
- a high degree of modularity (easy integration of domain-dependent sub–components).

Furthermore, the recognition of the topological structure (at least the way we do it) is domain independent. It identifies an additional layer of linguistically oriented structure

```
Weil die Siemens GmbH,die vom Export lebt, Verluste erlitt, musste sie Aktien verkaufen.
Because the Siemens GmbH which strongly depends on exports suffered from losses
they had to sell some of the shares.
⇓
Weil die Siemens GmbH, die ...[Verb-Fin], V. [Verb-Fin], [Modv-Fin] sie A. [FV-Inf].
⇓
Weil die Siemens GmbH [Rel-Cl], V. [Verb-Fin], [Modv-Fin] sie A. [FV-Inf].
⇓
[Subconj-CL], [Modv-Fin] sie A. [FV-Inf].
⇓
[Subconj-CL], [Modv-Fin] sie A. [FV-Inf].
⇓
[clause]
```

Figure 8: The different steps of the DC-PARSER.

without using domain–specific information (in a similar way as a POS–tagger or morphological component can be used domain–independently), and hence can be used as a generic device for unrestricted NL text processing.

## 5.1 Topological structure

The identification of the topological structure is focused around a particular property of verbs in a German sentence: Based on the fact that in German a verb group (like "hätte überredet werden müssen" — **have persuaded been should* meaning *should have been persuaded*) can be split into a left and a right verb part ("hätte" and "überredet werden müssen'") these parts (abbreviated as LVP and RVP) are used for the segmentation of a main sentence into several parts: the front field (FF), the left verb part, middle field (MF), right verb part, and rest field (RF). For example, in a sentence like "Er hätte gestern überredet werden müssen" (*He should have been persuaded yesterday.*), the verb group (once identified) splits the sentence as follows:

| FF | LVP | MF | RVP | RF |
|---|---|---|---|---|
| Er | hätte | gestern | überredet werden müssen | EMPTY |

Sub–clauses can also be expressed in such a way that the left verb part is either empty or occupied by a relative pronoun or a subjunction element (e.g., *because*, *since*), and the complete verb group is placed in the right verb part. Note that each separated field can be arbitrarily complex with very few restrictions on the ordering of the phrases inside a field. For example, the topological structure of the embedded sub–clause of the sentence "Der Mann, der gestern hätte überredet werden müssen, lief nach Hause." (*The man, who should have been persuaded yesterday, ran home*) is:

| FF | LVP | MF | RVP | RF |
|---|---|---|---|---|
| EMPTY | der | gestern | hätte überredet werden müssen | EMPTY |

Recognition of the topological structure of a sentence can be described in four steps, each realized by means of a finite state grammar (see also Figure 2; Figure 8 shows the different steps in action). In each case, the input string is rewritten with the identified elements and passed as input to the next step. The schematic structure of the algorithm is as follows (initially, the stream of tokens and named entities is separated into a list of sentences based on punctuation signs). For each sentence do:

1. identify verb group using verb group grammar
2. identify base clauses using base clause grammar
3. combine subsequent base clauses to form larger units;
   if no larger unit was identified go to step 4 else go to step 2
4. identify main clauses using main clause grammar

**Verb groups** A verb grammar recognizes all single occurrences of verb forms (in most cases corresponding to LVP) and all closed verbgroups (i.e., sequences of verb forms, corresponding to RVP). The major problem at this phase is not a structural one but the massive morphosyntactic ambiguity of German verbs (for example, most plural verb forms can also be non-finite or imperative forms). This kind of ambiguity cannot be resolved without taking into account a wider context. Therefore these verb forms are assigned disjunctive types, similar to the underspecified chunk categories proposed by (Federici,

| | |
|---|---|
| Type | `VG-final` |
| Subtype | `Mod-Perf-Ak` |
| Modal-stem | könn |
| Stem | lob |
| Form | nicht gelobt haben kann |
| Neg | T |
| Agr | . . . |

Figure 9: The structure of the verb fragment "nicht gelobt haben kann" – **not praised have could-been* meaning *could not have been praised*. It actually says that this verb group has been identified in the final position of a clause, and that it basically describes a negated modality of the main verb "to praise".

Monyemagni, and Pirrelli, 1996). These verbal types, like for example different forms of finite participle (*has connected* versus the finite verb form *connected*), reflect the different readings of the verb form and enable following modules to use these verb forms according to the wider context, thereby removing the ambiguity. For example, in German it would be possible to utter something like "He has the edges connected." (proper English would be *He has connected the edges*). Thus the right verb part *connected*—viewed in isolaton—is ambiguous wrt. its use as a finite or participle verb form. In addition to a type, each recognized verb form, is assigned a set of features which represent various properties of the form like tense and mode information. (cf. Figure 9).

**Base clauses (BC)** are subjunctive and subordinate sub–clauses. Although they are embedded into a larger structure, they can be recognized independently and simply on the basis of commas, initial elements (like complementizer, interrogative or relative item – see also Figure 8, where SUBCONJ-CL and REL-CL are tags for sub–clauses) and verb fragments. The different types of sub–clauses are described very compactly as finite state expressions. Figure 10 shows a (simplified) BC-structure in feature matrix notation.

**Clause combination** It is very often the case that base clauses are recursively embedded as in the following example:

> . . . weil der Hund den Braten gefressen hatte, den die Frau, nachdem sie ihn zubereitet hatte, auf die Fensterbank gestellt hatte.

$$\left[\begin{array}{ll} \text{Type} & \textit{Subj-Cl} \\ \text{Subj} & \text{wenn} \\ \text{Cont} & \left[\begin{array}{ll} \text{Type} & \textit{Spannsatz} \\ \text{Verb} & \left[\begin{array}{ll}\text{Type} & \textit{Verb}\\ \text{Form} & \text{stellten}\\ \ldots & \end{array}\right] \\ \text{MF} & (\text{die Arbeitgeber Forderungen}) \\ \text{NF} & \left(\left[\begin{array}{ll} \text{Type} & \textit{Inf-Cl} \\ \text{Subj} & \text{ohne} \\ \text{Cont} & \left[\begin{array}{ll} \text{Type} & \textit{Simple-Inf} \\ \text{Verb} & \left[\begin{array}{ll}\text{Type} & \textit{Verb}\\ \text{Form} & \text{zu schaffen}\\ \ldots & \end{array}\right] \\ \text{MF} & \begin{array}{l}(\text{als Gegenleistung}\\ \text{neue Stellen})\end{array}\end{array}\right]\end{array}\right]\right) \end{array}\right]\end{array}\right]$$

Figure 10: Simplified feature matrix of the base clause "…, wenn die Arbeitgeber Forderungen stellten, ohne als Gegenleistung neue Stellen zu schaffen."
*… if the employers made new demands without creating new jobs in return.*

*Because the dog ate the roast which the woman put on the windowsill after preparing it.*

Two sorts of recursion can be distinguished: 1) *middle field* (MF) recursion, where the embedded base clause is framed by the left and right verb parts of the embedding sentence, and 2) the *rest field* (RF) recursion, where the embedded clause follows the right verb part of the embedding sentence. In order to express and handle this sort of recursion using a finite state approach, both recursions are treated as iterations so that they destructively substitute recognized embedded base clauses with their type. Hence, the complexity of the recognized structure of the sentence is reduced successively.

However, because sub–clauses of MF-recursion may have their own embedded RF-recursion, the CLAUSE COMBINATION (CC) is used for bundling subsequent base clauses before they are combined with sub–clauses identified by the outer MF-recursion. The BC and CC module are called until no more base clauses can be reduced (see figure 11). If the CC module were not to be used, then the following incorrect segmentation could not be avoided:

… *[$_{Rel-Cl}$ daß das Glück [$_{Subj-Cl}$, das Jochen Kroehne empfunden haben sollte] [$_{Subj-Cl}$, als ihm jüngst sein Großaktionär die Übertragungsrechte bescherte], nicht mehr so recht erwärmt.]

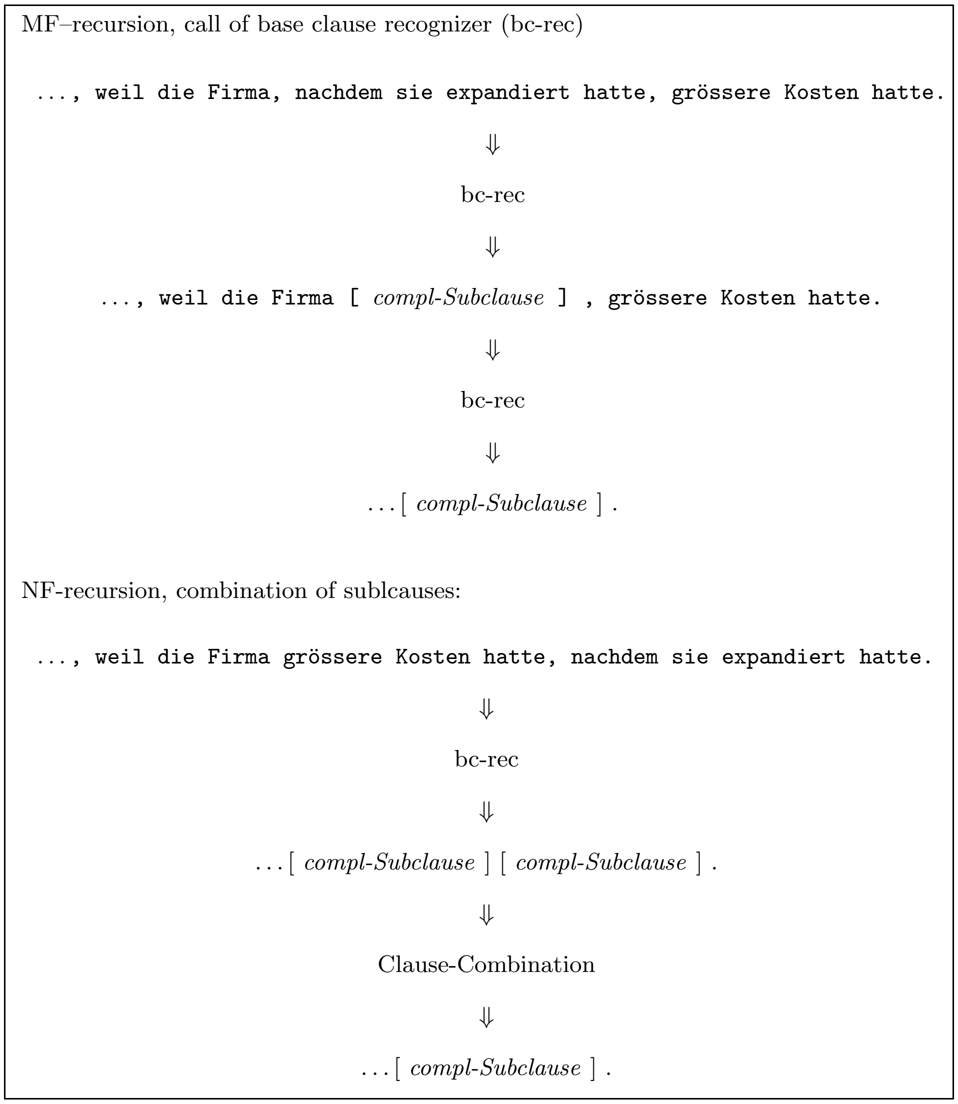

MF–recursion, call of base clause recognizer (bc-rec)
..., weil die Firma, nachdem sie expandiert hatte, grössere Kosten hatte.
⇓
bc-rec
⇓
..., weil die Firma [ *compl-Subclause* ] , grössere Kosten hatte.
⇓
bc-rec
⇓
... [ *compl-Subclause* ] .
NF-recursion, combination of sublcauses:
..., weil die Firma grössere Kosten hatte, nachdem sie expandiert hatte.
⇓
bc-rec
⇓
... [ *compl-Subclause* ] [ *compl-Subclause* ] .
⇓
Clause-Combination
⇓
... [ *compl-Subclause* ] .

Figure 11: The different treatment of MF and RF recursion for two sentences "‚ weil die Firma, nachdem sie expandiert hatte, [grössere Kosten hatte]." and "‚ weil die Firma [grössere Kosten hatte], nachdem sie expandiert hatte. "(both mean *, because after expanding, the company had increased costs*).

> *... that the happiness which Jochen Krhne should have felt when his major shareholder gave him the transfer rights recently is not really pleasing anymore.*

In the correct reading the second sub–clause "... als ihm jüngst sein ..." is embedded into the first one "... das Jochen Kroehne ...".

**Main clauses (MC)** Finally the MC module builds the complete topological structure of the input sentence on the basis of the recognized (remaining) verb groups and base clauses, as well as on the word form information not yet consumed. The latter basically includes punctuation and coordination. The following Figure schematically describes the current coverage of the implemented MC-module (see Figure 7 for an example structure):

| | | |
|---|---|---|
| *CSent* | ::= | . . . LVP . . . [RVP] . . . |
| *SSent* | ::= | LVP . . . [RVP] . . . |
| *CoordS* | ::= | CSent ( **,** CSent)$^*$ Coord CSent \| |
| | ::= | CSent (**,** SSent)$^*$ Coord SSent |
| *AsyndSent* | ::= | CSent **,** CSent |
| *CmpCSent* | ::= | CSent **,** SSent \| CSent , CSent |
| *AsyndCond* | ::= | SSent **,** SSent |

## 5.2 Grammatical function recognition

After the phrasal recognizer has expanded the corresponding phrasal strings (see the running example in section 2.2 on page 11), a further analysis step is done by the *grammatical function recognizer* (GFR), which identifies possible *arguments* on the basis of the lexical subcategorization information available for the local head. The final output of the clause level for a sentence is thus an underspecified dependence tree UDT. A UDT is a flat dependence-based structure of a sentence, where only upper bounds for attachment and scoping of modifiers are expressed (see Figure 3, page 13). In this example the PP's of each main or sub-clause are collected into one set. This means that although the exact attachment point of each individual PP is not known it is guaranteed that a PP can only be attached to phrases which are dominated by the main verb of the sentence (which is the root node of the clause's tree). However, the exact point of attachment is a matter of domain-specific knowledge and hence should be defined as part of the domain knowledge of an application. This is in contrast to the common approach of deep grammatical processing, where the goal is to find all possible readings of an expression wrt. all possible worlds. By just enumerating all possible readings such an approach is, to a certain extent, domain-independent. The task of domain-specificity is then reduced to the task of "selecting the right reading" of the current specific domain. In our approach, we provide a complete but underspecified representation by only computing a coarse-grained structure. This structure then has to be "unfolded" by the current application. In a way, this means that after shallow processing we only obtain a very general, rough meaning of an expression whose actual interpretation has to be "computed" (not selected) in the current application. This is what we mean by *underspecified text processing* (for further and alternative aspects of underspecified representations see e.g., (Gardent and Webber, 1998), (Muskens and Krahmer, 1998)).

A UDT can be partial in the sense that some phrasal chunks of the sentence in question could not be inserted into the head/modifier relationship. In that case, a UDT will represent the *longest matching sub–clause* together with a list of the non-recognized fragments. Retaining the non-recognized fragments is important, because it makes it possible that some domain specific inference rules have access to this information, even if it could not be linguistically analyzed.

**The subcategorization lexicon** The GFR exploits a subcategorisation lexicon for the identification of grammatical relations. The lexicon contains 11,998 verbs and a total of 30,042 subcategorisation frames (Buchholz, 1996). It also provides information about verbal arity, case (for NP complements), and the various types of sentential complements a given verb might take.

In general, a verb has several different subcategorization frames. As an example, consider the different frames associated to the main verb entry *fahr* ("to drive"):

*fahr*: $\{\langle np, nom\rangle\}$
$\{\langle np, nom\rangle,\ \langle pp, dat, mit\rangle\}$
$\{\langle np, nom\rangle,\ \langle np, acc\rangle\}$

Here, it is specified that *fahr* has three different subcategorization frames. For each frame, the number of subcategorized elements is given (through enumeration) and for each subcategorized element the phrasal type and its case information is given. In case of prepositional elements the preposition is also specified. Thus, a frame like $\{\langle np, nom\rangle,$ $\langle pp, dat, mit\rangle\}$ says that *fahr* subcategorizes for two elements, where one is a nominative NP and the other is a dative PP with preposition *mit* ("with"). There is no ordering presupposed for the elements, i.e., frames are handled as sets. The main reason is that German is a free word order language so that the assumption of an ordered frame would suggest a certain word order (or at least a preference). The main purpose of a (syntactic) subcategorization frame is to provide for syntactic constraints used for the determination of the grammatical functions. Other information of relevance is the state of the sentence (e.g., active vs. passive), the attachment borders of a dependence tree, and the required person and number agreement between verb and subject.

**Shallow strategy** Directly connected with any analysis of grammatical functions is the distinction between arguments and adjuncts as well as the choice of a unique frame for a certain verb. Recall that the output of the topological parser is a relatively flat underspecified dependence tree UDT (which still misses the grammatical functions, of course), underspecified with regard to PP attachment. This means that adjuncts are not distinguished from arguments and also that more than one frame can be compatible with the surface structure of the UDT. One solution to this problem is to simply spell out *all* possible frames compatible with the UDT and postpone resolving or reducing the ambiguity at a later stage. Instead, we chose to resolve the ambiguity heuristically by defaulting to the *maximal subcategorisation frame* that is compatible with the UDT.

Once the number and type of arguments of a given verb is determined, their functional role (e.g., subject or object) must be inferred. Because German is a semi-free world order language, the position of a phrase in a sentence (i.e., before or after the verb) does not provide reliable cues for determining grammatical functions. Instead, we check for feature compatibility between the candidate arguments and the chosen frame type. Consider as an example the sentence in Figure 3 on page 13. According to our subcategorisation dictionary the verb "haben" (*to have*) takes a nominative and an accusative NP as its complements. "Gewinn" (*revenue*) will be selected as the object of "hat" (*has*), only if it has accusative case; similarly, "Siemens" will be the subject only if it is nominative and agrees in number with the verb "hat" (*has*).

Feature checking is performed basically by looping through the dependent elements and checking whether there is morpho-syntactic agreement between the dependent and subcat arguments. Feature checking is performed by a simple (but fast) unifier which operates on feature vectors. Thus, the morpho–syntactic information of the dependent and subcat elements are expanded into feature vectors, where missing features of the subcat elements are set to the anonymous variable :no, so that information can be inherited from their dependents. One exception concerns nominative subcat arguments. In this case a feature vector is created by merging the case information with the feature vector of the verbal head. This is important in order to ensure that only nominative NPs are considered as subjects. We check for case agreement for all types of NP and PP arguments and for person agreement between the verb and its candidate subject. Other useful information for inferring grammatical relations is whether the verb is active or passive and the attachment borders of the dependency tree.

The grammatical functions recognized by GFR correspond to a set of role labels, implicitly ordered according to an *obliquity hierarchy*: SUBJ (deep subject), OBJ (deep object), OBJ1 (indirect object), P-OBJ (prepositional object), and XCOMP (subcategorized sub–clause). These labels are meant to denote *deep grammatical functions*, such that, for instance, the notion of subject and object does not necessarily correspond to the surface subject and direct object in the sentence. This is precisely the case for passive sentences, whose arguments are assigned the same roles as in the corresponding active sentence.

## 6 System performance

**Evaluation of lexical and phrasal level** We have performed a detailed evaluation on a subset of 20,000 tokens from a German text document (a collection of business news articles from the "Wirtschaftswoche") of 197,116 tokens (1.26 MB). The following table summarizes the results for the word and fragment level using the standard recall and precision measure:

| **Component** | **Recall** | **Precision** |
|---|---|---|
| Compound analysis | 98.53% | 99.29% |
| POS-filtering | 74.50% | 96.36% |
| Named entity (including dynamic lexicon) | | |
| *person names* | 81.27% | 95.92% |
| *organization* | 67.34% | 96.69% |
| *locations* | 75.11% | 88.20% |
| Fragments (NP, PP) | 76.11% | 91.94% |

In the case of compounds we measured whether a correct morpho-syntactical segmentation was determined ignoring, however, whether the segmentation was also the semantically most plausible one.[10] Evaluation of the POS-filtering showed an increase in the number of unique POS-assignments from 79.43% (before tagging) to 95.37% (after tagging). This means that from the approximately 20% ambiguous words about 75% were disambiguated with a precision of 96.36%. What concerns named entities, we only considered organizations, people's names and locations, because these are the more difficult ones and because the dynamic lexicon is automatically created for these alone. Our current experiments show that our named entity finder has a promising precision. The smaller recall is mainly due to the fact that we wanted to measure the performance of the dynamically created lexicon so we only used a small list of the 50 most well-known company names. In case of fragment recognition we only considered simple phrase structures, including coordinated NP's and NP's whose head is a named entity, but ignoring attachment and embedded sub–clauses.

**Evaluation of parsing** The DC-PARSER has been developed in parallel with the named entity and fragment recognizer and evaluated separately. In order to evaluate the DC-PARSER we collected a test corpus of 43 messages from different press releases (viz. DEUTSCHE PREESSEAGENTUR (dpa), ASSOCIATED PRESS (ap) and REUTERS) and different domains (equal distribution of politics, business, sensations). The corpus contains 400 sentences with a total of 6306 words. Note that it was also created after the DC-PARSER and all grammars were finally implemented. Table 1 shows the results of the evaluations

[10]We are not aware of any other publications describing evaluations of German compounding algorithms.

| Criterium | Matching of annotated data and results | Used by module |
|---|---|---|
| `Borders` | start and end points | verbforms, BC |
| `Type` | start and end points, type | verbforms, BC, MC |
| `Partial` | start or end point, type | BC |
| `Top` | start and end points, type for the largest tag | MC |
| `Struct1` | see `Top`, plus test of substructures using `Partial` | MC |
| `Struct2` | see `Top`, plus test of substructures using `Type` | MC |

Figure 12: Correctness criteria used during evaluation.

(the F-measure was computed with $\beta$=1). We used the correctness criteria as defined in Figure 12.

The evaluation of each component was measured on the basis of the result of all previous components. For the BC and MC module we also measured the performance by manually correcting the errors of the previous components (denoted as "isolated evaluation"). In most cases the difference between the precision and recall values is quite small, meaning that the modules keep a good balance between coverage and correctness. Only in the case of the MC-module the difference is about 5%. However, the result for the isolated evaluation of the MC-module suggests that this is mainly due to errors caused by previous components.

A more detailed analysis showed that the majority of errors were caused by mistakes in the preprocessing phase. For example, ten errors were caused by an ambiguity between different verb stems (only the first reading is chosen) and ten errors because of wrong POS-filtering. Seven errors were caused by unknown verb forms, and in eight cases the parser failed because it could not handle the ambiguities of some word forms properly, since they were either a separated verb prefix or adverb.

**Run-time behavior** The evaluation of the DC-PARSER was performed with the LISP-based version of SMES (cf. (Neumann et al., 1997)) by replacing the original bidirectional

| **Verb-Module** | | | | | | |
|---|---|---|---|---|---|---|
| *correctness* | *Verbfragments* | | | *Recall* | *Precision* | *F-measure* |
| *criterium* | total | found | correct | in % | in % | in % |
| `Borders` | 897 | 894 | 883 | 98.43 | 98.77 | 98.59 |
| `Type` | 897 | 894 | 880 | 98.10 | 98.43 | 98.26 |
| **Base-Clause-Module** | | | | | | |
| *correctness* | *BC-Fragments* | | | *Recall* | *Precision* | *F-measure* |
| *criterium* | total | found | correct | in % | in % | in % |
| `Type` | 130 | 129 | 121 | 93.08 | 93.80 | 93.43 |
| `Partial` | 130 | 129 | 125 | 96.15 | 96.89 | 96.51 |
| **Base-Clause-Module** (isolated evaluation) | | | | | | |
| *correctness* | *Base-Clauses* | | | *Recall* | *Precision* | *F-measure* |
| *criterium* | total | found | correct | in % | in % | in % |
| `Type` | 130 | 131 | 123 | 94.61 | 93.89 | 94.24 |
| `Partial` | 130 | 131 | 127 | 97.69 | 96.94 | 97.31 |
| **Main-Clause-Module** | | | | | | |
| *correctness* | *Main-Clauses* | | | *Recall* | *Precision* | *F-measure* |
| *criterium* | total | found | correct | in % | in % | in % |
| `Top` | 400 | 377 | 361 | 90.25 | 95.75 | 92.91 |
| `Struct1` | 400 | 377 | 361 | 90.25 | 95.75 | 92.91 |
| `Struct2` | 400 | 377 | 356 | 89.00 | 94.42 | 91.62 |
| **Main-Clause-Module** (isolated evaluation) | | | | | | |
| *correctness* | *Main-Clauses* | | | *Recall* | *Precision* | *F-measure* |
| *criterium* | total | found | correct | in % | in % | in % |
| `Top` | 400 | 389 | 376 | 94.00 | 96.65 | 95.30 |
| `Struct1` | 400 | 389 | 376 | 94.00 | 96.65 | 95.30 |
| `Struct2` | 400 | 389 | 372 | 93.00 | 95.62 | 94.29 |
| **complete analysis** | | | | | | |
| *correctness* | *all components* | | | *Recall* | *Precision* | *F-measure* |
| *criterium* | total | found | correct | in % | in % | in % |
| `Struct2` | 400 | 377 | 339 | 84.75 | 89.68 | 87.14 |

Table 1: Results of the evaluation of the topological structure

shallow bottom-up parsing module with the DC-PARSER. The average run-time per sentence (average length 26 words) is 0.57 sec. All components other than the topological parser (i.e., core technology, all lexical components, and the phrasal grammars for named entities, NP, PP, and the verb groups) are implemented in C++. The run-time behavior is already encouraging: processing of the mentioned German text document (1.26 MB) up to the fragment recognition level needs about 32 seconds on a PentiumIII, 500 MHz, 128 RAM, which corresponds to about 6160 words per second. This includes recognition of 11,574 named entities and 67,653 phrases.

## 7 Related work

To our knowledge, there are only very few other systems described which process free German texts. The new shallow text processor is a successor to the one used in the SMES–system, an IE-core system for real world German text processing (Neumann et al., 1997). Here, a bidirectional verb-driven bottom-up parser was used, where the problems described in this paper concerning parsing of longer sentences were encountered. Another similar divide-and-conquer approach using a chart-based parser for analysis of German text documents was presented by (Wauschkuhn, 1996). Nevertheless, comparing its performance with our approach seems to be rather difficult since he only measures how often his parser finds at least one result for an un-annotated test corpus (where he reports 85.7% "coverage" of a test corpus of 72,000 sentences) without measuring the accuracy of the parser. In this sense, our parser achieved a "coverage" of 94.25% (computing $found/total$), almost certainly because we use more advanced lexical and phrasal components, e.g., pos-filter, compound and named entity processing. (Peh and Ting, 1996) also describe a divide-and-conquer approach based on statistical methods, where the segmentation of the sentence is done by identifying so–called link words (solely punctuation marks, conjunctions and prepositions) and disambiguating their specific role in the sentence. On an annotated test corpus of 600 English sentences they report an accuracy of 85.1% based on the correct recognition of part-of-speech, comma and conjunction disambiguation, and exact noun phrase recognition.

## 8 Conclusion and Future work

We have presented an advanced domain-independent shallow text extraction and navigation system for processing of real-world German texts. The system is implemented based on advanced finite state technology and uses sophisticated linguistic knowledge sources. The system is very robust and efficient (at least from an application point of view) and has a very good linguistic coverage for German.

Our future work will concentrate on interleaved bi-lingual (German and English) as well as cross-lingual text processing applications on basis of SPPC's uniform core technology, and the exploration of integrated shallow and deep NL processing. Concerning the first issue we have already implemented prototypes of SPPC for English and Japanese up to the recognition of (simple) fragments. Processing of complex phrases and clause patterns will be realized through the compilation of stochastic lexicalized tree grammars (SLTG) into cascades of WFST. An SLTG will be automatically extracted from existing tree banks following our work described in (Neumann, 1998).

A further important research direction will be the integration of shallow and deep processing so that a deep language processor might be called for those structures recognized by the shallow processor as being of great importance. Consider (really) complex nominal phrases, for example. In the case of information extraction (IE), nominal entities are mostly used for filling slots of relational templates (e.g., filling the "company" slot in an "management appointment" template). However, because of a shallow NP analysis, it is often very difficult to decide which parts of an NP actually belong together. This problem is even more complex if we consider free word languages like German. However, taking advantage of SPPC's *divide-and-conquer* shallow parsing strategy, it would now be possible to call a deep parser only to those separated field elements which correspond to sequences of simple NPs and PPs (which could have been determined by the shallow parser, too). From this point of view, the shallow parser is used as an efficient preprocessor for dividing sentences into syntactically valid smaller units, where the deep parser's task would be to identify the exact constituent structure only on demand.

## Acknowledgements

The research underlying this paper was supported by a research grant from the German Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) to the DFKI project PARADIME, FKZ ITW 9704. Many thanks to Christian Braun, Thierry Declerck, Markus Becker, and Milena Valkova for their great support during the development of the system.

## References

Abney, S. 1996. Partial parsing via finite-state cascades. Proceedings of the ESSLLI 96 Robust Parsing Workshop.

Appelt, D., J. Hobbs, J. Bear, D. Israel, and M. Tyson. 1993. Fastus: A finite state processor for information extraction from real world text. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, August.

Assadi, H. 1997. Knowledge acquisition from texts: Using an automatic clustering method based on noun-modifier relationship. In *35th Annual Meeting of the Association for Computational Linguistics/8th Conference of the European Chapter of the Association for Computational Linguistics*, Madrid.

Bikel, D., S. Miller, R. Schwartz, and R. Weischedel. 1997. Nybmle: a high-performance learning name-finder. In *5th International Conference of Applied Natural Language*, pages 194–200, Washington, USA, March.

Borthwick, A. 1999. *A Maximum Entropy Approach to Named Entity Recognition*. Ph.D. thesis, New York University, Department of Computer Science, Courant Institute.

Brill, E. 1993. Automatic grammar induction and parsing free text: A transformation-based approach. In *31th Annual Meeting of the Association for Computational Linguistics*, Ohio.

Buchholz, Sabine. 1996. Entwicklung einer lexikographischen Datenbank für die Verben des Deutschen. Master's thesis, Universität des Saarlandes, Saarbrücken.

Busemann, S., T. Declereck, A. Diagne, L. Dini, J. Klein, and S. Schmeier. 1997. Natural language dialogue service for appointment scheduling agents. In *5th International Conference of Applied Natural Language*, pages 25–32, Washington, USA, March.

Ciravegna, F., A. Lavelli, N. Mana, L. Gilardoni, S. Mazza, M. Ferraro, J. Matiasek, W. Black, F. Rinaldi, and D. Mowatt. 1999. Facile: Classifying texts integrating pattern matching and information extraction. In *Proceedings of 16th International Joint Conference on Artificial Intelligence"*, Stockholm.

Cormen, T., C. Leiserson, and R. Rivest. 1992. *Introduction to Algorithms*. The MIT Press, Cambridge, MA.

Cowie, J. and W. Lehnert. 1996. Information extraction. *Communications of the ACM*, 39(1):51–87.

Engel, Ulrich. 1988. *Deutsche Grammatik*. 2., verbesserte edition. Julius Groos Verlag, Heidelberg.

Federici, Stefano, Simonetta Monyemagni, and Vito Pirrelli. 1996. Shallow parsing and text chunking: A view on underspecification in syntax. In *Workshop on Robust Parsing, 8th ESSLLI*, pages 35–44.

Finkler, W. and G. Neumann. 1988. Morphix: A fast realization of a classification–based approach to morphology. In H. Trost, editor, *Proceedings der 4. Österreichischen Artificial–Intelligence Tagung, Wiener Workshop Wissensbasierte Sprachverarbeitung*, Berlin, August. Springer.

Gardent, C. and B. Webber. 1998. Describing discourse semantics. In *Proceedings of the 4th International Workshop Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 50–53, University of Pennsylvania, PA.

Grinberg, Dennis, John Lafferty, and Daniel Sleato. 1995. A robust parsing algorithm for link grammars. In *Proceedings of the International Parsing Workshop 95*.

Grishman, R. 1995. The NYU MUC-6 System or Where's the Syntax? In *Sixth Message Understanding Conference (MUC-6)*. Morgan Kaufmann, November.

Grishman, R. and B. Sundheim. 1996. Message Understanding Conference – 6: A Brief History. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 466–471, Kopenhagen, Denmark, Europe.

Mohri, M. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23.

Mohri, M., F. Pereira, and M. Riley. 1996. A rational design for a weighted finite-state transducer library. Technical report, AT&T Labs - Research.

Muskens, R. and E. Krahmer. 1998. Description theory, ltags and underspecified semantics. In *Proceedings of the 4th International Workshop Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 112–115, University of Pennsylvania, PA.

Neumann, G. 1998. Automatic extraction of stochastic lexicalized tree grammars from treebanks. In *4th workshop on tree-adjoining grammars and related frameworks*, Philadelphia, PA, USA, August.

Neumann, G., R. Backofen, J. Baur, M. Becker, and C. Braun. 1997. An information extraction core system for real world german text processing. In *5th International Conference of Applied Natural Language*, pages 208–215, Washington, USA, March.

Neumann, G., C. Braun, and J. Piskorski. 2000. A divide-and-conquer strategy for shallow parsing of german free texts. In *Proceedings of the 6th International Conference of Applied Natural Language*, Seattle, USA, April.

Neumann, G. and S. Schmeier. 1999. Combining shallow text processing and machine learning in real world applications. In *Proceedings of the IJCAI-99 workshop IRF-2 "Machine Learning for Information Filtering", T. Joachims (ed)*, Stockholm, Sweden.

Oflazer, K. 1999. Dependence parsing with a finite state approach. In *37th Annual Meeting of the Association for Computational Linguistics*, Maryland.

Peh, Li Shiuan and Christopher Hian Ann Ting. 1996. A divide-and-conquer strategy for parsing. In *Proceedings of the ACL/SIGPARSE 5th International Workshop on Parsing Technologies*, pages 57–66.

Piskorski, J. 1999. DFKI FSM Toolkit. Technical report, Language Technology Laboratary, German Research Center for Artificial Intelligence (DFKI).

Piskorski, J. and G. Neumann. 2000. An intelligent text extraction and navigation system. In *Proceedings of the 6th International Conference on Computer-Assisted Information Retrieval (RIAO-2000)*. Paris, April.

Roche, E. and Y. Schabes. 1995. Deterministic part-of-speech tagging with finite state transducers. *Computational Linguistics*, 21(2):227–253.

Roche, E. and Y. Schabes. 1996. Introduction to finite-state devices in natural language processing. Technical report, Mitsubishi Electric Research Laboratories, TR-96-13.

SAIC, editor. 1998. *Seventh Message Understanding Conference (MUC-7)*, http://www.muc.saic.com/. SAIC Information Extraction.

Sekine, S. and C. Nobata. 1998. An information extraction system and a customization tool. In *Proceedings of Hitachi workshop-98*, http://cs.nyu.edu/cs/projects/proteus/sekine/.

Staab, S., C. Braun, A. Düsterhöft, A. Heuer, M. Klettke, S. Melzig, G. Neumann, B. Prager, J. Pretzel, H.-P. Schnurr, R. Studer, H. Uszkoreit, and B. Wrenger. 1999. GETESS — searching the web exploiting german texts. In *CIA'99 — Proceedings of the 3rd Workshop on Cooperative Information Agents*, LNCS, Berlin. Springer.

Sundheim, B., editor. 1995. *Sixth Message Understanding Conference (MUC-6)*, Washington. Distributed by Morgan Kaufmann Publishers, Inc.,San Mateo, California.

Wauschkuhn, Oliver. 1996. Ein werkzeug zur partiellen syntaktischen analyse deutscher textkorpora. In Dafydd Gibbon, editor, *Natural Language Processing and Speech Technology. Results of the Third KONVENS Conference*. Mouton de Gruyter, Berlin, pages 356–368.

## List of Figures