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

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 which are realized by means of cascaded weighted finite state machines and generic dynamic tries. The system has been fully implemented for German which 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 of 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 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

There is no doubt that the amount of textual information electronically available today has passed its critical mass leading to the emerging problem that the more electronic text data is available the more difficulty it is to find, extract or merge relevant information. Furthermore, the growing amount of unstructured on-line documents and the growing population of users who want to extract an ever-widening diversity of types of information, requires easy portability of information services to new applications on demand.

In order to master the information overflow new technologies for intelligent information and knowledge management systems are explored in a number of research areas like information retrieval, adaptive information extraction, text data mining, knowledge acquisition from text documents or semi-structured data bases. In all of these cases the degree and amount of structured data a text pre-processor is able to extract is crucial for the further high-level processing of the extracted data (see also Figure 1). In the majority of current information management approaches linguistic text analysis is restricted to be performed on the word level (e.g., tokenization, steming, 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 very few 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 todays 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)).

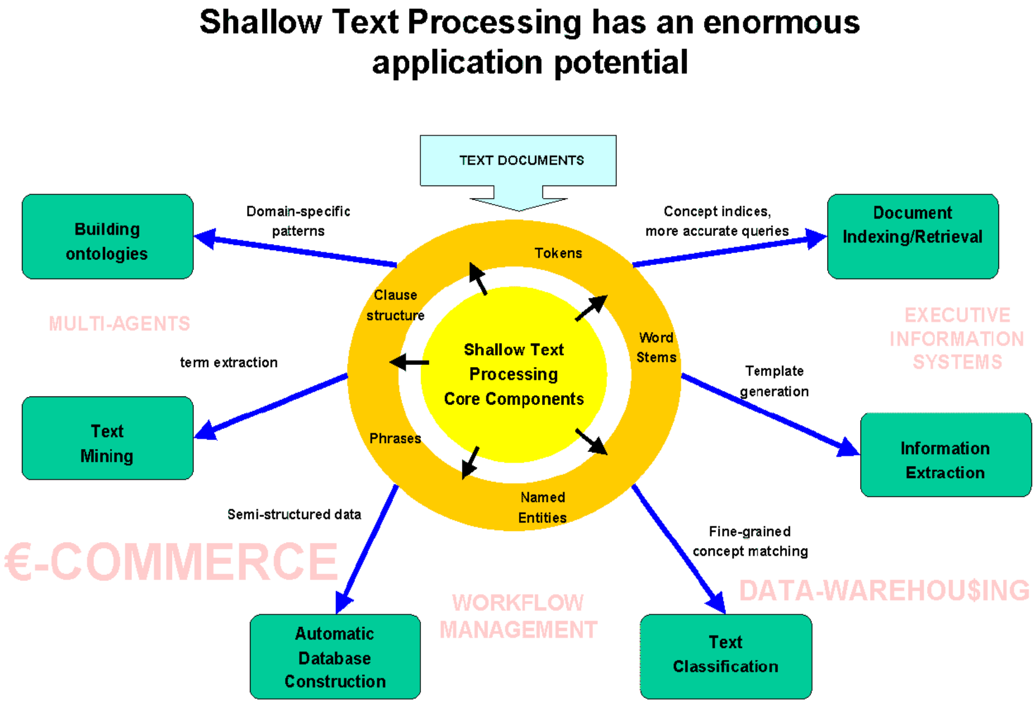

Figure 1: Shallow text processing has an high application impact.

Why shallow processing? Recently, the development of highly domain adaptive systems on the basis of re-usable, modular technologies are becoming important in order to support easy customization of text extraction core technology. Such adaptive techniques are critical for rapidly porting systems to new domains. In principle it would be possible to use an exhaustive and deep general 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 would be possible to formalize and represent the complete grammar of a natural language, the system would still need a very high degree of robustness and efficiency. It has been shown that realizing such a system is at least today impossible. However, in order to fulfill the ever increasing demands for improved processing of real-world texts, NL researchers especially in the area of information extraction 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" text 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 pragmatical, e.g., by restricting the depth of recursion on the basis of a corpus analysis or by making us of heuristic rules, like "longest matching substrings". This engineering view on language has lead to a renaissance and improvement of well-known efficient techniques, most notably finite state technology for parsing.

NLP as normalization If we consider the main task of free text processing as a preprocessing stage for extracting as much as linguistic structure as possible (as a basis for the computation of domain-specific complex relations) then it is worthwhile to treat NL analysis as a *step-wise 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 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" (*to 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 advantage 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)).[2]

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. The major tasks of the core shallow text processing tools are

- extract as much linguistic structure from text as possible,
- represent all extracted information (together with its frequency) in one data structure as compactly as possible in order to ease uniform access the set of solutions.

In order to achieve this in the desired efficient and robust way, SPPC makes use of advanced finite state technology on all levels of processing and a data structure called *text chart*. SPPC has a high application potential and has already been used in different application areas ranging from processing of 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.

2 System architecture

In this section we will introduce the architecture of SPPC shown in Figure 2. It consists of two major components, a) the Linguistic Knowledge Pool LKP and b) STP, the shallow text processor itself. STP processes a NL-text through a chain of modules. We distinguish two

[2]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 seen as paraphrases because they "mean" the same.

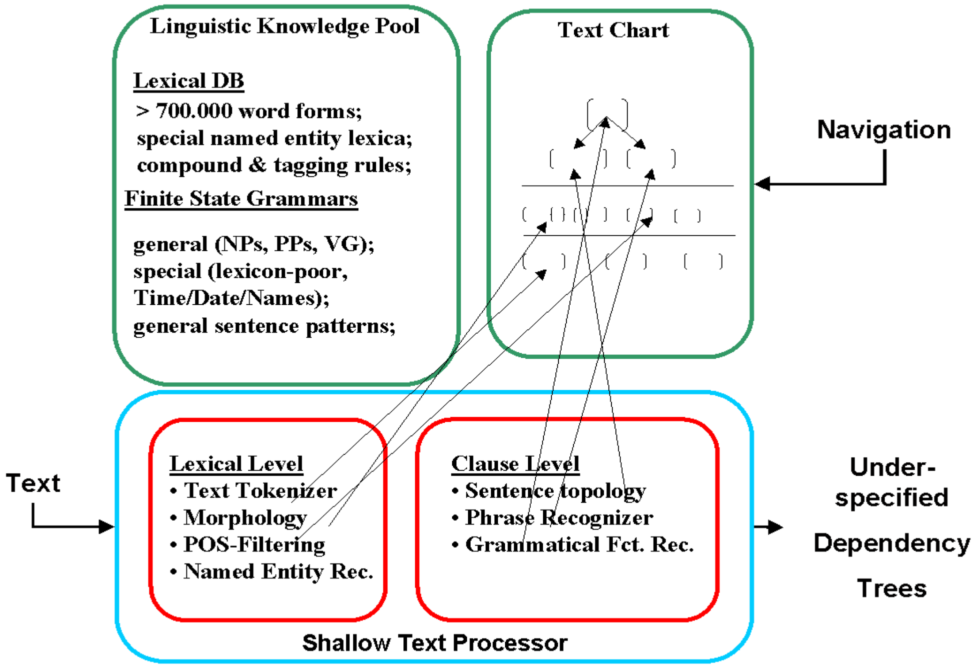

Figure 2: The blueprint of the system architecture.

primarily levels of processing, the *lexical level* and the *clause level*. Both are subdivided into several components.

The first component on the lexical level is the TEXT TOKENIZER which maps sequences of characters into greater units, usually called tokens. Each token, which is identified as a potential word form is then morphological processed by the MORPHOLOGY. Morphological processing includes retrieval of lexical information (on the basis of a word form lexicon), on-line recognition of compounds and hyphen coordination. Afterwards the control is passed to a POS-FILTER, which performs word-based disambiguation. In the final step, the NAMED-ENTITY-FINDER identifies specialized expressions for time, date and several name expressions.

The task of the clause level modules is to construct a hierarchical structure over the words of a sentence. In contrast to deep parsing strategies, where phrases and clauses are interleaved, we separate these steps. The clause level processing in STP is divided into three modules. 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 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. In the final step, the grammatical structure is computed using a large subcategorization lexicon. The final output of the parser for a sentence, is then an underspecified dependence structure, where only upper bounds for attachment and scoping of modifiers are expressed (see Figure 11).

All components of SPPC are realized by means of cascaded weighted finite state machines and generic dynamic tries. Thus before we are going to describe the modules in more detail we will first introduce this core technology.

3 Core technology

Finite-state devices recently have been used extensively in many areas of language technology (Mohri, 1997), especially in shallow processing, which is motivated by both computational and linguistic arguments. Computationally, finite-state devices are time and space efficient due to their closure properties and the existence of efficient determinization, minimization and other optimization and transformation algorithms. From the linguistic point of view local recognition patterns can be easily and intuitively expressed as finite-state devices. Obviously, there exists much more powerful formalisms like context-free or unification based grammars, but since industry prefers more pragmatic solutions finite-state technology is recently in the center of attention. Hence core finite-state software was developed, consisting of the DFKI FSM Toolkit for constructing and manipulating finite-state devices (Piskorski, 1999) and generic dynamic tries, which we describe briefly in this section.

3.1 DFKI FSM Toolkit

The DFKI FSM Toolkit is a library of tools for building, combining and optimizing finite-state machines (FSM), which are generalizations of weighted finite-state automata (WFSA) and transducers (WFST) (Mohri, 1997). Finite-state transducers are automata

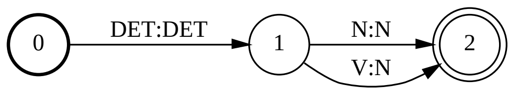

Figure 3: A simple FST

for which each transition has an output label in addition to the more familiar input label. For instance, the finite-state transducer in Figure 3 represents a contextual rule for part-of-speech disambiguation: change tag of word form from noun or verb to noun if the previous word is a determiner. Weighted automata or transducers are automata or transducers in which each transition has a weight as well as input/output labels. There already exist a variety of well known finite-state packages, e.g., FSA6 (van Noord, 1998), Finite-State Tool from Xerox Parc (Kartunen et al., 1996) or AT&T FSM Tools (Mohri, Pereira, and Riley, 1996). However only few of them provide algorithms for WFSA's and even less support operations for WFST's. Algorithms on WFSA's have strong similarities with their better known unweighted counterparts, but the proper treatment of weights introduces some additional computations. On the other hand algorithms on transducers are in many cases much more complicated than corresponding algorithms for automata. Further the closure properties of transducers are much weaker than those of automata (Roche and Schabes, 1996).

The DFKI FSM Toolkit is divided in two levels: an user-program level consisting of a stand-alone application for manipulating FSM's by reading and writing to files and a C++-library level consisting of an archive of C++ classes, methods and functions, which implement the user-program level operations. The latter allows an easy embedding of single elements of the toolkit into any other application. Finite-state machines in the DFKI FSM Toolkit can be represented either in compressed binary form, optimized for processing or textual format. A semiring on real numbers, consisting of an associative extension operator, an associative and commutative summary operator and neutral elements of these operators (Cormen, Leiserson, and Rivest, 1992), may be chosen in order to determine the semantics of the weights of an FSM. The associative extension operator is used for computing the weight of a path (combining the weights of arcs on the path), whereas the summary operator is used to summarize path weights. Most of the operations work with arbitrary semirings on real numbers (only a computational representation of the semiring is needed). The operations of the package are divided into three main pools:

1. converting operations: converting textual representation into binary format and vice versa, creating graph representations for FSMs, etc.

2. rational and combination operations: union, closure, reversion, inversion, concatenation, local extension, intersection and composition

3. equivalence transformations: determinization, bunch of minimization algorithms, epsilon elimination, removal of inaccessible states and transitions

Most of the provided operations are based on recent approaches proposed in (Mohri, 1997), (Mohri, Pereira, and Riley, 1996), (Roche and Schabes, 1995) and (Roche and Schabes, 1996). Some of the operations may only be applied to a restricted class of FSM's due to the limited closure properties of finite-state transducers, for instance only epsilon-free transducers are closed under intersection. Furthermore only a small class of transducers is determinizable (Mohri, 1997), but it turned out that even in languages with free word order like German local phenomena can be captured by means of using determinizable transducers.

The architecture and functionality of DFKI FSM Toolkit is mainly based on the tools developed by AT&T. As well as AT&T we allow only use of letter transducers (only a simple input/output symbol as a label of a transition is allowed) since most of the algorithms require such type of transducers as input and thus time consuming conversion steps may be omitted. Unlike AT&T we do not provide operations for computing shortest paths, but instead provide some operations not included in the AT&T package, which are of great importance in shallow text processing. The algorithm for local extension, which is crucial for an efficient implementation of Brill's tagger, proposed in (Roche and Schabes, 1995) was adapted for the case of weighted finite-state transducers and a very fast operation for direct incremental construction of minimal deterministic acyclic finite-state automata (Daciuk, Watson, and Watson, 1998) is provided. Besides that, we modified the general algorithm for removing epsilon-moves (Piskorski, 1999), which is based on the computation of the transitive closure of a graph representing epsilon moves. Instead of computing the transitive closure of the entire graph (complexity $O(n^3)$) we first try to remove as many epsilon transitions as possible and then compute the transitive closure of each connected component in the graph representing the remaining epsilon transitions. This proved to be a considerable improvement in practice.

3.2 Generic Dynamic Tries

The second crucial core tool is the generic dynamic trie, a parameterized tree-based data structure for efficiently storing sequences of elements of any type, where each such sequence is associated with an object of some other type. Unlike classical tries for storing sequences of letters (Cormen, Leiserson, and Rivest, 1992) in lexical and morphological processing (stems, prefixes, inflectional endings, full forms etc.) generic dynamic tries are capable of storing far more complex structures, like for example sequences containing components of a phrase, where such components contain lexical information. The trie in Figure 4 is used for storing verb phrases and their frequencies, where each component of the phrase is represented as pair containing the string and part-of-speech information. In addition to the standard functions for inserting and searching, an efficient deletion function is provided, which is indispensable in self organizing lexica, like for instance lexica for storing most frequent words, which are updated periodically. Furthermore, a variety of other more complex functions relevant for linguistic processing are available, including recursive trie traversal and functions supporting recognition of longest and shortest prefix/suffix of a given sequence in the trie. The latter ones are a crucial part of the algorithms for the decomposition of compounds and hyphen coordination in German (see sec. 4).

4 Lexical processing

Text Tokenizer The TEXT TOKENIZER maps sequences of consecutive characters into larger units called tokens and identifies their types. Currently we use more than 50 domain-

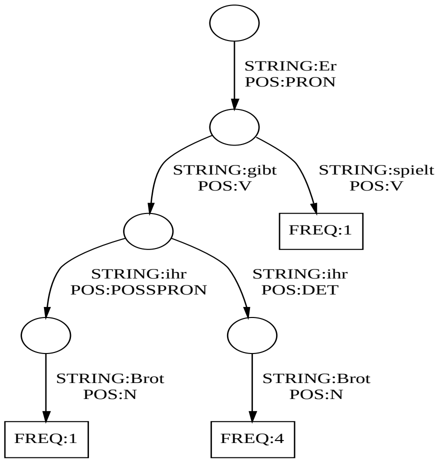

Figure 4: Example of a generic dynamic trie. The following pathes are encoded: (1) "Er(PRON) gibt(V) ihr(DET) Brot(N)." (*He gives her bread.*), (2) "Er(PRON) gibt(V) ihr(POSSPRON) Brot(N).", and "(3) Er(PRON) spielt(V)." (*He plays.*)

independent token classes including generic classes for semantically ambiguous tokens (e.g., "10:15" could be a time expression or volleyball result, hence we classify this token as number-dot compound) and complex classes like abbreviations or complex compounds (e.g., "AT&T-Chief"). It proved that such variety of token classes simplifies the processing of subsequent submodules significantly.

Morphology Each token identified as a potential word form is submitted to the morphological analysis including on-line recognition of compounds (which is crucial since compounding is a very productive process of the German language) and hyphen coordination. Each token recognized as a valid word form is associated with the list of its possible readings, characterized by stem, inflection information and part of speech category. Here follows the result of the morphological analysis for the word "wagen" (which has at least two readings: *to dare* and *a car*):

STEM	WAG	
POS	V	
INFL	TENSE: PRES, PERS: 1, NUM: PL	TENSE: PRES, PERS: 3, NUM: PL
	TENSE: SUBJUNCT, PERS: 1, NUM: PL	TENSE: SUBJUNCT, PERS: 3, NUM: PL
	FORM: INFIN	
STEM	WAGEN	
POS	N	
INFL	GENDER: M, CASE: NOM, NUM: SG	GENDER: M, CASE: DAT, NUM: SG
	GENDER: M, CASE: ACC, NUM: SG	GENDER: M, CASE: NOM, NUM: PL
	GENDER: M, CASE: GEN, NUM: PL	GENDER: M, CASE: DAT, NUM: PL
	GENDER: M, CASE: ACC, NUM: PL	

The complete output of the morphology returns a list of so called lexical items represented as tuples of the form $\langle start, end, lex, compound, pref\rangle$, where:

- *start* and *end* point to first and last token of the lexical item: usually tokens are directly mapped to lexical items, but due to the handling of separations some tokens may be combined together, e.g., "Ver-" and "kauf" will be combined to "Verkauf" (sale) in forms like "Ver- und Ankauf" (*sale and purchase*),
- *lex* contains the lexical information of the word (actually *lex* only points to an adequate lexical information in the lexicon, no duplicate copies are created),
- *compound* is a boolean attribute, which is set to true in case the word is a compound,
- *pref* is the prefered reading of the word, determined by the POS FILTER for ambiguous words.

In our current run-time system we use a full-form lexicon in combination with an on-line compound analysis (see next paragraph).[3] Currently more than 700.000 German full-form

[3]The full-form lexicon has been automatically created by means of the morphological component MORPHIX, (Finkler and Neumann, 1988). The simple reason why we did not use MORPHIX in our run-time system is that it is not available in C++ with its full functionality. On the other hand-side it is clear that making use of a full-form lexicon simplifies on-line morphological processing. So that our combined approach — full-form lexicon together with on-line compounding — seems also to be an interesting practical alternative.

words (created from about 120.000 stem entries) are used. However, in combination with the compound analysis the actual lexical coverage is much higher. Several experiments on unseen texts showed an lexical coverage of about 95% from which 10%-15% are compounds (the majority of the remaining 5% are either proper names, technical terms or spelling errors).

Compound analysis Each token not recognized as a valid word form is a potential compound candidate. The compound analysis is realized by means of a recursive trie traversal. The special trie functions we mentioned earlier for recognition of longest and shortest prefix/suffix are used in order to break up the compound into smaller parts, so called compound-morphems. Often more than one lexical decomposition is possible. In order to reject invalid decompositions a set of rules for validating compound-morphems is used (this is done separately for prefixes, suffixes and infixes). Additionally a list of valid compound prefixes is used since there are compounds in German beginning with a prefix, which is not necessarily a valid word form (e.g., "Multiagenten" (*multi-agents*) is decomposed into "Multi" + "Agenten", where "Multi" is not a valid word form). Another important issue is hyphen coordination. It is mainly based on the compound analysis, but in some cases the last conjunct (coordinate element) is not necessarily a compound, e.g.,

(a) "Leder-, Glas-, Holz- und Kunststoffbranche" *leather, glass, wooden, plastic, and synthetic materials industry*

    The word "Kunststoffbranche" is a compound consisting of two morphems: "Kunstoff" and "branche". Hence "Leder-", "Glas-" and "Holz-" can be resolved to "Lederbranche", "Glasbranche" and "Holzbranche".

(b) "An- und Verkauf" *purchase and sale*

    The word "Verkauf": is a valid word form, but not a compound. An internal decomposition of this word must be undertaken in order to find suitable suffix for proper coordination of "An-".

POS Filter Since a high amount of German word forms are ambiguous, especially the word forms with verb reading and due to the fact that the quality of the results of the CLAUSE RECOGNIZER relies essentially on the proper recognition of verb groups, filtering out unplausible readings is crucial task of the shallow text processing engine. Generally, only nouns (and proper names) are written in standard German with a capitalized initial letter (e.g., "das Unternehmen" - *the enterprise* vs. "wir unternehmen" - *we undertake*). Since typing errors are relatively rare in press releases (or similar documents) the application of case-sensitive rules is a reliable and straightforward tagging means for German. However they are obviously insufficient for disambiguating word forms appearing at the beginning of the sentence. In such cases contextual filtering rules are applied. For instance, the rule which forbids the verb written with a capitalized initial letter to be followed by a finite verb would filter out the verb reading of the word "unternehmen" in the sentence "Unternehmen sind an Gewinnmaximierung interesiert." (*Enterprises are interested in maximizing their profits*). A major subclass of ambiguous word forms are those which beside the verb reading have an adjective or attributive-used-participle reading. For instance, in the sentence "Sie bekannten, die bekannten Bilder gestohlen zu haben." (*They confessed they have stolen the famous pictures.*) the word form "bekannten" is firstly used as a verb (confessed) and secondly as an adjective (famous). Since adjectives and attributive-used-participles are in most cases part of a nominal phrase a convenient rule would reject the verb reading if the previous word form is a determiner or the next word form is a noun. It is important to notice that such contextual rules are based on some regularities, but they may yield false results, like for instance in the sentence "Unternehmen sollte man aber trotzdem etwas." (*One should undertake something anyway.*), the verb reading of the word "unternehmen" would be filtered out by the contextual rule for handling verbs written with a capitalized initial letter, which we mentioned earlier. Furthermore in some cases ambiguities can not be resolved solely by using contextual rules. For instance, in order to distinguish between adverbs and predicate adjectives in German one has to know the corresponding verb (e.g. "war schnell" - *was fast* vs. "lief schnell" - *ran quickly*, but this information is not necessarily available at this stage of processing (e.g., in the sentence "Ben Johnson lief die 100m-Strecke sehr [schnell]" - *Ben*

Johnson ran very quickly the 100m distance). In addition, we use rules for filtering out the verb reading of some word forms extremely rarely used as verbs (e.g., "recht" - *right*, *to rake* (3rd person,sg)). All rules are compiled into a single finite-state transducer according to the approach described in (Roche and Schabes, 1995).[4]

4.1 Named Entity Finder

The task of the NAMED ENTITY FINDER is the identification of entities (proper names like organizations, persons, locations), temporal expressions (time, date) and quantities (monetary values, percentages, numbers). Usually, text processing systems recognize named entities primarily using local pattern-matching techniques. In our systems we first identify a phrase containing potential candidates for a named entity by using recognition patterns expressed as WFSA's, where the longest match strategy is used. Secondly, additional constraints (depending on the type of the candidate) are used for validating the candidates and an appropriate extraction rule is applied in order to recover the named entity. The output of this component is a list of so called named-entity items, represented as four-tuples of the form $\langle start, end, type, subtype \rangle$, where *start* and *end* point to the first and last lexical item in the recognized phrase, *type* is the type and *subtype* the subtype of the recognized named entity. For instance, the expression "von knapp neun Milliarden auf über 43 Milliarden Spanische Pesetas" (*from almost nine billions to more than 43 billion spanish pesetas*) will be identified as named entity of type CURRENCY and subtype CURRENCY-PREPOSITIONAL-PHRASE. Since the NAMED-ENTITY FINDER operates on a stream of lexical items, the arcs of the WFSA's representing the recognition patterns can be viewed as predicates for lexical items. There are three types of predicates:

1. *string*: s, holds if the surface string mapped by the current lexical item is of the form s
2. *stem*: s, holds if: (1) the current lexical item has a prefered reading with stem s or (2) the current lexical item does not have prefered reading, but at least one reading

[4] The manually constructed rules proved to be a useful means for disambiguation, however not sufficient enough to filter out all unplausible readings. Hence supplementary rules determined by Brill's tagger were used in order to achieve broader coverage.

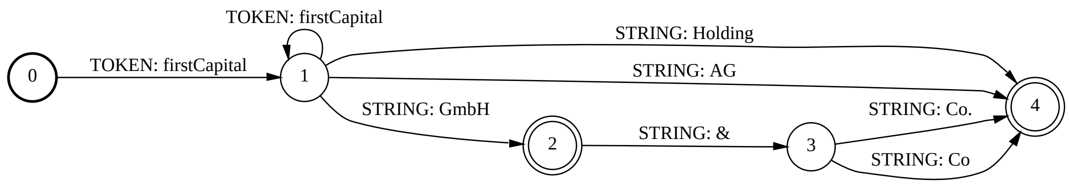

Figure 5: The automaton representing the pattern (here a simplified version) for recognition of company names.

with stem s exist

3. *token*: x, holds if the token type of the surface string mapped by current lexical item is x.

In Figure 5 we show the automaton representing the pattern for the recognition of company names. This automaton recognizes any sequence of words beginning with an upper-case letter, which are followed by a company designator (e.g., Holding, Ltd, Inc., GmbH, GmbH & Co.). A convenient constraint for this automaton should disallow the first word in the recognized fragment to be a determiner (usually it does not belong to the name) and in case the recognized fragment consists solely of a determiner followed directly by company designator, the candidate should be rejected. Hence the automaton would recognize "Die Braun GmbH & Co.", but only "Braun GmbH & Co." would be extracted as named entity, whereas the phrase "Die GmbH & Co." would be rejected.

Dynamic lexicon Many of the proper names can be recognized because of the specific context they appear in. For instance company names often include so called company designators as we already seen above (e.g., GmbH, Inc., Ltd etc.). However such company names may appear later in the text without such designators and they would probably refer to the same object. Hence we use a *dynamic lexicon* to store recognized named entities (organizations, persons) without their designators in order to identify subsequent occurrences correctly. For example after recognizing the company name "Apolinaris &

Schweepes GmbH & Co." we would save the phrase "Apolinaris & Schweepes" in this dynamic lexicon. The succeeding occurrences of the latter phrase without a designator would be then treated as a named entity. However proper names identified in such way could be somewhat ambiguous. Let us consider the proper name "Ford", which could be either a company name ("Ford Ltd.") or a last name of some person ("James Ford"). Obviously the local context of such entities does not necessarily provide sufficient information in order to infer its category. Therefore we set a reference for the entity identified by a lookup in the dynamic lexicon to the last occurrence of the same entity with the designator (heuristic) in the analyzed text. Unfortunately another problem arises: a proper name stored in the dynamic lexicon consisting solely of one word may be also a valid word form. At this stage it is usually impossible to decide for such words whether they are proper names or not. Hence they are recognized as candidates for named entities (there are three special candidate types: organization candidate, person candidate and location candidate). For instance, after recognizing the entity "Braun AG", the subsequent occurrence of "Braun" would be recognized as an organization candidate since "braun" is a valid word form (*brown*). Disambiguation is performed on a higher level of processing or by applying some heuristics. Finally we also handle proper recognition of acronyms since they are usually introduced in the local context of named entities (e.g., "Bayerische Motorwerke (BMW)") and this can be achieved by exploring the structure of the named entities.

Recognition of named entities could be postponed and integrated into the fragment recognizer (see next section), but performing this task at this stage of processing seems to be more appropriate. Firstly the results of POS-filtering could be partially verified and improved and secondly the amount of the word forms to be processed by subsequent modules could be considerably reduced. For instance the verb reading of the word form "achten" (*watch* vs. *eight*) in the time expression "am achten Oktober 1995" (*at the eight of the October 1995*) could be filtered out if not done yet. Furthermore the recognition of sentence boundaries, which is performed afterwards can be simplified since the NAMED ENTITY FINDER reduces the amount of potential sentence boundaries. For example, some tokens containing punctuation signs, like abbreviations or number-dot compounds are recognized as part of named entities (e.g., "an Prof. Ford" – *for Prof. Ford* or "ab 2. Oktober 1995" – *from the second October 1995*).

5 Clause level processing

In most of the well-known shallow text processing systems (cf. (Sundheim, 1995) and (SAIC, 1998)) cascaded chunk parsers are used 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:

> "[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 be not 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 according to the linguistic *field theory* (cf. (Engel, 1988)) are determined domain-independently. In a second phase,

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

This information couldn't be verified by the Border Police, Kinkel spoke of horrible figures that he didn't believe.

Figure 6: An example of a topological structure.

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 (see Figure 6)

This approach offers several advantages:

- improved robustness, because parsing of the sentence topology is based only on simple indicators like verbgroups and conjunctions and their interplay,
- the resolution of some ambiguities, including relative pronouns vs. determiner, subjunction vs. preposition and sentence coordination vs. NP coordination, and
- a high degree of modularity (easy integration of domain-dependent subcomponents).

5.1 A Shallow Divide-and-Conquer Strategy

The DC-PARSER consists of two major domain-independent modules based on finite state technology: 1) construction of the topological sentence structure, and 2) application of phrasal grammars on each determined subclause. Figure 7 shows an example result of the DC-PARSER. It shows the separation of a sentence into the front field (VF), the verb group (VERB), and the middle field (MF). The elements of different fields have been computed by means of fragment recognition which takes place after the (possibly recursive) topological structure has been computed. Note that the front field consists only of one but complex subclause which itself has an internal field structure.

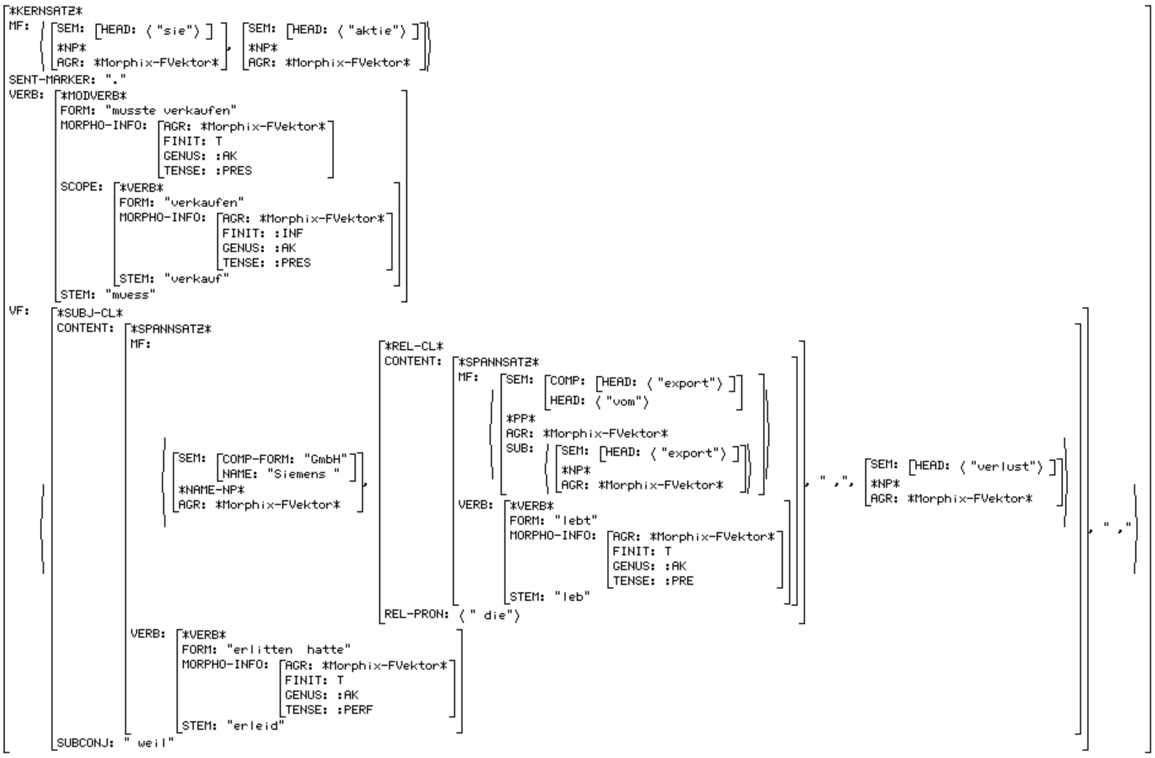

Figure 7: The result of the DC-PARSER for the sentence "Weil die Siemens GmbH, die vom Export lebt, Verluste erlitten hat, musste sie Aktien verkaufen." *Because the Siemens GmbH which strongly depends on exports suffered from losses they had to sell some of the shares.* (abbreviated where convenient)

5.1.1 Topological structure

The DC-PARSER applies cascades of finite-state grammars to the stream of lexical tokens and named entities in order to determine the topological structure of the sentence according to the linguistic field theory (Engel, 1988).

Based on the fact that in German a verb group (like "hätte überredet werden müssen" — **have convinced been should* meaning *should have been convinced*) 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 (VF), the left verb part, middle field (MF), right verb part, and rest field

```
Weil die Siemens GmbH,die vom Export lebt, Verluste erlitt, musste sie Aktien verkaufen.
                                          ⇓
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 for the sentence of figure 7.

(RF). Subclauses can also be expressed in that way such that the left verb part is either empty or occupied by a relative pronoun or a subjunction element, and the complete verb group is placed in the right verb part, cf. Figure 7. Note that each separated field can be arbitrarily complex with very few restrictions on the ordering of the phrases inside a field.

Recognition of the topological structure of a sentence can be described by the following four phases realized as cascade of finite state grammars (see also Figure 2; Figure 8 shows the different steps in action). Initially, the stream of tokens and named entities is separated into a list of sentences based on punctuation signs.

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 parts of discontinuous verb groups (e.g., separated LVP and RVP or separated verbs and verb-prefixes) cannot be put together at that step of processing because one needs contextual information which will only be available in the next steps. The major problem at this phase is not a structural one but the massive morphosyntactic ambiguity of 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, Monyemagni, and Pirrelli, 1996).

$$\begin{bmatrix} \text{Type} & \texttt{VG-final} \\ \text{Subtype} & \texttt{Mod-Perf-Ak} \\ \text{Modal-stem} & \text{könn} \\ \text{Stem} & \text{lob} \\ \text{Form} & \text{nicht gelobt haben kann} \\ \text{Neg} & \text{T} \\ \text{Agr} & \ldots \end{bmatrix}$$

Figure 9: The structure of the verb fragment "nicht gelobt haben kann" – **not praised have could-been* meaning *could not have been praised*

These types, like for example `Fin-Inf-PastPart` or `Fin-PastPart`, reflect the different readings of the verbform and enable following modules to use these verb forms according to the wider context, thereby removing the ambiguity. 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 subclauses of type subjunctive and subordinate. Although they are embedded into a larger structure they can independently and simply be recognized 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 subclauses) and verb fragments. The different types of subclauses are described very compactly as finite state expressions. Figure 10 shows a (simplified) BC-structure in feature matrix notation.

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

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

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.
>
> *Because the dog ate the beef which was put on the window sill after it had been prepared by the woman.*

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

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

In the correct reading the second subclause "... 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 includes basically punctuations and coordinations. The following Figure schematically describes the current coverage of the implemented MC-module (see Figure 6 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

5.1.2 Phrase recognition

After the topological structure of a sentence has been identified, each substring is passed to the FRAGMENT RECOGNIZER in order to determine the internal phrasal structure. Note that processing of a substring might still be partial in the sense that no complete structure need to be found (e.g., if we cannot combine sequences of nominal phrases to one larger unit). The FRAGMENT RECOGNIZER also uses finite state grammars in order to extract nominal and prepositional phrases, where the named entities recognized by the preprocessor are integrated into appropriate places (unplausible phrases are rejected by agreement checking; see (Neumann et al., 1997) for more details)). The phrasal recognizer currently only considers processing of simple, non-recursive structures (see Figure 7; here, *NP* and *PP* are used for denoting phrasal types). Note that because of the high degree of modularity of our shallow parsing architecture, it is very easy to exchange the currently domain-independent fragment recognizer with a domain-specific one, without effecting the domain-independent DC-PARSER.

5.2 Grammatical function recognition

After the fragment recognizer has expanded the corresponding phrasal strings (see Figure 7), 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 structure UDS. An UDS is a flat dependence-based structure of a sentence, where only upper bounds for attachment and scoping of modifiers are expressed (see Figure 11). In this example the PPs 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.[5]

An UDS 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 an UDS will represent the *longest matching subclause* together with a list of the non-recognized fragments. To retain the non-recognized fragments is important, because it renders possible that some domain specific inference rules will have access to this information even in the case it could not be linguistically analyzed.

The subcategorization lexicon Most of the linguistic information actually used in grammatical function recognition comes from a subcategorization lexicon. The lexicon we use in our implementation, which now counts more than 25.500 entries for German verbs, has been obtained by automatically converting parts of the information encoded in the SARDIC lexical database described in (Buchholz, 1996). In particular, the information conveyed by the verb subcategorization lexicon we use, includes subcategorization patterns, like arity, case assigned to nominal arguments, preposition/subconjunction form for other

[5]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. In some sense such an approach is domain-independent by just enumerating all possible readings. 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 some sense 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)).

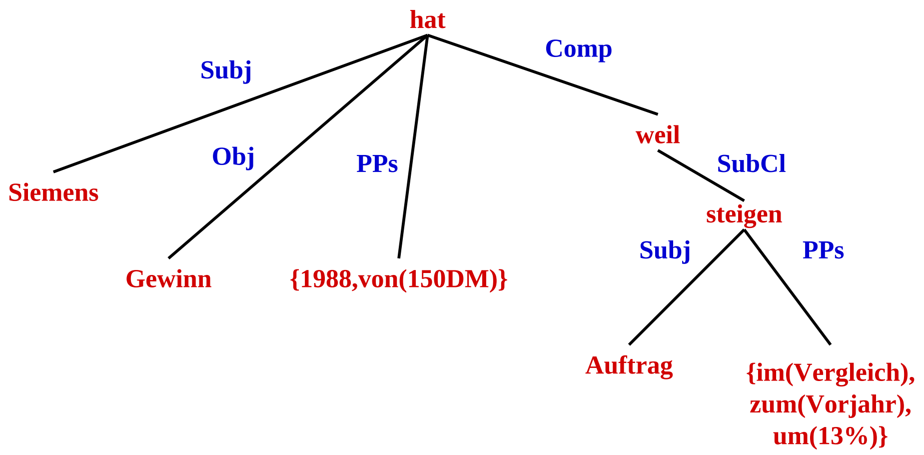

Figure 11: The underspecified dependence structure of the sentence: "Die Siemens GmbH hat 1988 einen Gewinn von 150 Millionen DM, weil die Aufträge im Vergleich zum Vorjahr um 13% gestiegen sind" (*The Siemens company had made a revenue of 150 million marks in 1988 since the orders increased by 13% compared to last year*).

classes of complements.

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"). For the elements, there is no ordering presupposed, 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 problem of possible ambiguity between argumental and adjunct attachment, as well as between the assignment of different possible roles to the same "modifier" (in the special sense explained above). Since all the linguistic processing in our system, thus including GFR, is supposed to be of shallow nature, it does not seem a reasonable solution to simply compute *all* the attachment possibilities that the subcategorization information of the verb allows in principle, and to possibly postpone any attempt at solving or reducing the ambiguity to a later stage of the analysis. Rather, what is needed is a kind of *heuristic* resolution of the attachment ambiguity, in order for the GFR to produce unambiguous and reasonably accurate results.

The strategy we implemented to deal with this important requirement is a rather minimalistic one: given a set of different subcategorization frames that the lexicon associates to a verb, the structure chosen as the final ("disambiguated") solution is the one corresponding to the *maximal subcategorization frame* available in the set, which is the frame mentioning the largest number of arguments that may be successfully applied to the input dependence 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 subclause). 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.

Adjuncts All the NPs, PPs and Subclauses that have not been previously recognized as arguments compatible with the selected frame are collected in *adjunct lists*. We distinguish two different sets of adjunct lists; the first one includes three *general adjunct lists*, namely modifier for NPs (NP-MODS), PPs (PP-MODS), and Subclauses (SC-MODS).

After grammatical function recognition, all the three above attributes are present (possibly empty) in every dependence tree. Besides them, a second set of optional *special adjunct lists* is available. The most remarkable feature of this list is that they are not defined as fixed , being rather triggered "on the fly" by the presence of some special information in adjunct phrases themselves: in the current implementation, this "special information" is encoded by means of the feature SUBTYPE. It is possible to define specialized finite state grammars (see 5.1.2), aimed at identifying restricted classes of phrases (for instance, temporal or locational expressions), which impose for that phrases to be treated as special adjuncts by means of the value assigned to SUBTYPE. In the GFR grammar, a set of correspondences between values of SUBTYPE and special adjunct lists can be declared; here is some example:

- {LOC-PP, LOC-NP, RANGE-LOC-PP} $\mapsto$ LOC-MODS
- {DATE-PP, DATE-NP} $\mapsto$ DATE-MODS

This means that, for example, if a modifier marked as LOC-NP is present in an dependence tree, then an attribute LOC-MODS is going to be created in the corresponding underspecified functional description after grammatical function recognition, whose value is a list containing that modifier; then, if another modifier belonging to the same class (say a RANGE-LOC-PP) is found, it is simply added to that list. The same mechanism applies for each class of SUBTYPE values considered by GFR.

5.3 Information access

It is important that the system can store each partial result on each level of processing in order to make sure that as much as contextual information as possible is available on each level of processing. We will call the knowledge pool that maintains all partial results computed by the shallow text processor system a *text chart*. Each component outputs the

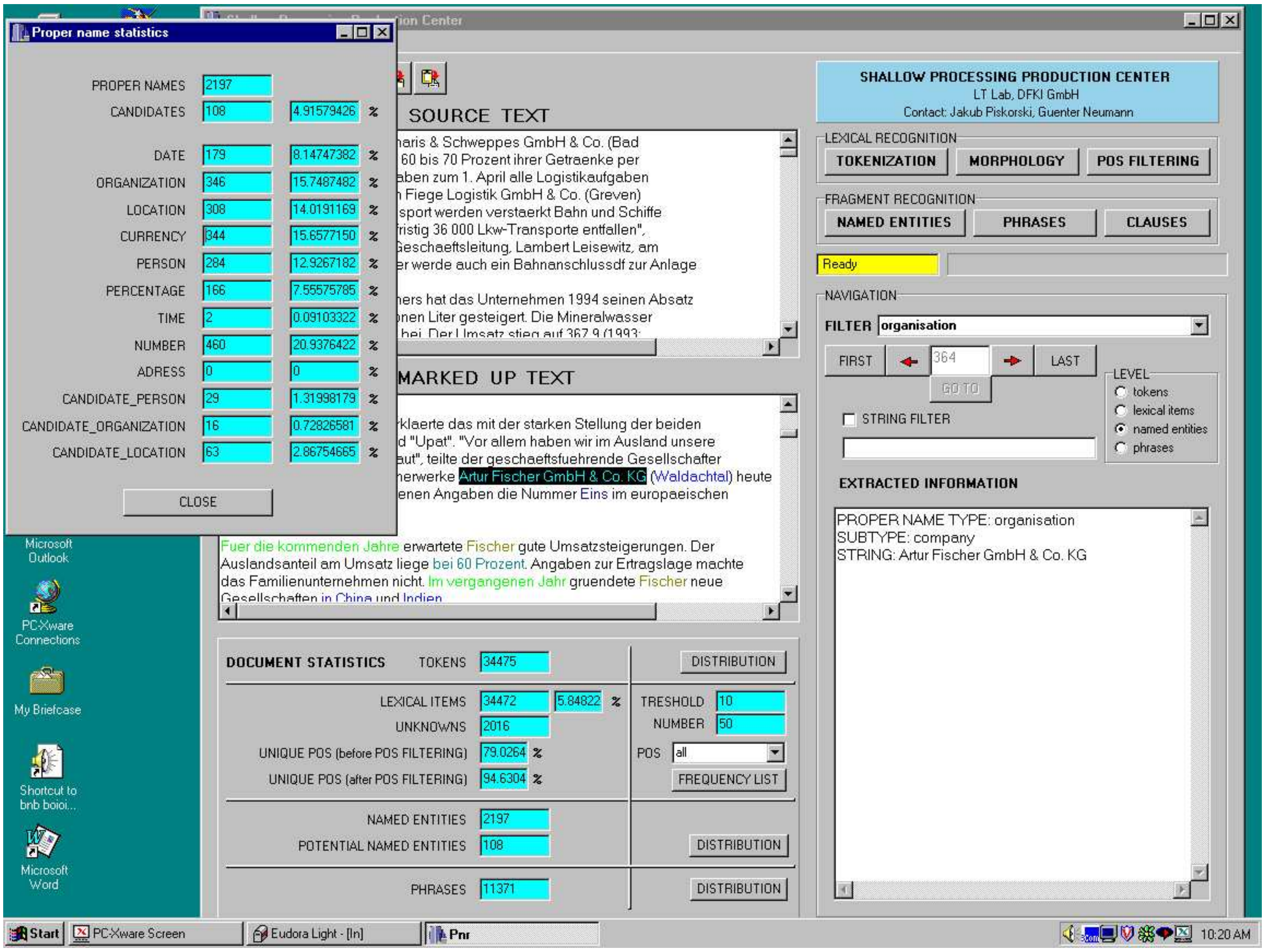

Figure 12: A screen dump of the system in action.

resulting structures uniformly as feature value structures, together with their type and the corresponding start and end positions of the spanned input expressions. We call these output structures *text items*.

The different kind of index information computed by the individual components (e.g., text position, reduced word form, category, phrase) allow an uniform flexible and efficient access to each individual partial result. This will avoid unnecessary re-computations and on the other hand, it provides rich contextual information in case of disambiguation or the handling of unknown constructions. Furthermore, it provides the basis for the automatic computation of hyperlinks of text fragments and their corresponding internal linguistic information. This highly-linked data structures provide for the internal representation for navigating through the space of all extracted information of each processing level. For example Figure 12 shows a screen dump of the current GUI of SPPC. 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", "organisaton", "date" etc). Then she can browse through the marked up text where relevant expressions are highlighted by different colours and their output structure is represented in readable form in an additional window (called "extracted information"). The user can also constraint the navigation through query filters (like "only visit organizations containing the substring software" or "show all compound nouns containing the suffix "-ung").

In order to support automatic knowledge acquisition performed by some external tools (e.g., text classification, text mining) we collect type-specific frequency information on all levels of computation, i.e., distribution of token classes, frequency list of word forms according to part-of-speech (e.g., all verbs, all nouns, all compounds), distribution of named entities and phrases. Additionally, the system also provides for an 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 different applications demands.

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 197116 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 case of compounds we measured whether a morpho-syntactical correct segmentation was determined ignoring, however, whether the segmentation was also the semantically most plausible one.[6] 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% have been disambiguated with a precision of 96.36%. Concerning named entities we only considered organization, person names and locations because these are the more difficult one and because only for these the dynamic lexicon is automatically created. 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 well-known company names. Of course, if we would have used a quite larger list (e.g., from some standard databases) than the recall (as well as the precision) would have been much higher. In case of fragment recognition we only considered simple phrase structure including coordinated NPs and NPs whose head is a named entity but ignoring attachment and embedded subclauses.

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 also was created after the DC-PARSER

[6]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 13: Correctness criteria used during evaluation.

and all grammars were finally implemented. Table 1 shows the result of the evaluations (the F-measure was computed with β=1). We used the correctness criteria as defined in Figure 13.

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 properly handle the ambiguities of some word forms being either a separated verb prefix or adverb.

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

Run-time behaviour The evaluation of the DC-PARSER has been performed with the LISP-based version of SMES (cf. (Neumann et al., 1997)) by replacing the original bidirectional shallow bottom-up parsing module with the DC-PARSER. The average run-time per sentence (average length 26 words) is 0.57 sec. A C++-version is nearly finished missing only the re-implementation of the base and main clause recognition phases. 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 11574 named entities and 67653 phrases. Since this is an increase in speed-up by a factor > 20 compared to the LISP-version, we expect the final C++-version to be able to process 75-100 sentences per second.

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 of 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 for an un-annotated test corpus how often his parser finds at least one result (where he reports 85.7% "coverage" of a test corpus of 72.000 sentences) disregarding to measure 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 punctuations, 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 such that a deep language processor might be called for those structures recognized by the shallow processor as being of great importance. Consider the case of (really) complex nominal phrases. In 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 belongs together. This problem is even more complex if we consider free word languages like German. However, taking advantage of SPPC' *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). Thus seen the shallow parser is used as an efficient preprocessor for dividing a sentence in 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.

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.

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.

Daciuk, J., R. E. Watson, and B. W. Watson. 1998. Incremental construction of acyclic finite-state automata and transducers. In *Finite State Methods in Natural Language Processing*, Bilkent University, Ankara, Turkey.

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. 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.

Kartunen, L., J-P. Chanod, G. Grefenstette, and A. Schiller. 1996. Regular expressions for language engineering. *Natural Language Engineering*, 2:2:305–328.

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 on Machine Learning for Information Filtering*, Stockholm, Sweden.

van Noord, G. 1998. Fsa6 - manual. Technical report, http://odur.let.rug.nl/ vannoord/Fsa/Manual/.

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.

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, page (to appear), 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