An Intelligent Text Extraction and Navigation System

Jakub Piskorski
DFKI, Saarbrücken
piskorsk@dfki.de

Günter Neumann
DFKI, Saarbrücken
neumann@dfki.de

November 5, 1999

Abstract

We present SPPC, a high-performance system for intelligent text extraction and navigation from German free text documents. SPPC consists of a set of domain-independent shallow core components which are realized by means of cascaded weighted finite state machines and generic dynamic tries. All extracted information is represented uniformly in one data structure (called the text chart) in a highly compact and linked form in order to support indexing and navigation through the set of solutions.

German text processing includes (among others) compound processing, high performance named entity recognition and chunk parsing based on a divide-and-conquer strategy. SPPC has a good performance (4380 words per second on standard PC environments) and high linguistic coverage.

1 Introduction

The information society will gradually provide its members with nearly unlimited access to all sorts of information. Without highly effective sophisticated information extraction applications for dealing with this wealth of information, the human user will not be able to utilise more than a tiny fraction of the immense potential the new technology offers.

The bulk of information which companies, administrations and private citizens have to deal with is in written language. Therefore one important application area for language technology is classification, indexing and retrieval in very large collections of short texts (e.g., news stories from press agencies, e-mail messages, abstracts) and large collections of long texts (e.g., on-line manuals, on-line business and financial reports, on-line journal articles). Here, advanced language technologies can be used to extract fine-grained information for supporting more accurate indexing and retrieval. Furthermore, the growing amount of on-line text and the growing population of users who want to extract an ever-widening diversity of types of information, requires easy portability of information extraction systems to new applications which can be done by trained non-specialists on the basis of a fast development cycle. Hence, ease of re-usability and customatization of IE core technology are critical for rapidly porting systems to new domains [Grishman and Sundheim, 1996; Cowie and Lehnert, 1996], as well as for an easy and intuitive navigation through the space of extracted information.

From our perspective, natural language analysis is a "step-wise process of normalization". 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 has *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

$\langle head = für, comp = \langle head = wirtschaft, quant = def, mod = deutsch\rangle\rangle$.

One of the main advantage of following a dependency 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 dependency approaches especially for its use in shallow text analysis (e.g., [Grinberg *et al.*, 1995; Oflazer, 1999]).[1]

In this paper we will describe SPPC an advanced system for intelligent text extraction and navigation. 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,

[1]From our current point of view, 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.

- represent all extracted information (together with its frequency) in one data structure as compactly as possible in order to support navigation through 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*.

2 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 most of the relevant local phenomena in the empirical study of language can be easily 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 centre 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 the section.

2.1 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 for which each transition has an output label in addition to the more familiar input label. For instance, the finite-state transducer in figure 1 represents a contextual rule for part-of-speech desambiguation: change tag of wordform 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 *et*

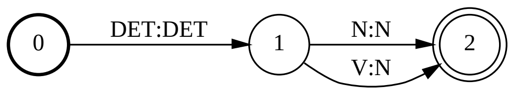

Figure 1: A simple FST

al., 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 coresponding algorithms for automata. Besides that the closure properties of transducers are much weaker than those of automata.

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 aplication. 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 *et al.*, 1992], may be chosen in order to determinize 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 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 *et al.*, 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 *et al.*, 1998] is provided. Besides that, we modified the general algorithm for removing epsilon-moves in [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.

2.2 Generic Dynamic Tries

The second crucial core tool is the generic dynamic trie, a parametrized 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 *et al.*, 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 2 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 (as done in our system). Furthermore, a variety of other more complex functions relevant for linguistic processing are available, including robust 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 German compounds and hyphen coordination in German (see sec. 3.1).

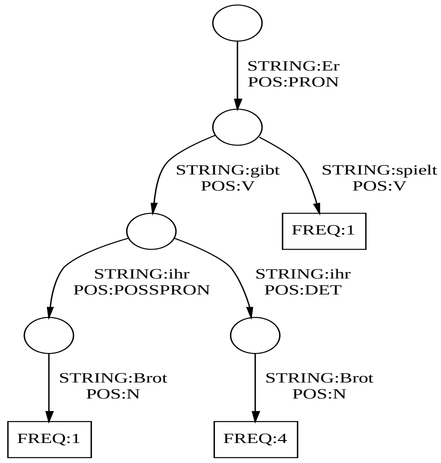

Figure 2: Example of a generic dynamic trie. The following pathes are encoded: (1) "Er(PRON) gibt(V) ihr(DET) Brot(N)." (*He gives her bread.*), (2) "Er(PRON) gibt(V) ihr(POSSPRON) Brot(N).", and "(3) Er(PRON) spielt(V)." (*He plays.*)

3 System overview

3.1 Architecture

In this section we will introduce the architecture of SPPC shown in figure 3. 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 primarily levels of processing, the word level and the phrase level. Both are subdivided into several components. First, the TEXT TOKENIZER maps sequences of characters into greater units, usually called tokens. Each token, which is identified as a potential wordform is then lexically processed by the LEXICAL PROCESSOR. Lexical processing includes retrieval of

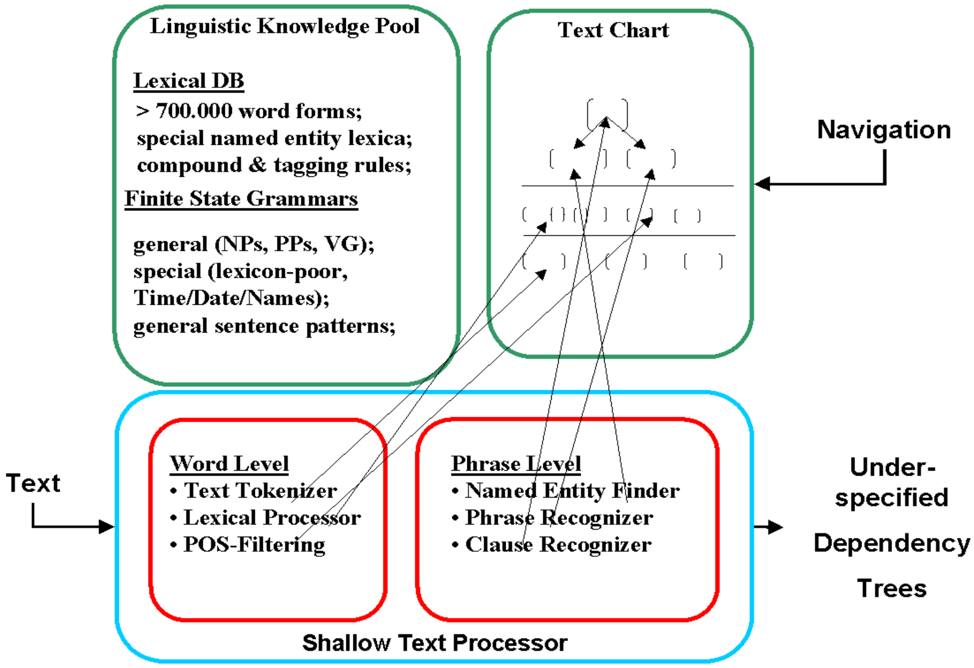

Figure 3: The blueprint of the system architecture. Figure 5 shows more details of the text chart.

lexical information (on the basis of a word form lexicon), on-line recognition of compounds and hyphen coordination. Afterwards the control is passed to POS FILTER, which performs word-based disambiguation, which is the final step of processing on the word level. The overall task of the phrase-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 phrase level processing in STP is divided into three modules. First the NAMED ENTITY FINDER identifies specialized expressions for time, date and several name expressions. Secondly PHRASE RECOGNIZER identifies general nominal phrases, prepositional phrases and verb groups. The structure of potential phrasal fragments is defined using finite-state devices. In the final third step, CLAUSE RECOGNIZER analyzes the dependency-based structure of the fragments of each sentence using a set of specific patterns, similarly expressed by means of finite-state devices.

Text Tokenizer Each text is preprocessed by the TEXT TOKENIZER, which performs two tasks. Firstly a text scanner maps sequences of consecutive characters into word-like units, usually called tokens. In a second step a token classifier identifies the type of each token (e.g., two digit number, decimal number, lower case word, word begining with capital). The token classifier is based on one WFSA, which is build up by applying union operation on WFSA's representing each token type. Hence the token classifier can be easily updated or extended by introducing new token classes without affecting the overall performance of the tokenizer. The output of the TEXT TOKENIZER is a list of so called token items represented as triples of the form $\langle start, end, type \rangle$, where *start* and *end* are pointers to the beginning and end of the token and *type* represents the type of the token. Unlike conventional tokenizers we do not recognize dates, time expressions etc. at this stage. We rather use generic (based on the syntax) token classes in such case. The string "13:15" is classified as number-dot compositum since it does not necessary has to be a time expression (could be also a result of a volleyball game). Similarly "1.3.96" would be classified as number-dot compositum. Since the TEXT TOKENIZER handles only single words disregarding the context, the recognition of date and time expressions is postponed and done at a higher stage by the NAMED ENTITY FINDER. However there exist some strings, which have an unambiguous reading and in such cases we use specific token classes (e.g., e-mail and URL adresses). Complex token classes include: abbreviation (on the basis of a set of known german abbreviations, ca. 600), candidate for abbreviation, complex compositum (e.g., "AT&T- Chief") and a variety of classes for representing words containing dashes (the word “d‘Italia-Chefs-” is classified as complex compositum word begining with lower case letter and ending with a dash) are provided, where the latter ones are relevant for hyphen coordination. Overall there are currently 51 token classes and it turned out that such variety simplifies processing on higher stages.

Lexical Processor Each token, which is identified as a potential wordform is lexically processed by the LEXICAL PROCESSOR. Lexical processing includes retrieval of lexical information and recognition of compounds and hyphen coordination. The sole storage device for full form lexicon used in SPPC is a generic dynamic trie. Hence the retrieval of lexical information is reduced to simple trie traversal. Currently over 700.000 German full-form words are available. Each word form is represented as a list of triples (each triple represents one reading) of the form $\langle stem, infl, pos \rangle$, where *stem* is a string representing the stem of the word form, *infl* is the inflectional information and *pos* is the part of speech. The inflectional information is expressed in disjunctive normal form. There are 11 inflectional features, among others including: *tense*, $form$, *person*, *gender*, *number* and *case*. In case a wordform misses one of the features (e.g., a noun form has no *tense* feature) the missing feature is returned with the special value NO. Here follows the result of LEXICAL PROCESSOR for the word “wagen” (two readings: *to dare* and *a car*):

stem	wag	
pos	v	
infl	(tense: pres, person: anrede, number: sg)	(tense: pres, person: 1, number: pl)
	(tense: pres, person: 3, number: pl)	(tense: pres, person: anrede, number: pl)
	(tense: subjunct-1, person: anrede, number: sg)	(tense: subjunct-1, person: 1, number: pl)
	(tense: subjunct-1, person: 3, number: pl)	(tense: subjunct-1, person: anrede, number: pl)
	(form: infin)	(form: imp, person: anrede)
stem	wagen	
pos	n	
infl	(gender: m, case: nom, number: sg)	(gender: m, case: dat, number: sg)
	(gender: m, case: akk, number: sg)	(gender: m, case: nom, number: pl)
	(gender: m, case: gen, number: pl)	(gender: m, case: dat, number: pl)
	(gender: m, case: akk, number: pl)	

The output of the LEXICAL PROCESSOR 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 boolean attribute, which is set to true in case word is a compound,
- *pref* is the prefered reading of the word, which is computed later by POS FILTER for ambiguous words

Compund analysis Each token not recognized as a valid wordform 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). Additionaly a list of valid compound prefixes is used since there are compounds in German beginning with a prefix, which is not necesarly a valid wordform (e.g., "Multiagenten" (multi-agents) is decomposed into "Multi" + "Agenten", where "Multi" is not a valid wordform). The capability of efficiently processing compounds is crucial since compounding is a very productive process of the German language. 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 necessarly 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 easily resolved to "Lederbranche", "Glasbranche" and "Holzbranche"

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

    The word "Verkauf: is a valid wordform, 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 If we consider the lexical analysis of words in isolation (which is the usual case), then each word is potentially ambiguous. In order to decrease the set of possible candidates for the following modules local and very efficient desambiguation strategies are applied in order to filter out unplausible readings. This is done by applying POS-taggers as well as by applying case-sensitive rules. Generally, only nouns (and proper names) are written in standard German with an capitalized initial letter (e.g., "der Wagen" - *the car* vs. "wir wagen" - *we venture*). 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. The task of a POS-FILTER is to compute the prefered reading (part-of-speach) of a current word in its current context using case-sensitive and contextual rules based on part-of-speach information (e.g., change tag of wordform from noun or verb to noun if the previous word is determiner). The latter group of rules is a combination of manually constructed rules and rules determined by Brill's tagger [Brill, 1993]. It has been shown that this set of filter rules can be represented as a single finite-state transducer [Roche and Schabes, 1995]. We will briefly describe the construction procedure. First, the application of each rule is encoded as nondeterministic finite-state transducer (see figure 1). The next step consists of turning the transducers produced in the preceding step into transducers that operate globally on the input in one pass by applying the local extension operation. In the third step all transducers are combined into one single transducer (composition of transducers) and finally this transducer is transformed into an equivalent minimal deterministic transducer.

Named Entity Finder The task of the NAMED ENTITY FINDER is the identification of entities (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. Similarly to tokenizer extending the set of such patterns does not affect the overall performance of the recognition-part since all WFSA's are merged to a single WFSA. 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

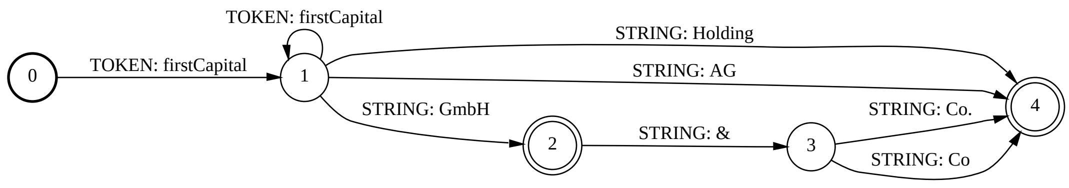

Figure 4: The automaton representing the pattern (here a simplified version) for recognition of company names

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 current lexical item is of the form s
2. *stem*: s, holds if: (1) current lexical item has a prefered reading with stem s or (2) current lexical item does not have prefered reading, but at least one reading 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 4 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.

For retrieving named-entities we use an additional lexicon for geographical names, first names (persons) and company names, which is compiled as WFSA. Each string mapped by lexical item beginning with an upper-case letter is searched in this lexicon and in case it is found it's token type will be temporarily changed to special token class (e.g., city, first name, company name). For instance the word "Berlin" is classified by the tokenizer as first-capital word, but since it would be found in the special named-entity lexicon mentioned above it's token class would be set to city.

Many of the named entities 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 for storing named-entities, which may appear later in the text without designators (organisations, persons). For example after recognizing the company name "Apolinaris & Schweepes GmbH & Co." we would save the phrase "Apolinaris & Schweepes" in this dynamic lexicon. The succeeding occurences of the latter phrase without a designator would be then treated as a occurence of named-entity and a reference to the last occurence the same phrase with the designator would be set (heuristic). Unfortunately one problem arises, the named-entities stored in the dynamic lexicon, consisting only of one word may be also a valid word form. At this stage it is usually impossible to decide for such words wether they are named-entities or not. Hence such words 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 named-entity "Braun AG", the following occurence of "Braun" would be recognized as organization candidate since "braun" is valid word form (*brown*). Such ambiguities can be resolved on higher level of processing or by applying some heuristics.

Phrase Recognizer The PHRASE RECOGNIZER is responsible for the extraction of nominal and prepositional phrases and verb groups. This is done in a very similar way to recognizing named entities. The fragment extraction patterns are expressed as WFSA's (mainly based on the part-of-speech and type of the named-entity information) and appropriate constraints (e.g., for checking agreement or morphological features of the verbs) are used for validation of candidate phrases. The PHRASE RECOGNIZER takes as input both the list of lexical items and the list of named-entities computed on preceding levels. The output of this module is a list of phrase items represented as 2-tuple of the form $\langle type, comps \rangle$, where *type* is the type of the phrase and *comps* is a list of the components of the phrase.

The information gathered on all stages of processing (including phrase recognizer) for the sequence "an die Frankfurter Boerse" (*for the Frankfurt Stock Exchange Market*) is shown in figure 5 (in a simplified form).

The phrasal grammars only consider continous substrings and try to recognize the longest matching substring. Since we do not recognize discontinous phrases at that level, recognition of verb groups is only partial as long as a verb group is splitted into a left and right part, as is common in German. For example in passive sentences the auxillar and the corresponding main verb are splitted into two parts such that other phrases are spliced into the splitting point:

"Gestern [ist] der Linguist vom neuen Manager [entlassen worden]."
Yesterday, the linguist [has] [beeb fired] by the new manager.

Note that it is possible to configure the phrase grammars in different ways, e.g., to run verb groups independently from the other phrases, so that the phrase recognizer can be called as subroutine on different places during chunk parsing. Actually we make use of this sort of flexibility in the clause recognizer.

Clause Recognizer The main task of CLAUSE RECOGNIZER is to construct a dependency relation between all recognized phrases. Basically this includes combination of coresponding left and right verb groups, identification of subclauses, and collection of all phrases which are dominated by the found verb groups. Rules for clause recognition are expressed as WFST's. In most well-known shallow text processing systems (cf. [Sundheim, 1995] and [Extraction, 1998]) cascaded chunk parsers are used which perform clause recognition *after* phrase recognition following a bottom-up style as described in [Abney, 1996]. In the SMES – the predeccesor system of SPPC[Neumann *et al.*, 1997] – we also used a similar bottom-up strategy for processing of German texts. However, the main problem we experienced using the bottom-up strategy was insufficient robustness: because the parser depends on the lower phrasal recognisers, 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. Consider for example the following case:

"[Die vom Bundesgerichtshof und den Wettbewerbshütern als Verstoß gegen das Kartellverbot gegeisselte zentrale TV-Vermarktung] ist gängige Praxis."

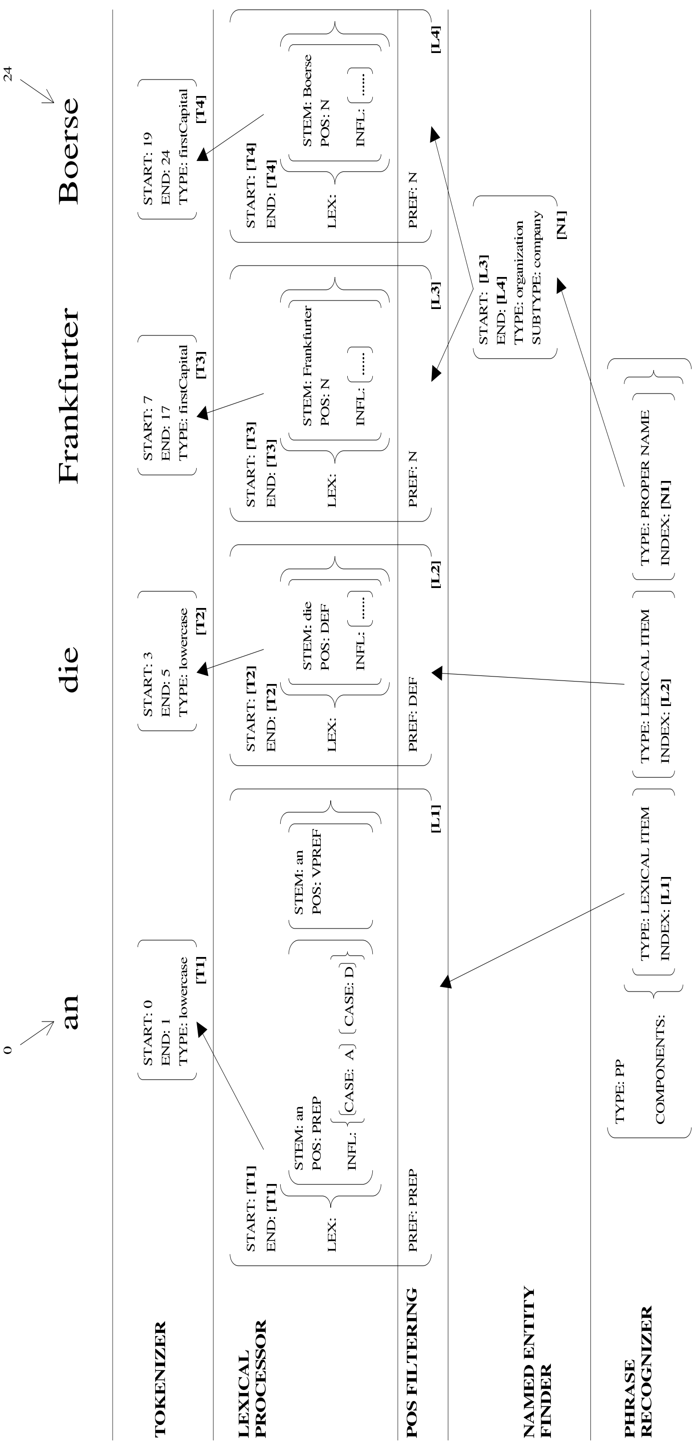

Figure 5: Example of the text chart for the expression “an die Frankfurter Börse” (*for the Frankfurt Stock Exchange Market*)

Central television marketing censured by the German Federal High Court and the guards against unfair competition as an act of contempt the central ban is common praxis.
has a quite complex NP (marked by [ ]). During free text processing it might be not possible (or even desirable) to recognize such a phrase completely. However, this can lead to interpolations wrt. recognition of the 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 developed an *top-down/bottom-up* chunk recognizer: 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. In a second phase the phrasal grammars (nominal and prepositional phrases, proper names) are applied to the contents of the different fields of the main and subordinate clauses. The following example shows the result of the first step:

"[$_{coord}$ [$_{core}$ Diese Angaben konnte der Bundesgrenzschutz aber nicht bestätigen], [$_{core}$ 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.
This approach offers two main advantages:

- improved robustness, because parsing of the sentence structure is based only on some reliable 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.

First experiments showed very promising results using a blind test of 400 sentences containing 6300 word forms from press releases selected messages of different kind (political, economy and others) from DPA and Reuters from 2. and 5. July 1999. Here we obtained an f-mesaure of 87.14%.

3.2 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 resulting structures uniformly as feature value structures, together with its type and the corresponding start and end positions of the spanned input expressions. We call these output structures *text items* (see figure 5).

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 6 shows a screen dump of the current GUI of SPPC.

The shallow text processor thus described can "only" compute linguistic information in the sense that it does not tell us what information is relevant for our domain modelling. However, we assume that the quantity and distribution of each information unit on every level of processing will tell us something about its relevance.

3.2.1 "Pure" Statistics

The computed structures on each level can uniformily be treated as a stream of units. For example, the text tokenizer computes a stream of tokens, the lexicon component a stream of morphological readings, and the phrasal recognizer returns a stream of phrases. Each unit is represented as a text item which encodes its start and end position in the input text, its type (e.g., token class, part-of-speech or phrasal type), and a representation of its output structure (see also previous section).

Now, by "pure" statistical information we mean the collection of frequency and distribution information of extracted text fragments without considering its internal structure, i.e., we only consider the text fragment's string, its text position and its type (e.g., noun (N), adjective (Adj), proper name (PN), nominal phrase (NP) or verb group (VG) etc.). Using this information we can easily compute the relative frequency of certain instances of each type (a certain noun or nominal phrase). The positional information is mainly used in order to compute:

- N-grams, or collocations: A collocation is a "recurrent combination of words that co-occur more often than expected by chance and that correspond to arbitrary word

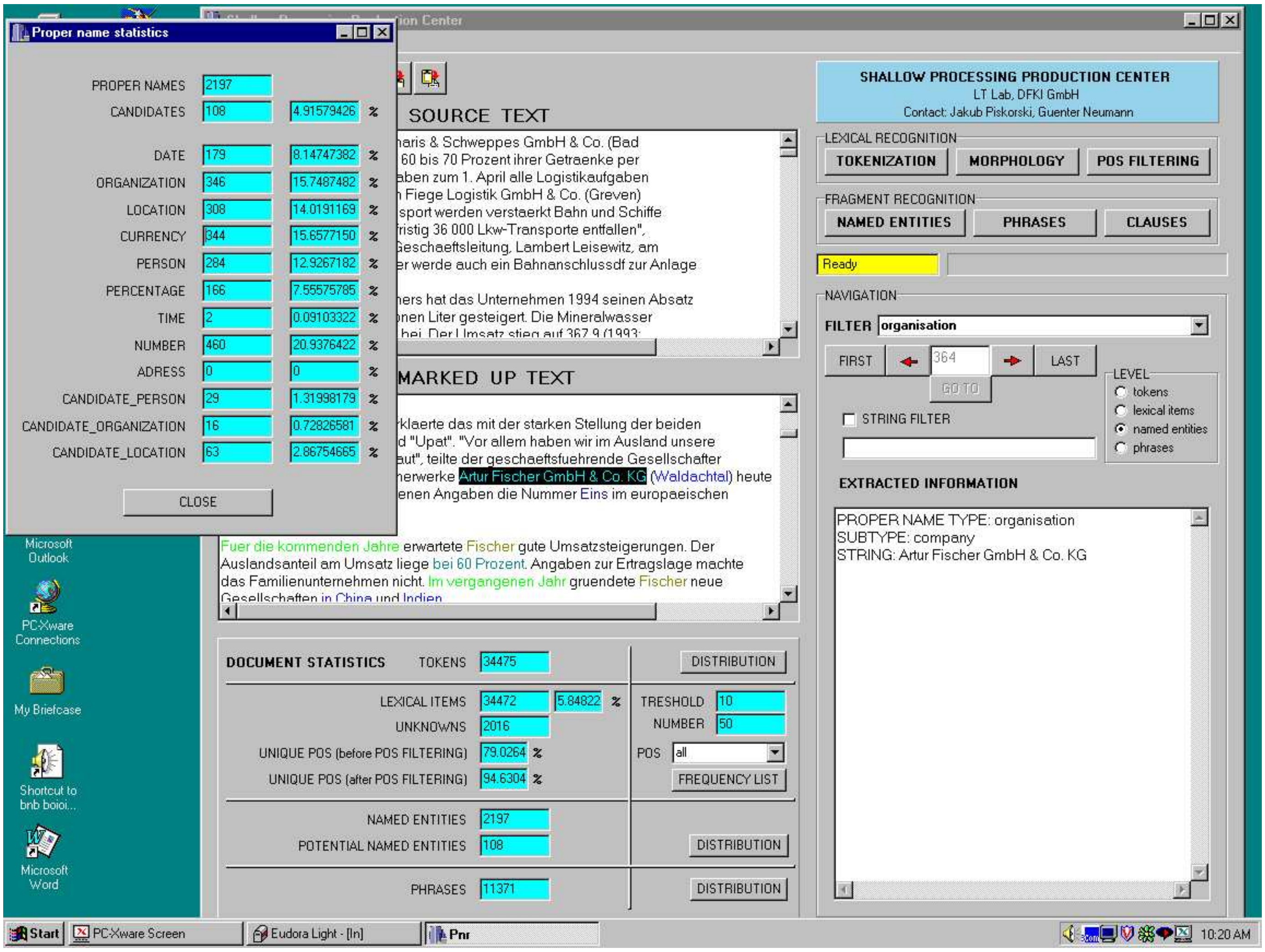

Figure 6: A screen dump of the system in action: The user can choose which level to inspect, as well as the type of expressions (e.g., in case of named entities, she can select "all", "organisaton", "date" etc). Then she can browse through the marked up text where relevant expressions are highlighted 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 containg the substring software" or "show all compound nouns containing the suffix "-ung"). On each level, the user can inspect the frequency and distribution of the found elements, and can choose to redirect all extracted information to a set of corresponding output streams, e.g., simple ASCII or XML format.

usages" [Smadja, 1993] and they typically are found by computing the probability of bigrams and trigrams (see also [Charniak, 1993]).

- the history of individual words and its life-cycle, i.e., to get information when a certain word or phrase was introduced and how often it has been used in a certain context. The assumption here is that a text fragment with a low frequency (relative to the complete document) will have a high frequency for some time after it has been introduced. This information could be used as a basis for inducing domain-specific terms.

Pure statistical approaches will also be used for unsupervised classification. For example, AutoClass [Cheeseman and Stutz, 1996] is a unsupervised Bayesian classification system that seeks a maximum posterior probability classification. The inputs consist of a database of attribute vectors, and a class model. The system finds the set of classes that is maximally probable with respect to the data and model. The output is a set of class descriptions, and partial membership of the cases in the classes. This approach seems very promising for term extraction, in case linguistic structure computed by the shallow text processor is used for defining the attribute set and class model.

4 System implementation and performance

The system has fully been implemented and can process text documents up to several megabytes very efficiently and robustly. It 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].

The current version of the DFKI FSM Tool was implemented in C++ and is available as an object-oriented C++ library. Most of the algorithms work well with arbitrary semirings on real numbers, where the general semiring for real numbers is an abstract datatype. The generic dynamic tries were implemented as a C++ template. Similarly the linguistic modules of the SPPC system were implemented in C++ and can be used separately. The GUI for SPPC was implemented in Borland C++ for Windows NT operating system, whereas the code for all other tools mentioned above is portable across different platforms (UNIX , Windows NT/98).

The system has a very good run-time behaviour: processing of a German text document (a collection of buisness news articles from the "Wirtschaftswoche") of 197118 tokens (1.26 MB) needs 45 seconds on a PentiumII, 266 MHz, 128 RAM, which corresponds to 4380 words per second. This includes recognition of 10553 named entities and 64666 phrases. Only 6.11% of the lexical items where unknown. Before POS-filtering, 79.35 % of the lexical items had unique pos, and after POS-filtering 94.37 % got a unique pos assignment. This means that from the approx. 20% ambiguous words about 75% have been disambiguated with an accurracy of 97.9%. Concerning named entities (considering organization, person names and locations) we obtained a precision of 95.77% and a recall of 85% on a 200KB subset of the mentioned text (90.1% F-measure).

An evaluation tool that determines the quality of the NP-grammar has been developed to automatically evaluate the coverage of the NP-grammar using the reference corpus (see above). It is important to notice that the NP-grammar was developed without seeing the corpus. Moreover, the intended domain was appointment scheduling. The following matching criteria where used: exact match, start point match, end point match, and a weighted matching score depending on the size of the overlapping areas. For our evaluation purposes only the bracketing of simple nominal phrases was used. The following table summarizes the result:

Criteria:	Precision:	Recall:
Exact-match	67.5 %	69.7 %
Start-point-match	71.9 %	74.2 %
End-point-Match	86.6 %	89.4 %

The average of these values then represent the global precision value (99.5 %) and the global recall value (78.6 %).

The verb-group grammar was independently evaluated on a blind test set of 400 sentences (6300 words) achieving an F-measure of 98.59%. With the same test set, the recognition of the topological sentence structure achieved an F-mesaure of 87.14%.

5 Related work

SPPC is mostly related to the shallow text processor of *smes*, an IE-core system for real world German text processing [Neumann *et al.*, 1997]. The main differences are the use of advanced WFSM tools, the name entity finder, and the divide-and-conquer clause recognizer. To our knowledge, there are only very few other systems described which process free German texts. For example [Wauschkuhn, 1996] describes a tool for the partial analysis of German text documents. He presents a similar divide-and-conquer approch for parsing using a chart-based parser. However, it is difficult to compare its performance with our approach because 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). However, this result is not very informative because it does tell nothing about the accuraccy of the parser. In a similar test environment, our parser achieved a "coverage" of 94.25%. This might be the case because we are using more advanced lexical and phrasal components, e.g., compound and named entity processing. [Peh and Ting, 1996] also describe a divide-and-conquer approach following a statistical-based approach. On an annotated test corpus of 600 English sentences they report an accuracy of 85.1%.

6 Conclusion and future work

We have presented an advanced domain-independent shallow text extraction and navigation system for processing of real-world German text. The system is implemented based on advanced weigthed finite state machines and uses sophisticated linguistic knowledge sources. The system is very robust and efficient (at least from an application point of view) and has a good coverage. Currently we focus on the German language but have started to consider English as well, in order to support multi-lingual shallow text processing. Although the core functionality is the same, it has been shown that the general model has to be refined if a specific (class of) language has to be processed.

References

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

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

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

[Charniak, 1993] E. Charniak. *Statistical Language Learning.* MIT - Pres, 1993.

[Cheeseman and Stutz, 1996] P. Cheeseman and J. Stutz. *Bayesian Classification (AutoClass): Theory and Results*, chapter 6. AAAI Press/MIT Press, 1996.

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

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

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

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

[Extraction, 1998] SAIC Information Extraction, editor. *Proceedings of MUC-7*, http://www.muc.saic.com/, 1998.

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

[Grishman and Sundheim, 1996] R. Grishman and B. Sundheim. Message Understanding Conference – 6: A Brief History. In *Proceedings of the 16th International Conference on*

Computational Linguistics (COLING), pages 466–471, Kopenhagen, Denmark, Europe, 1996.

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

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

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

[Neumann and Schmeier, 1999] G. Neumann and S. Schmeier. 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, 1999.

[Neumann *et al.*, 1997] G. Neumann, R. Backofen, J. Baur, M. Becker, and C. Braun. 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 1997.

[Oflazer, 1999] K. Oflazer. Dependency parsing with an extended finite state approach. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, ACL-99*, Maryland, 1999.

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

[Piskorski, 1999] J. Piskorski. Dfki fsm toolkit. Technical report, DFKI - LT Lab, 1999.

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

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

[Smadja, 1993] F. Smadja. Retrieving collocaionts from text:xtract. *Computational linguistics*, 19(3), 1993.

[Staab *et al.*, 1999] S. Staab, 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. GETESS — searching the web exploiting german texts. In *CIA'99 — Proceedings of the 3rd Workshop on Cooperative Information Agents*, LNCS, page (to appear), Berlin, 1999. Springer.

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

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

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