Chapter 2
The MOTS Workbench

Manfred Stede and Heike Bieler

Abstract. Standardization of processing frameworks for text documents has been an important issue for language technology for quite some time. This paper states the motivation for one particular framework, the MOTS workbench, which has been under development at Potsdam University since 2005 for purposes of research and teaching. We describe the overall architecture, the analysis modules that have been integrated into the workbench, and the user interface. Finally, after five years of experiences with MOTS, we provide a critical evaluation of the design decisions that were taken and draw conclusions for future development.

2.1 Introduction and Overview

The development of general frameworks for quickly configuring natural language processing (NLP) applications started in the 1990s, with the release of GATE in 1996 being the most significant first milestone [6]. Nowadays, several of such frameworks are in successful use; we will provide a brief overview in Section 2.2. Our aim in this paper is to provide a description and critical review of our own framework, which has been implemented at Potsdam University since 2005. The MOTS (MOdular Text processing System) Workbench was devised as a framework[1] for integrating diverse NLP modules by means of a specific standoff XML

Manfred Stede · Heike Bieler
Applied Computational Linguistics, EB Cognitive Science, University of Potsdam, Karl-Liebknecht-Str. 24-25, D-14476 Golm, Germany
e-mail: `{stede,bieler}@uni-potsdam.de`

[1] We use the term ‘framework’ in the same sense as [5]: a software infrastructure that supports the effective re-use of services for a particular domain (here: language processing). In Computer Science, a more specific term emphasizing the integration of heterogeneous components is ‘middleware’. When graphical tools for configuring and testing systems are being added, the framework turns into a ‘development system’; MOTS provides a first step here but does not go all the way, as will become clear later.

format that serves as "interlingua" or "pivot" format to mediate between modules. An important goal in designing this format (called PAULA for 'Potsdamer Austauschformat für Linguistische Annotation') was to serve both the worlds of manual annotation and of automatic processing:

- Manual annotation, in our setting, primarily serves the needs of a large collaborative research center on information structure (IS), where data of very different kinds and in different languages was hand-annotated with different types of information that can support IS research. *Heterogeneity* thus is a key concept here. The different annotation needs are best suited by different existing tools, which can be used to label the same data from different viewpoints. The challenge then is to merge the output formats of the annotation tools into a single linguistic database, where the data can be queried across the layers in order to check for correlations of features. This setting is described in detail in [4]
- Automatic processing, the focus of the present paper, was conceived in the same way: analysis modules contribute separate layers of annotation to a text document, and – in addition – a module in charge of a "complex" task should be able to access the results of modules that have already solved a less complex task. Thus layers of analysis are being stacked upon each other, and each layer can be stored in the PAULA format.

The basic idea therefore was to realize a layer-based annotation of text data, where layers might either be independent or in their annotations refer to other layers, and where the origin of the layer (manual annotation or automatic analysis) would not matter. Our emphasis on an XML-based pivot format results on the one hand from the need to make the output of manual annotation tools (such as MMAX2, EXMARaLDA, RSTTool, ANNOTATE) inter-operable, and on the other hand from the desire to integrate automatic analysis tools written in arbitrary programming languages – at present, MOTS includes modules written in Java, C, C++, Lisp, Perl, and Python.

A common problem for integrative approaches of the kind described here is to ensure that all annotations refer correctly to the intended spans of text. Our approach is to submit each incoming document to a preprocessing phase where sentence splitting and tokenisation is performed, and the basic layout of the document represented in a light-weight XML "logical document structure" format (henceforth called 'LDOC'). The sentence and token boundaries are then enforced for all other modules – which can be difficult to do when integrating external modules that perform their own tokenization, thus generating the need for synchronization. One advantage of the standardized LDOC base format is that different types of documents (XML, HTML, plain text) can all be translated to LDOC during preprocessing, so that all subsequent analysis modules are not being bothered by idiosyncratic data formats.

The first task that MOTS was designed for was automatic text summarization. Our user interface thus was given the job to not only show all the intermediate analysis results (for debugging purposes) but also to allow for a flexible visualization of summaries (extracts) of different lengths. From the outset, we targeted a web-based UI, as development of the text summarizer involved external project partners who had to have quick access to our results. This decision was also supported (and proved very helpful) by the desire to use MOTS also for teaching purposes: In classes on NLP and text analysis, results can be shown and compared, and new modules implemented by students be integrated into the framework with relatively little effort. In this sense, we see MOTS as a ‘workbench’: a flexible environment for combining NLP modules and showing results. However, MOTS was not conceived as a framework for building real-time or even industry-standard applications – which is why we use the term ‘NLP workbench’ rather than ‘Language Engineering (LE) workbench’.

Having described the “starting position” of the MOTS development, we now turn first to a brief review of related approaches along with a comparison to MOTS (section 2.2). Afterwards, we provide a description of the PAULA XML format (section 2.3) and give overviews of the overall architecture of the processing pipeline (section 2.4), the analysis modules we have integrated so far (section 2.5), and the user interface (section 2.6). Finally, we mention some ongoing work on MOTS, and provide a critical evaluation of the design decisions in the light of several years of experience with the system (section 2.7).

2.2 Background: Natural Language Processing Frameworks

The central task of an NLP framework is to support an effective re-use of processing modules – usually called *components* when part of a framework – and possibly also of linguistic (data) resources, and their flexible re-combination to systems made up of a particular set of components. Thus, a component would be in charge of a particular subtask, and the composition of the subtasks to the overall application is being designed with the help of the framework, which ensures the inter-operability of components. One of the first major steps to the design of such frameworks for language processing was the TIPSTER architecture [13], conceived at the time primarily for the task of information extraction. In general, we can distinguish between frameworks aiming primarily at document processing and handling rather diverse types of information, and those focusing more specifically on linguistic analysis, emphasizing the role of unifying *formalisms* for representing information and performing computations. A recent example for the latter is ‘Heart of Gold’ [19]; our focus in this paper, however, is on the former group.

When devising GATE[2], [5] characterized the basic design decision as one between the ‘embedded’ and the ‘referential’ type of annotation (p. 98), where the former was represented by SGML “inline” markup in the document itself, and the latter by a database model where columns in the table represent annotations referring to sequences of characters in the source document – i.e., the TIPSTER approach. GATE adopted the database solution, mainly because it seemed to provide

[2] `http://gate.ac.uk`

more efficient access in comparison to the inherently sequential data storage using SGML, and because of the possibility to represent graph structures in the annotations. Nowadays, SGML inline markup has developed to XML standoff markup, where each layer of annotation resides in a separate file, the source text document remains unchanged, and annotations merely *refer* to it (borrowing from TIPSTER's idea of the database table). This allows for more flexibility in the annotations (e.g., it is possible to represent overlapping spans) and it also allows for representing graph structures; this has been suggested in the Linguistic Annotation Format (LAF, [15]) as well as in PAULA, and it is the approach pursued in MOTS, as will be explained in the next section.

Apart from this difference in basic design, MOTS obviously is a much more modest endeavour than GATE. For one thing, at the moment we provide only single-document processing, so there is no corpus management facility. For the time being, MOTS offers neither a systematic description of component behaviour nor a treatment of linguistic non-processing resources (e.g., lexicons), as it has been introduced in version 2 of GATE. More importantly, as will become clear in Section 2.5, a unified access to the analysis results of the components is only now becoming available in the form of a Java API; apart from that, components need to parse the standoff representations and extract the required data themselves.

Another important issue for NLP frameworks is their providing for distributed and/or parallel processing. Regarding distribution, the introduction of service oriented architectures and in particular web services was a very influential development in recent years, and it is beginning to play a more important role in language processing as well. The possibility of flexibly coupling services that do not have to be locally installed can speed up the prototyping of NLP applications significantly (but, on the other hand, it of course may slow down execution speed); notable examples are the web services provided by the project Deutscher Wortschatz at Leipzig University.[3] The MOTS approach of exchanging analysis results via standoff XML annotations makes the integration of external services quite straightforward – they can be wrapped in much the same way as modules running on the local machine.

Non-linear execution, on the other hand, is as of today not a standard feature in NLP frameworks. Various proposals had been made in the 1990s, among them the PVM-based ICE manager [1] in the Verbmobil project. Today, however, the well-known and widely-used frameworks LT-XML2[4] and GATE realize strictly sequential "pipe/filter" architectures. So does MOTS, and at the moment it gives the user only limited means to influence the processing chain: In contrast to, for example, the possibility of dynamically adding components and menu-based construction of processes in GATE, we merely allow for selecting the modules that are to take part in the process from a hard-wired list – the user can thus activate modules, while MOTS ensures the well-formedness of the resulting chain (keeping track of dependencies).

[3] `http://wortschatz.uni-leipzig.de/Webservices/`
[4] `http://www.ltg.ed.ac.uk/software/ltxml2`

Parallel processing *is* enabled in implementations of the Unstructured Information Management Architecture (UIMA)[5]. This is the most ambitious approach, in principle targeting not only text documents but other kinds of unstructured information as well (audio, video), with the basic mission to turn unstructured into structured information (e.g., database tables). Though sharing many similarities with GATE, UIMA is more explicitly oriented to scalable processing of vast amounts of data, hence to the professional construction of language engineering applications. In contrast to GATE, the basic unit of processing in UIMA is a general feature structure rather than an annotated text (which would be just one special kind of feature structure). Also, these structures are strongly typed, requiring very explicit descriptions of components' interfaces. This clearly facilitates sharing of components (a number of repositories already exist) but at the same time is not trivial and requires negotiation between parties interested in using the components, possibly for quite different purposes.

With its much more modest goals, the idea of MOTS is to provide a lean, easy-to-use framework for effective development and testing of modules for single-document processing, in the spirit of rapid prototyping. It is aimed explicitly at research and teaching and not at building applications, so there is more emphasis on simplicity than on performance.

2.3 The PAULA XML Format

The rationale behind our representation format PAULA[6] (a German acronym for 'Potsdam interchange format for linguistic annotation') is the integration of different annotation structures, whether resulting from manual or from automatic annotation. With respect to manual annotation, we provide conversion tools that map the output of the following tools to PAULA: *annotate* for syntax annotation; *Palinka*[7] and *MMAX2*[8] for discourse-level annotations such as co-reference; *EXMARaLDA*[9] for dialogue transcription and various layer-based annotations. The conversion scripts are publicly available via the PAULA webpage: Users can upload their data and annotations, and the data is converted automatically to PAULA. The mappings from the tool outputs to our format are defined such that they only transfer the annotations from one format into another without *interpreting* them or adding any kinds of information.

[5] `http://incubator.apache.org/uima/`
[6] See [7] and
`http://www.sfb632.uni-potsdam.de/projects/d1/paula/doc/`
[7] `http://clg.wlv.ac.uk/projects/PALinkA/`
[8] `http://mmax2.sourceforge.net`
[9] `http://exmaralda.org`

2.3.1 PAULA: Logical Structure

The conceptual structure of the PAULA format is represented by the PAULA Object Model (POM). It operates on a labeled directed acyclic graph. Similar to the NITE Object Model [12, NOM] and the GrAF data model [16], nodes correspond to annotated structures, and edges define relationships between independent nodes. Both nodes and edges are labeled, and generally, labels define the specifics of the annotation. Nodes refer to other nodes, or point to a stream of primary data.

Besides labels that define concrete annotation values, a specialized set of labels serves to indicate the *type* of an edge or a node. For a specific set of pre-defined edge labels, POM defines the semantics of the relation expressed by the corresponding edge. For instance, the *dominance* relation is characterized as a transitive, non-reflexive, antisymmetric relation, which requires that the primary data covered by the dominated node is covered by the dominating node as well. On the basis of these dominance relations, tree structures can be represented, e.g. syntactic trees.

Another pre-defined edge type is *reference*, a non-reflexive, antisymmetric relation. Reference relations may occur with different annotation-specific labels. Reference relations with the same label, e.g. ‘anaphoric_link’, or ‘dependency_link’ are also transitive. Reference relations serve to express, for instance, dependency trees, coreference relations, or alignment of text spans in multilingual data.

The PAULA Object Model differs from related proposals, e.g. GrAF, in the definition of explicit semantics for certain edge types. The specifications of the dominance relation are comparable to the NITE Object Model, but while NOM focuses on hierarchical annotation, POM also formulates the semantics of pointing relations.

On the basis of this general object model, annotation-specific data models are then defined with reference to POM.

2.3.2 PAULA: Physical Structure

The elements of the PAULA representation format along with their corresponding POM entities are given in Table 2.1. For illustration, Figure 2.1 shows a sample annotation data set, as it is distributed across different layers (and files). The token layer, via xlink, identifies sequences of characters in the primary text files, and thereby provides the ultimate reference objects (in POM, terminal nodes) for other levels of annotation. We call such objects ‘markables’, and hence the token layer is of type ‘mark list’. The POS (part of speech) layer, in contrast, does not define new markables but merely provides labels to existing ones; its type therefore is ‘feat list’. The sentence layer, not surprisingly, provides objects that are sequences of tokens; these can then also be given attributes, such as the term relevance values in our example (on the Term layer). Paragraphs are then represented as sequences of sentences in the Div layer (see Section 2.4.1) and can in turn receive attributes, as by the Zone layer in the example (see Section 2.5).

Table 2.1 Predefined structure elements in the PAULA Object Model.

PAULA element	POM entity
`tok`(en)	terminal node
`mark`(able)	non-terminal node (containing *references* to nodes)
`struct`(ure)	non-terminal node (containing *dominance relations* to nodes)
`rel`(ation)	within `struct`: *dominance*, otherwise *reference* relation
`feat`(ure)	annotation label
`multiFeat`(ure)	bundles of annotation labels

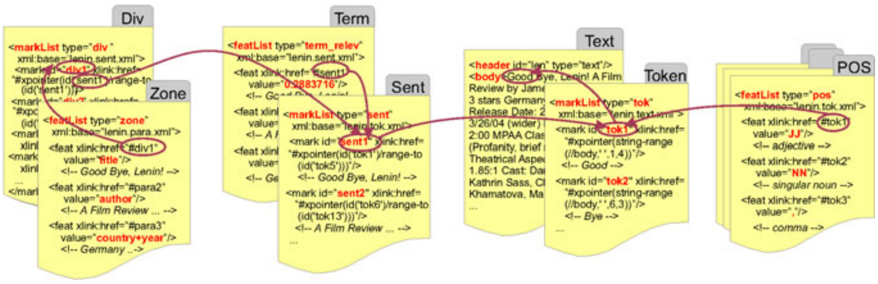


Fig. 2.1 Illustration of PAULA standoff annotation.

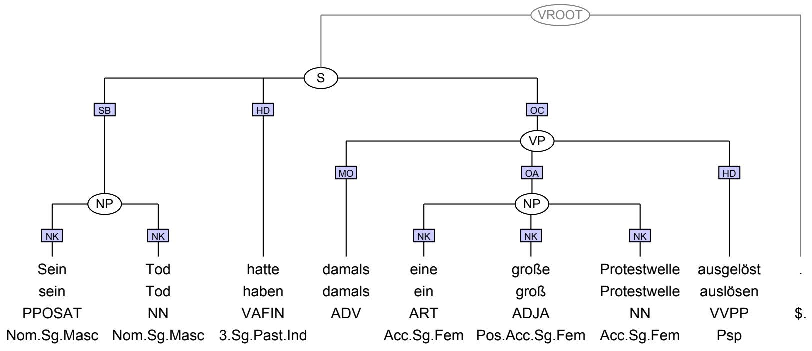

VROOT
S
SB
HD
OC
VP
MO
OA
HD
NP
NK
NK
NP
NK
NK
NK
Sein sein PPOSAT Nom.Sg.Masc
Tod Tod NN Nom.Sg.Masc
hatte haben VAFIN 3.Sg.Past.Ind
damals damals ADV
eine ein ART Acc.Sg.Fem
große groß ADJA Pos.Acc.Sg.Fem
Protestwelle Protestwelle NN Acc.Sg.Fem
ausgelöst auslösen VVPP Psp
. $.

Fig. 2.2 Example of a TIGER tree.

For the encoding of hierarchical structures, including labeled edges, PAULA provides the specific elements `struct` and `rel`. Like markables, a `struct` element represents a node in POM, but in this case a node which is the parent node of a *dominance* relation. The dominance relation is expressed by the `rel` element. An annotation example with hierarchical syntax annotation in the TIGER format is shown in Figure 2.2. A PAULA `struct` element with its daughters corresponds to a local TIGER subtree, i.e. a mother node and its immediate children. For instance, the subtree dominated by the first NP in Figure 2.2, *sein Tod*, ‘his death’, is represented by a `struct` element that, via `rel` elements, embeds the daughter tokens with IDs `tok_26/27` (these are stored in a separate file called “tiger.ex.tok.xml”). The NP subtree itself is dominated by another `struct` element, with ID `const_14`. `feat` elements encode the categorial status of these subtrees, “NP” and “S” respectively, and their grammatical functions. For example, the `rel` element with ID `rel_39`, which connects the subtree of S with the subtree of the NP, is marked as “SB” relation by the `feat` element pointing to `#rel_39`.

File *tiger.TIG49796.const.xml*:
```
...
<struct id="const_11">
   <rel id="rel_30" type="edge" xlink:href="tiger.ex.tok.xml#tok_26"/>
      <!-- Sein -->
   <rel id="rel_31" type="edge" xlink:href="tiger.ex.tok.xml#tok_27"/>
      <!-- Tod -->
</struct>
<struct id="const_14">
  <rel id="rel_38" type="edge" xlink:href="tiger.TIG49796.tok.xml#tok_28"/>
      <!-- hatte-->
  <rel id="rel_39" type="edge" xlink:href="#const_11"/>
  <rel id="rel_40" type="edge" xlink:href="#const_13"/>
</struct>
...
```

File *tiger.TIG49796.const˙cat.xml*:
```
...
<feat xlink:href="#const_11" value="NP"/>
<feat xlink:href="#const_14" value="S"/>
...
```
File *tiger.TIG49796.const˙func.xml*:
```
...
<feat xlink:href="#rel_30" value="NK"/><!-- Sein -->
<feat xlink:href="#rel_31" value="NK"/><!-- Tod -->
<feat xlink:href="#rel_39" value="SB"/>
...
```

A consequence of the decision to have annotations point – possibly by transitivity – to tokens is that the information cannot be directly read off in the opposite directions, i.e., for a particular token we do not represent explicit links to all its annotations. If needed, this has to be computed by traversing the various annotation layers, which is a functionality provided by our Java API (see Section 2.7).

2.4 Processing Pipeline

We now turn to the description of the MOTS workbench itself, which is realized as a pipeline architecture.

When a text document is submitted, a preprocessing stage first transforms it into the PAULA standoff format, which will be explained below. Then, during the “proper” processing stage it is enriched with further layers by the analysis components (see Section 2.5). Finally, all resulting PAULA layers are being merged into a standard inline XML representation, which is used for visualization purposes (see Section 2.6.1).

The pipeline is implemented as a shell script. It manages the flow of processing, starting from the input document and leading to an output representation that can be shown to the user, while the analysis steps can be flexibly switched on and off. Each component in the processing pipeline constitutes a distinct application and thus can also be executed outside of MOTS. The components (to be described in the next section) were developed in various programming languages; some of them are external off-the-shelf solutions, others were developed in-house by our research group and students.

MOTS offers a set of parameters for configuring the analysis pipeline. One class of parameters, as mentioned above, serves to de-/activate analysis components; violations of dependencies are detected automatically. An important parameter is the *genre* of the text, as several of our components provide genre-specific information tailored to, for example, news articles, film reviews, or court decisions. Other parameters reflect document properties such as the language (German or English) and the technical format of the input (see below). A parameter specific to the summarization task is the desired definition of ‘term’ (wordform, lemma, Porter stems, character based n-grams), which is then used for calculating sentence relevance. Certain other, more technical, parameters defining the use of meta information and output directories can be used on the command line, but are hidden in the regular user interface (the GUI page for setting these parameters is shown in Figure 2.6, Section 2.6).

A common problem for integrative NLP platforms is character encoding. All our intermediate representations are encoded in UTF-8, but because some of the modules work only with ISO 8859-1, we use only characters compatible to this encoding. In a first step, the input document is converted to ISO 8859-1. This conversion uses some intelligent features, such as mapping Cyrillic letters to their ISO 8859-1 transcriptions. Afterwards we convert the ISO 8859-1 file back to UTF-8. For the modules using ISO 8859-1, we use *gnu iconv* for converting the input from UTF-8 to ISO 8859-1 and the output back from ISO 8859-1 to UTF-8.

The preprocessing, i.e., the first phase of the pipeline, is performed obligatorily for each input document. It consists of three steps: conversion to a normalized format, tokenization, and conversion to PAULA standoff. The result is a PAULA representation of the input document with layout information, tokens and sentence boundaries. Next, we discuss the three steps in turn, and afterwards describe the integration of “real” analysis modules.

2.4.1 Normalized Input Format: LDOC

The system accepts input in various forms: plain text, some XML formats, and HTML. In the first preprocessing step, the input document is converted into our “normalized” XML format LDOC [21], which provides markup for layout features. The conversion identifies headers, paragraphs and highlighted text, and it extracts metadata from XML or HTML headers.

```
<?xml version="1.0" encoding="utf-8"?>
<ldoc id="text.utf8.in">
<body>
 <div id="div_1" type="heading" typeConf="high" >
      29. Dezember 2005
 </div>
 <div id="div_2" type="heading" typeConf="high">
     Bube, Damenopfer, König, As und Sieg
 </div>
 <div id="div_3" type="heading" typeConf="high">
      Match Point
 </div>
 <div id="div_4" type="paragraph" typeConf="medium">
     Ein junger Mann gerät unversehens in die High Society.
     Aufgrund seines gestiegenen Selbstbewusstseins verkalkuliert er sich
     im Privaten und schreckt schließlich vor einem Doppelmord nicht zurück,
     um seine Stellung zu verteidigen.
 </div>
 ...
 <div id="div_22" type="paragraph" typeConf="medium">
     © filmrezension.de
 </div>
</body>
</ldoc>
```

Fig. 2.3 LDOC representation of a film review.

```
<html>, <meta>, <head>, <body>
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>
<p>, <div>, <span>
<a>, <img>, <q>, <abbr>
<table>, <menu>, <ul>, <ol>, <dl>
<th>, <td>, <tr>, <li>, <dt>, <dd>
<i>, <b>, <u>, <strike>, <big>, <small>, <sub>, <sup>, <em>,
<strong>
```

Fig. 2.4 HTML tags considered for LDOC.

The LDOC format is defined and validated with a RELAX NG specification. Figure 2.3 shows an excerpt from a sample LDOC file. The general structure is as follows. A <header> tag encloses meta information, and the <body> tag encloses the document content. The layout is marked by <div> and <span>. <div> stands for *division* and marks the document structure, i.e., headings and paragraphs. <span> is used to mark smaller highlighted units within a <div>. Both tags have attributes. The *type* attribute, for instance, determines whether a <div> is a heading or a paragraph. Other attributes can encode the confidence value assigned by the converter. For instance, if a line in the text contains only one or two words, it is very likely that this line is a heading, while the confidence value for a longer line to be a header is lower.[10]

Our prototypical converter from HTML to LDOC resulted from a students project, where rules for all those HTML tags that are relevant for LDOC were

[10] These confidence values play a role predominantly for converting plain text documents, where quite a bit of guesswork can be involved.

developed; these tags are listed in Figure 2.4. All other tags are ignored. The converter works on XHTML, which is first produced from the HTML input.

2.4.2 Tokenization

The second preprocessing step is tokenization. Our tokenizer accepts an LDOC document as input, determines the character positions of individual tokens, and assigns a label to each token, which gives its type. Some of the types we use are XMLTAG, WORD, PUNCT, DATE, SBOUND, ABBREV, QUOTE, BRACE-OPEN, FLOAT, MIXEDSTRING.

Another task of the tokenizer is sentence boundary detection. Using lists of abbreviations, full stops are identified, disambiguated, and labeled accordingly. For German texts, the presence of upper-case letters at the beginning of sentences is taken into consideration: A determiner starting with an upper-case letter after an abbreviation or ordinal number marks the beginning of a sentence, while other tokens do not.

The tokenizer follows a decidedly "surface-level" approach and does not recognize any multi-word expressions such as proper names. This step is left to a dedicated named-entity recognition component that can be adapted to specific domains, while tokenization is a domain-independent task.

2.4.3 Conversion to PAULA

Tokenized LDOC is the input for the last preprocessing step: the conversion to PAULA standoff. The output is a set of PAULA layers: markables for text, token, sentence, div, span; features for div and span. The conversion distinguishes between XML and text tokens: XML tokens mark the layout, text tokens contain the original text. As indicated above, the PAULA 'text' layer is a sequence of all text tokens, while the 'token' layer records the character positions of each token. Each token is attached to a sentence in the corresponding layer, where headings are also regarded as sentences. Sentence boundaries are determined by full stops as well as closing division boundaries (</div>). Other markable layers are 'span' (referring to tokens) and 'div' (referring to sentences). For each attribute of <div> and <span>, a separate feature layer is produced.

Now the preprocessing is completed and the basic PAULA layers are available for the "real" analysis components in the pipeline, whose job it is to add further layers to the PAULA set.

2.4.4 Integrating Analysis Components

The flexible part of the pipeline can be configured for each execution by selecting the active processing modules. The pipeline script then manages the order of processing and resolves all module dependencies. Input and output of the modules in general are one or more layers of the PAULA standoff set. On the output side, in addition to PAULA layers, most modules also provide a human-readable format, which can be used for debugging purposes.

Any external, off-the-shelf tools that are to be integrated into MOTS need to be wrapped by converters that generate input for the tool from PAULA, and create PAULA layers from the output. The effort needed for building a converter depends on the type of the annotation. Writing a converter for a tool just annotating tokens with features is easily done. For annotations with complex structures, the task becomes more difficult. The efficiency of a converter at runtime depends on the complexity of annotations as well. For example, when running the Tree Tagger[11] and chunker [20], the time needed to convert from and to PAULA is $1/3$ of the overall runtime of the component. For this reason, we skip the conversion steps when components depend directly on another's output.

Another well-known issue with integrating a variety of off-the-shelf components is their tokenization behavior. Many "black box" components read their input as text and tokenize it by their own rules. In such cases, the converter has to align the tokens produced by the component with the "standard" MOTS tokenization performed in preprocessing. Usually, the differences are restricted to the interpretation of spaces, punctuation and sentence boundaries, but some tools also identify multiword tokens (such as *Golden Gate Bridge*) and portmanteau words (such as German *im*, which is evaluated as *i+m = in dem*). In such cases, the features of the additional or reduced tokens are determined by heuristic rules.

The central advantage of using PAULA standoff in MOTS is the flexibility in the architecture and pipeline configuration. Components such as taggers or parsers can be easily replaced, or both run on the same text for evaluation purposes. The results, always in the shape of different annotation layers on the same text, can be visualized quite straightforwardly (see Section 2.6). On the other hand, the substantial XML "packing and unpacking" at runtime comes with a cost that may be prohibitive for "real-time" applications; we will return to this issue at the end of the paper.

2.5 Analysis Components

In this section, we provide an overview of the analysis components that we have integrated into MOTS so far, sorted (roughly) by the level of analysis to which they apply: token, clause/sentence, and discourse.

Token Level

To provide a basic level of analysis for other components to build on, we integrated the Tree Tagger with the off-the-shelf models for English and German, along with its chunking mode. Thus two separate levels of analysis are created, one for part-of-speech tags and one for NP- and PP-chunks. Also operating on the level of tokens,

[11] `http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/`

a named entity recognizer combines gazetteer-based lookup of person's first names, location and company names with a set of rules that hypothesize the presence of a last name after a first name or a title, and the like [11]. Similarly, a component in charge of temporal information [17] identifies German time and date expressions (including their manifold linguistic variants such as: *11.25, fünfundzwanzig nach elf, fünf vor halb zwölf, ...*) and assigns corresponding formal representations in the *Temporal Expression Language* [10], which is similar in spirit to TimeML[12].

On the level of (sub-)tokens, we compute weights based on TF/IDF with respect to a genre-specific reference corpus. This is one of the points where the user's choice of genre for the input text plays a role. (If no genre is selected, a "generic" corpus is used.) These weights are later used for computing sentence weights (for summarization/extraction) as well as for isolating 'topic' identifiers in text tiles (see below). As for the unit of analysis, MOTS allows for selecting regular tokens as well as character n-grams. As shown by [2], different definitions are better-suited for different purposes.

Clause and Sentence Level

As a preparatory step for discourse-level analysis of coherence relations, we provide the Brill tagger with a model that we incrementally trained in order to perform disambiguation of certain connectives as to their sentential versus discourse reading. Based on the part-of-speech context, this tagger tries to distinguish the readings of, for example, German *darum* (causal connective versus verbal particle) – as we had reported in [8], off-the-shelf taggers for German often do not provide this information correctly, sometimes because of errors and sometimes because the part-of-speech tag is the same for both the discourse and the sentential reading.

Using the sentence boundaries and the part-of-speech tags, a first version of a subjectivity analysis component employs a rule-based approach to recognizing opinions in text. It was designed for the genre of student's evaluations of their classes and uses lists of subjective vocabulary tailored to this task. After identifying positive/negative lexemes from the lists, it checks for the presence of negations (using a simple fixed-size word window) and if appropriate, applies rules to reverse the polarity of the opinion. On the basis of these results, sentences are labelled as positive, negative, or neutral.

To enable deeper linguistic analyses, we integrated the Connexor syntax parser[13] for German. It delivers dependency structures that are required by the Rosana anaphora resolution (see below), and we also use them as the basis for other discourse-level tasks. In addition, we built a wrapper for the BitPar parser[14], which delivers constituent structures of sentences in the TIGER format.

[12] `http://www.timeml.org`
[13] `http://www.connexor.com`
[14] `http://www.ims.uni-stuttgart.de/tcl/SOFTWARE/BitPar.html`

Discourse Level

The anaphora resolution component *Rosana* was originally developed by [22] for English, and later (in collaboration with our group) also for German. Based on the Connexor dependency structures, Rosana tries to resolve pronouns, proper names and definite descriptions. A significant gap for the coreference analysis (especially for the genre of newspaper texts) is the handling of named entities. We are currently designing an approach to fusing the algorithms employed by Rosana with the results of our named-entity recognizer, working towards an integrated approach that would cover the whole variety of (direct) nominal anaphora.

For statistical text tiling and topic detection, we implemented algorithms by [14] and [24], which determine topic boundaries in the text (either in correspondence with or in ignorance of the paragraphs encoded in our LDOC layer) and also compute keywords that are representative for the particular "tile". For this purpose, we also make use of the aforementioned genre-specific reference corpora.

As an investigation into the 'rhetorical structure' of paragraphs, we developed a component for local coherence relations, which specifically identifies causal relations in German text. On the basis of a declarative lexicon, the presence of a relation is established by analyzing connectives and, if necessary, performing disambiguation (using rules operating on the part-of-speech context). Then, the spans related by the connective are hypothesized on the grounds of the syntactic dependency trees. This works quite well for conjunctions and prepositions; whereas for adverbial connectives, we can only guess that the two neighbouring sentences are in fact the related spans.

Finally, operating on the level of the complete text, our *IDOC* component performs an analysis of the functional role of the individual LDOC portions of the text (i.e., lines and paragraphs), similar to the "argumentative zoning" approach by [23]. The target of this approach are semi-structured documents, which display regularities as to the inventory of labels needed to describe the role of paragraphs and the linear order of the elements. Also, some of the zones need to be recognizable on the basis of surface features (words, length of paragraph, etc.). Our approach performs two steps: In the first phase, regular surface patterns are matched using LAPIS [18], thus identifying a certain number of zones reliably. In the second phase, the zones already found are used to hypothesize the presence of other zones in the remaining material, using likelihoods derived from surface features and from the neighbourhood of zones already found. So far, we implemented the approach for two genres: film reviews [3] and court decisions.

2.6 User Interface

When the execution of the pipeline script is complete, and all analyses are accomplished, the PAULA standoff set contains a lot of XML files that the average user would not want to inspect. Hence, the MOTS user interface is a convenient way to evaluate and compare all the results of the analysis pipeline.

```
<div _id="id_1893" _org_id="div_2" type="heading" typeConf="high"
zone="tagline|title">
 <sent _id="id_1101" _org_id="s_2" W5g="0.028237089618056307" Wxdoc="0.64">
  <tok _org_id="tok_4" _id="id_4" pos="NN" lemma="Bube">Bube</tok>
  <tok _org_id="tok_5" _id="id_5" pos="$," lemma=",">,</tok>
  <tok _org_id="tok_6" _id="id_6" pos="NN"
  lemma="Damenopfer">Damenopfer</tok>
  <tok _org_id="tok_7" _id="id_7" pos="$," lemma=",">,</tok>
  <tok _org_id="tok_8" _id="id_8" pos="NN" lemma="Koenig">Koenig</tok>
  <tok _org_id="tok_9" _id="id_9" pos="$," lemma=",">,</tok>
  <tok _org_id="tok_10" _id="id_10" pos="NN" lemma="As">As</tok>
  <tok _org_id="tok_11" _id="id_11" pos="KON" lemma="und">und</tok>
  <tok _org_id="tok_12" _id="id_12" pos="NN" lemma="Sieg">Sieg</tok>
 </sent>
</div>
```

Fig. 2.5 Inline representation.

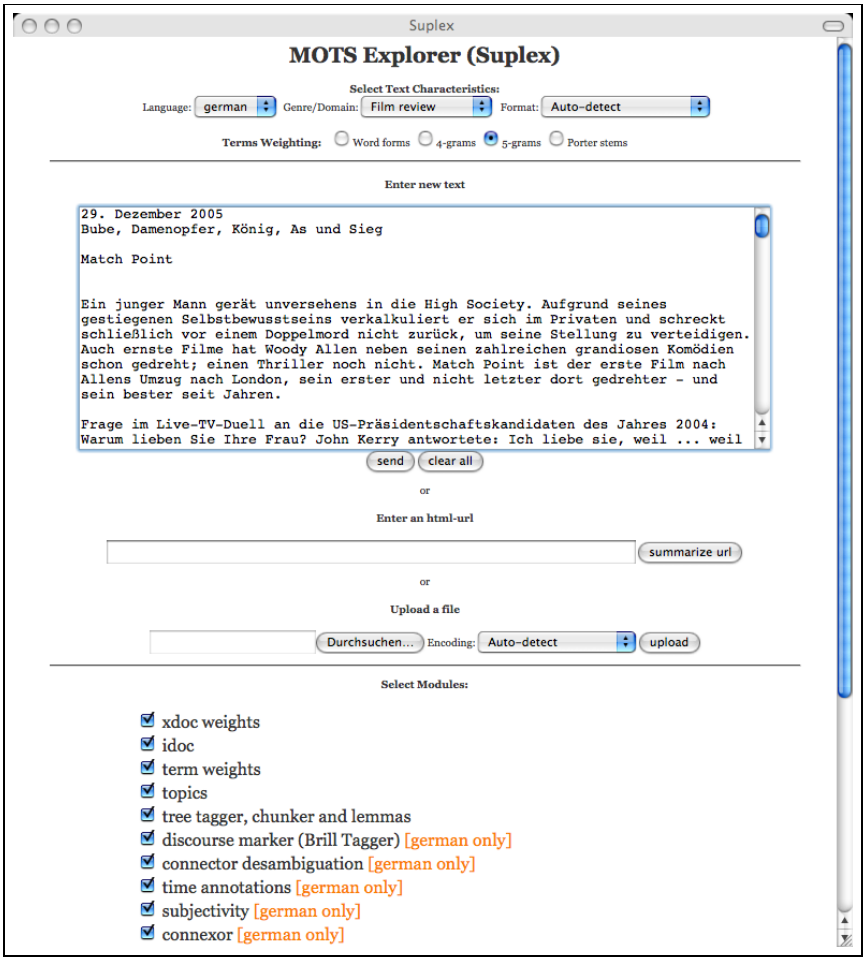
2.6.1 XML Inline Representation

The input to the visualization component is a standard inline-XML file whose format is called ‘PAULA-inline’. It results from merging all the PAULA layers into a single file – a step needed solely to facilitate the graphical presentation of the output. The inline-XML document contains an XML element for each markable (token or span). All features referring to this markable are annotated as attributes of this element. Smaller spans of markables are added as children of wider spans.

Figure 2.5 gives an extract of the inline representation of the beginning of a text we had shown earlier (Figure 2.3). The example shows a headline, which is also annotated as sentence. The tokens contained therein are represented as children of the sentence element. Tokens are annotated with *pos* and *lemma*. PAULA markables do not have any dominance relation between each other; they just mark spans in the text. If two markables cover exactly the same span of text, in the inline document one is arbitrarily chosen to embed the other. Thus in the example, the division element could also be included into the sentence element.

To mark “real” embedding in structures such as trees (PAULA <struct> elements), we use the special element <_rel>, which explicitly encodes the dominance relation in PAULA-inline. This allows us to annotate edges between nodes.

XML embedding cannot be used for the representation of overlapping segments. For such data, we use the strategy of fragmentation: One of the overlapping elements is broken into smaller units and an attribute *gid* (‘group id’) is added to the fragmented elements to explicitly mark elements that belong together. For further details on creating the PAULA-inline representation, see [9].



```
29. Dezember 2005
Bube, Damenopfer, König, As und Sieg

Match Point


Ein junger Mann gerät unversehens in die High Society. Aufgrund seines
gestiegenen Selbstbewusstseins verkalkuliert er sich im Privaten und schreckt
schließlich vor einem Doppelmord nicht zurück, um seine Stellung zu verteidigen.
Auch ernste Filme hat Woody Allen neben seinen zahlreichen grandiosen Komödien
schon gedreht; einen Thriller noch nicht. Match Point ist der erste Film nach
Allens Umzug nach London, sein erster und nicht letzter dort gedrehter - und
sein bester seit Jahren.

Frage im Live-TV-Duell an die US-Präsidentschaftskandidaten des Jahres 2004:
Warum lieben Sie Ihre Frau? John Kerry antwortete: Ich liebe sie, weil ... weil
```

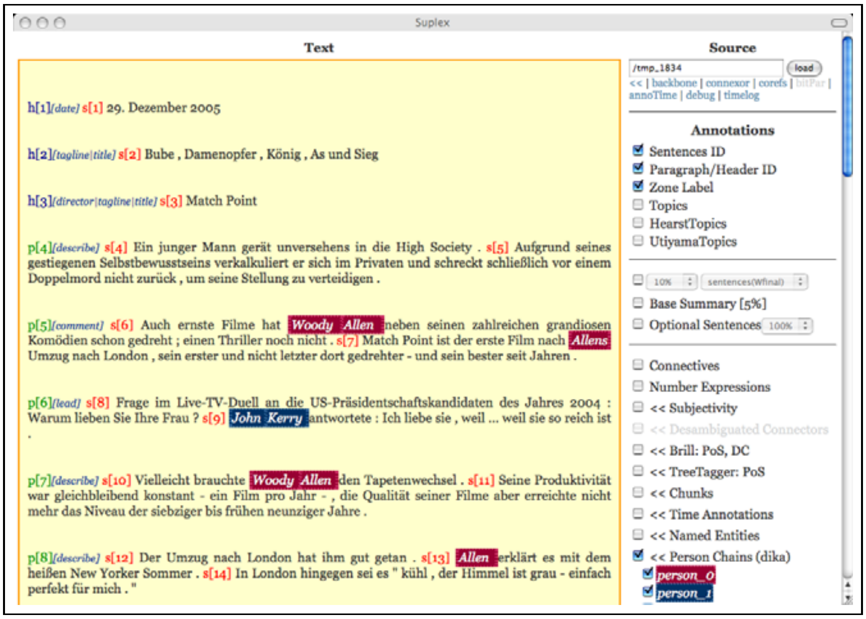
- ✓ xdoc weights
- ✓ idoc
- ✓ term weights
- ✓ topics
- ✓ tree tagger, chunker and lemmas
- ✓ discourse marker (Brill Tagger) [german only]
- ✓ connector desambiguation [german only]
- ✓ time annotations [german only]
- ✓ subjectivity [german only]
- ✓ connexor [german only]

Fig. 2.6 Graphical user interface: parameter selection.

2.6.2 *Visualization*

The user interface is a PHP script accessible with a web browser – see Figure 2.6. It is to be used in two steps: First, the user defines the input and the processing parameters. The input can be directly entered as text, uploaded as a file, or defined by a URL. Then, the pipeline script is started with the 'send' button. When processing is complete, the resulting PAULA-inline document is converted by an XSLT script to HTML and thus available for viewing with PHP and Javascript.

In the second step, the user can browse the results and compare various analyses (Figure 2.7). All annotations on a specific token are shown on a mouse-over. Other annotations on larger spans are highlighted upon request. Some component-specific



h[1]*[date]* s[1] 29. Dezember 2005

h[2]*[tagline|title]* s[2] Bube , Damenopfer , König , As und Sieg

h[3]*[director|tagline|title]* s[3] Match Point

p[4]*[describe]* s[4] Ein junger Mann gerät unversehens in die High Society . s[5] Aufgrund seines gestiegenen Selbstbewusstseins verkalkuliert er sich im Privaten und schreckt schließlich vor einem Doppelmord nicht zurück , um seine Stellung zu verteidigen .

p[5]*[comment]* s[6] Auch ernste Filme hat *Woody Allen* neben seinen zahlreichen grandiosen Komödien schon gedreht ; einen Thriller noch nicht . s[7] Match Point ist der erste Film nach *Allens* Umzug nach London , sein erster und nicht letzter dort gedrehter - und sein bester seit Jahren .

p[6]*[lead]* s[8] Frage im Live-TV-Duell an die US-Präsidentschaftskandidaten des Jahres 2004 : Warum lieben Sie Ihre Frau ? s[9] *John Kerry* antwortete : Ich liebe sie , weil ... weil sie so reich ist .

p[7]*[describe]* s[10] Vielleicht brauchte *Woody Allen* den Tapetenwechsel . s[11] Seine Produktivität war gleichbleibend konstant - ein Film pro Jahr - , die Qualität seiner Filme aber erreichte nicht mehr das Niveau der siebziger bis frühen neunziger Jahre .

p[8]*[describe]* s[12] Der Umzug nach London hat ihm gut getan . s[13] *Allen* erklärt es mit dem heißen New Yorker Sommer . s[14] In London hingegen sei es " kühl , der Himmel ist grau - einfach perfekt für mich . "

/tmp_1834
<< | backbone | connexor | corefs | bitPar | annoTime | debug | timelog

Annotations
- ✓ Sentences ID
- ✓ Paragraph/Header ID
- ✓ Zone Label
- ☐ Topics
- ☐ HearstTopics
- ☐ UtiyamaTopics

- ☐ 10% sentences(Wfinal)
- ☐ Base Summary [5%]
- ☐ Optional Sentences 100%

- ☐ Connectives
- ☐ Number Expressions
- ☐ << Subjectivity
- ☐ << Desambiguated Connectors
- ☐ << Brill: PoS, DC
- ☐ << TreeTagger: PoS
- ☐ << Chunks
- ☐ << Time Annotations
- ☐ << Named Entities
- ✓ << Person Chains (dika)
  - ✓ *person_0*
  - ✓ *person_1*

Fig. 2.7 Graphical user interface: browsing results.

visualization is being realized. Sentence and paragraph boundaries and headings are marked by a tag at the beginning of the text range. Tile boundaries are marked more prominently and are completed by keywords describing the topic. Most annotations assign labels to ranges of text (mostly tokens). The labels appear at the right frame of the page ordered by the kind of annotation and can be selected for colored highlighting. In addition to the merged graphical presentation, the output of each component is accessible in its original form by following hyperlinks; likewise we provide debugging and runtime information on separate pages. Also, there is a button for downloading the PAULA files that have been produced, so that they can be used as input for other software.

2.7 Current Developments and Conclusion

The file-centric approach of MOTS, managing interfaces through a fairly generic standoff annotation format, resulted on the one hand from the dual needs of (i) bridging between manual annotation tools and (ii) automatic processing. At the same time, it provides a simple way of allowing for interoperability of components that enables rapid prototyping, which has been utilized in a number of students' projects and diploma theses. One advantage here is the independence on programming languages: Existing modules in script languages or more traditional languages (e.g., Lisp) can be added quite easily. In this way, MOTS enables quick experiments with combining modules for new tasks, without right away taking the step to work with a more complex system such as GATE or UIMA.

In this spirit of a lean, "low-cost" framework, we are currently making several additions to MOTS, whose primary goal is to somewhat reduce the effort of XML processing in a pipeline of components. A Java API is being developed that will be in charge of parsing PAULA files and providing convenient access to the data via the PAULA Object Model (cf. Section 2.3.1). One perspective then is to replace the shell script managing the processing pipeline with a Java application, so that PAULA processing can be restricted to the interfaces of non-Java components.

For the time being, we will hold on to the menu-based de-/activation of components when the pipeline is started. For most purposes of text processing, it has proven sufficient, and moving to a fully-flexible component management system would amount a leap toward GATE-style frameworks, which we are not intending. However, a slightly more generic way of describing component behavior (using simple configuration files) could be added, so that integrating new modules can be done more systematically than by changing the shell script. Likewise, it should be possible to integrate annotations that have been produced manually with a suitable tool. This can be interesting when a higher-level component builds on the output of a lower-level one and for testing purposes, "perfect results" can be provided in place of the lower-level component's output.

Our approach to visualising results currently creates problems with long texts holding many annotations, as Javascript execution becomes prohibitively expensive. While this can be attended to with limited effort, a more substantial problem is the conversion from (possibly many) standoff files to inline-XML, which our GUI is currently based on. This merging step is an inherently complex task, and rather than trying to improve our current merging solution, it is probably more effective to try to circumvent it altogether and base the visualisation on the standoff files. This would amount to a general overhaul of the output side of the GUI, offering the opportunity to establish a more generic solution that maps types of annotations to their visual counterparts (both menu items for clicking annotations on/off and the annotation visualization itself).

Our PAULA tools are being made available via the webpage mentioned in Section 2.3, and in conjunction with our ANNIS linguistic database (focusing on the scenario of manual annotation) [4]. Likewise, the MOTS software is available to interested parties for research or teaching.

Acknowledgements

Part of the work reported in this paper was funded by Bundesministerium für Bildung und Forschung (BMBF), grant 03WKH22.

The following researchers (listed in alphabetical order) have contributed to the development of the MOTS workbench and the PAULA format and tools: Christian Chiarcos, Stefanie Dipper, Michael Götze, Peter Kolb, Uwe Küssner, Julia Ritz, Johannes Schröder, Arthit Suryiawongkul. Also, many of our Computational Linguistics students helped building conversion tools or analysis components.

We are grateful to two anonymous reviewers for their helpful comments on an earlier version of this paper.

References

[1] Amtrup, J.: Ice - intarc communication environment user guide and reference manual version 1.4. Tech. rep. Universität Hamburg (1995)

[2] Bieler, H., Dipper, S.: Measures for term and sentence relevances: an evaluation for german. In: Proceedings of the 6th LREC Conference, Marrakech (2008)

[3] Bieler, H., Dipper, S., Stede, M.: Identifying formal and functional zones in film reviews. In: Proceedings of the Eighth SIGDIAL Workshop, Antwerp (2007)

[4] Chiarcos, C., Dipper, S., Götze, M., Ritz, J., Stede, M.: A flexible framework for integrating annotations from different tools and tagsets. In: Proc. of the First International Conference on Global Interoperability for Language Resources, Hongkong (2008)

[5] Cunningham, H.: Software architecture for language engineering. PhD thesis, University of Sheffield (2000)

[6] Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A framework and graphical development environment for robust NLP tools and applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (2002)

[7] Dipper, S.: XML-based stand-off representation and exploitation of multi-level linguistic annotation. In: Eckstein, R., Tolksdorf, R. (eds.) Proceedings of Berliner XML Tage, pp. 39–50 (2005)

[8] Dipper, S., Stede, M.: Disambiguating potential connectives. In: Butt, M. (ed.) Proceedings of KONVENS 2006, Konstanz, pp. 167–173 (2006)

[9] Dipper, S., Götze, M., Küssner, U., Stede, M.: Representing and querying standoff XML. In: Proceedings of the Biennial GLDV Conference 2007. Data Structures for Linguistic Resources and Applications, Narr, Tübingen (2007)

[10] Endriss, U., Küssner, U., Stede, M.: Repräsentation zeitlicher Ausdrücke: Die Temporal Expression Language. Verbmobil Memo 133, Technical University Berlin, Department of Computer Science (1998)

[11] Ernst, C.: Auffinden von Named Entities in Nachrichtentexten. Diplomarbeit, Institut für Linguistik, Universität Potsdam (2008)

[12] Evert, S., Carletta, J., O'Donnell, T., Kilgour, J., Vögele, A., Voormann, H.: The nite object model. version 2.1. Tech. rep., University of Edinburgh, Language Technology Group (2003)

[13] Grishman, R.: Tipster architecture design document version 2.3. Tech. rep., DARPA (1997),
`http://www.itl.nist.gov/div894/894.02/related_projects/tipster/`

[14] Hearst, M.A.: Multi-paragraph segmentation of expository text. In: Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Las Cruces/NM, pp. 9–16 (1994)

[15] Ide, N., Romary, L.: International standard for a linguistic annotation framework. Natural Language Engineering 10(3-4), 211–225 (2004)

[16] Ide, N., Suderman, K.: Graf: A graph-based format for linguistic annotation. In: Proceedings of The Linguistic Annotation Workshop (LAW), Prague (2007)
[17] Luft, A.: Automatisches Tagging von zeitlichen Ausdrücken. Diplomarbeit, Institut für Informatik, FH Mittweida (2006)
[18] Miller, R.C.: Lightweight structure in text. PhD thesis, Carnegie Mellon University (2002)
[19] Schäfer, U.: Integrating deep and shallow natural language processing components - representations and hybrid architectures. PhD thesis, Universität des Saarlandes (2007)
[20] Schmid, H.: Probabilistic part-of-speech tagging using decision trees. In: Proceedings of International Conference on New Methods in Language Processing, Manchester, pp. 44–49 (1994)
[21] Stede, M., Suriyawongkul, A.: Identifying logical structure and content structure in loosely-structured documents. In: Witt, A., Metzing, D. (eds.) Linguistic Modeling of Information and Markup Languages - Contributions to Language Technology, pp. 81–96. Springer, Dordrecht (2010)
[22] Stuckardt, R.: Design and enhanced evaluation of a robust anaphor resolution algorithm. Computational Linguistics 27(4), 479–506 (2001)
[23] Teufel, S., Moens, M.: Summarizing scientific articles – experiments with relevance and rhetorical status. Computational Linguistics 28(4), 409–445 (2002)
[24] Utiyama, M., Isahara, H.: A statistical model for domain-independent text segmentation. In: Proceedings of the ACL/EACL Conference, Toulouse (2001)