# Integrating the Rule-Based IE Component TEXTMARKER into UIMA

**Peter Kluegl** and **Martin Atzmueller** and **Frank Puppe**
Department of Computer Science,
University of Würzburg, Germany
{pkluegl, atzmueller, puppe}@informatik.uni-wuerzburg.de

## Abstract

In this paper we describe the integration of the rule-based IE component TEXTMARKER into the UIMA framework. We present a conceptual overview on the TEXTMARKER system before we describe the UIMA integration in detail.

## 1 Introduction

UIMA [Ferrucci and Lally, 2004], the *Unstructured Information Management Architecture* is a powerful component-based framework for systems dealing with the analysis of unstructured information. UIMA already contains various components for information extraction. However, to the best of the author's knowledge, so far there is no prominent open source example for a sophisticated rule-based information extraction component. Rule-based components are especially important due to their understandability, explainability and extensability. In information extraction there are several reason to prefer a knowledge engineering approach to a machine learning approach [Appelt, 1999]. The necessary knowledge can usually be captured quite easily by a domain specialist.

In this paper, we describe the integration of the rule-based information extraction component TEXTMARKER into the UIMA framework. TEXTMARKER is a versatile IE system and allows for both low-level and higher-level information extraction tasks due to its simple rule-based scripting language. The developed scripts can then also be coupled with other UIMA components.

The rest of the paper is structured as follows: Section 2 gives a conceptual overview on the TEXTMARKER system, describing the core concepts, the syntax and semantics of the TEXTMARKER language and its special features. Section 3 discusses the integration into the UIMA framework in detail. Finally, Section 4 concludes with a discussion of the presented work, and provides several interesting options for future work.

## 2 Conceptual Overview on the TEXTMARKER System

Whenever humans perform manual information extraction they often apply a strategy according to a *highlighter metaphor*: First relevant headlines are considered and classified according to their content by coloring them with different highlighters. The paragraphs of the annotated headlines are then considered further. Relevant text fragments or single words in the context of that headline can then be colored. Necessary additional information can be added that either refers to other text segments or contains valuable domain specific information. Finally the colored text can be easily analyzed concerning the relevant information.

The TEXTMARKER[1] system tries to imitate this manual extraction method by formalizing the appropriate actions using *matching rules*: The rules mark sequences of words, extract text segments or modify the input document depending on textual features. The default input for the TEXTMARKER system is semi-structured text, but it can also process structured or free text. Technically, HTML is often used as an input format, since most word processing documents can be easily converted to HTML.

### 2.1 Core TEXTMARKER Concepts

As a first step in the extraction process the TEXTMARKER system uses a scanner to tokenize the input document and to create a stream of basic symbols, providing the initial feature selection. The types of the possible tokens are predefined by a taxonomy of annotation types. Annotations simply refer to a section of the input document and assign a type or concept to the respective text fragment. Figure 1 shows an excerpt of a basic annotation taxonomy: E.g., *CW* describes all tokens, that contains a single word starting with a capital letter, *MARKUP* corresponds to HTML/XML tags and *PM* refers to punctuation.

Using the taxonomy, the knowledge engineer is able to choose the most adequate types and concepts when defining new matching rules. If the capitalization of a word, for example, is of no importance, then the annotation type *W* that describes words of any kind can be used.

The initial scanner creates a basic set of annotations that are used by the matching rules of TEXTMARKER. Most information extraction applications require domain specific concepts and annotations. Therefore, the knowledge engineer is able to define new annotation types depending on the requirements of the given domain. These types can then be flexibly integrated in the taxonomy of annotation types.

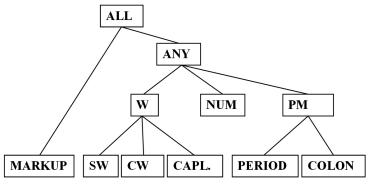


Figure 1: Part of a taxonomy for basic types. (W=Word, NUM=Number, PM=Punctuations, SW=Word without capitals, CW=Word starting with a capital letter)

[1] textmarker is a common german word for a highlighter

## 2.2 Syntax and Semantics of the TEXTMARKER Language

One of the goals in developing a new information extraction language was to maintain an easily readable syntax while still providing a scalable expressiveness of the language. Basically, the TEXTMARKER language consists of definitions of new annotation types and matching rules. These rules are based on a list of rule elements headed by the type of the rule.

The purpose of the different rule types is to increase the readability of rules by making their semantic intention explicit. Each rule element contains at least a basic matching condition referring to text fragments or already given annotations. Additionally a list of conditions and actions may be specified for a rule element. Whereas the conditions describe necessary attributes of the matched text fragment, the actions point to operations and assignments on the current fragments. Needless to say these actions will only be executed if all basic conditions matched on a text fragment or annotation and the related conditions are fulfilled. Table 2.2 contains a short and simplified excerpt of the TextMarker syntax concerning matching rules.

| | |
|---|---|
| Rule | → RuleType RuleElement+ ';' |
| RuleType | → 'ADDTYPE' \| 'DEFAULT' \| ... |
| RuleElement | → MatchType Conditions? Actions? '+'? |
| MatchType | → Literal \| Annotation |
| Annotation | → 'ALL'\|'ANY'\|'MARKUP'\|'W'\|... |
| Conditions | → '{' Condition (';' Condition)* '}' |
| Condition | → '-'? CondType (',' parameter)* |
| CondType | → 'PARTOF'\|'CONTAINS'\|'NEAR'\|... |
| Actions | → '(' Action (';' Action)* ')' |
| Action | → ActionType (',' parameter)* |
| ActionType | → 'MARK'\|'FILTER'\|'REPLACE'\|... |

Table 1: BNF-Extract of the TEXTMARKER language

Due to the limited space it is not possible to describe all of the various conditions and actions available in the TEXTMARKER system. However, the common usage of the language and its readability can be demonstrated by simple examples:

```
ADDTYPE CW{INLIST,animals.txt}(MARK,animal);
ADDTYPE animal 'and' animal
        (MARK,animalpair,0,1,2);
```

The first rule looks at all capitalized words that are listed in an external document *animals.txt* and creates a new annotation of the type *animal* using the boundaries of the matched word. The second rule searches for an annotation of the type *animal* followed by the literal *and* and a second *animal* annotation. Then it will create a new annotation *animalpair* covering the text segment that matched the three rule elements (the digit parameters refer to the number of matched rule element).

```
ADDTPYE W(MARK,firstname,firstnames.txt)
ADDTYPE firstname CW(MARK,lastname)
LOGGING paragraph{VOTE,firstname,lastname}
    (LOG,'Found more firstnames
    than lastnames')
```

In this example, the first rule annotates all words that occur in the external document *firstnames.txt* with the type *firstname*. The second rule creates a *lastname* annotation for all capitalized words that follow a *firstname* annotation. The last rule finally processes all *paragraph* annotations. If the *VOTE* condition counts more *firstname* than *lastname* annotations, then the rule writes a log entry with a predefined message.

```
ADDTYPE ANY{PARTOF,paragraph,ISINTAG,
        font,color=red}(MARK,delete,+)+;
ADDTYPE firstname(MARK,delete,0,1) lastname;
DEFAULT delete(DEL);
```

Here, the first rule looks for sequences of any kind of tokens except markup and creates one annotation of the type *delete* for each sequence, if the tokens are part of a *paragraph* annotation and colored in red. The + signs indicate this greedy processing. The second rule annotates first names followed by last names with the type *delete* and the third rule simply deletes all text segments that are associated with that *delete* annotation.

## 2.3 Special Features of the TEXTMARKER Language

The TEXTMARKER language features some special characteristics that are usually not found in other rule-based information extraction systems. The possibility of creating new annotation types and integrating them into the taxonomy facilitates an even more modular development of information extraction systems than common rule-based approaches do. Beside others, there are three features that deserve a closer look in the scope of this work: The robust extraction by filtering the token or annotation set, the usage of scoring rules for uncertain and heuristic extraction and the shift towards a scripting language.

**Robust extraction using filtering**

Rule-based or pattern-based information extraction systems often suffer from unimportant fill words, additional whitespace and unexpected markup. The TEXTMARKER System enables the knowledge engineer to filter and to hide all possible combinations of predefined and new types of annotations. Additionally, it can differentiate between any kind of HTML markup and XML tags. The visibility of tokens and annotations is modified by the actions of rule elements and can be conditioned using the complete expressiveness of the language. Therefore the TEXTMARKER system supports a robust approach to information extraction and simplifies the creation of new rules since the knowledge engineer can focus on important textual features. If no rule action changed the configuration of the filtering settings, then the default filtering configuration ignores whitespaces and markup. Using the default setting, the following rule matches all three types of input in this example (see [von Schoen, 2006]):

```
DEFAULT 'Dr' PERIOD CW CW;

Dr. Peter Steinmetz, Dr.PeterSteinmetz,
Dr. <b><i>Peter</i> Steinmetz</b>
```

**Heuristic extraction using scoring rules**

Diagnostic scores [Puppe, 2000] are a well known and successfully applied knowledge formalization pattern for diagnostic problems. Single known findings valuate a possible solution by adding or subtracting points on an account of that solution. If the sum exceeds a given threshold, then the solution is derived. One of the advantages of this pattern is the robustness against missing or false findings, since a high number of findings is used to derive a solution.

The TEXTMARKER system tries to transfer this diagnostic problem solution strategy to the information extraction problem. In addition to a normal creation of a new annotation, a *MARK* action can add positive or negative scoring

points to the text fragments matched by the rule elements. If the amount of points exceeds the defined threshold for the respective type, then a new annotation will be created. Further, the current value of heuristic points of a possible annotation can be evaluated by the *SCORE* condition. In the following, the heuristic extraction using scoring rules is demonstrated by a short example:

```
ADDTYPE p{CONTAINS,W,1,5}(MARK,hl,5);
ADDTYPE p{CONTAINS,W,6,10}(MARK,hl,2);
ADDTYPE p{CONTAINS,emph,80,100,%}(MARK,hl,7);
ADDTYPE p{CONTAINS,emph,30,80,%}(MARK,hl,3);
ADDTYPE p{CONTAINS,W,0,0}(MARK,hl,-50);
ADDTYPE hl{SCORE,10}(MARK,realhl);
LOGGING hl{SCORE,5,10}(LOG,'Maybe a hl');
```

In the first part of this rule set, annotations of the type *p* (*paragraph*) receive scoring points for a *hl* (*headline*) annotation, if they fulfill certain *CONTAINS* conditions. The first condition, for example, evaluates to *true*, if the paragraph contains at least one and up to five words, whereas the fourth conditions is fulfilled, if the paragraph contains thirty up to eighty percent of *emph* annotations. The last two rules finally execute their actions, if the score of a *headline* annotation exceeds ten points, or lies in the interval of five and ten points, respectively.

**Shift towards a scripting language**

Some projects using the TEXTMARKER system have revealed that a rule-based language with a knowledge representation only based on annotations may not overcome all challenges of a high level information task. Often it is not possible to express the complex background knowledge with simple matching rules or to control the matching process without control structures like loops. Therefore the TEXTMARKER language is being extended with several common features of scripting languages.

- **Imperative programming:** The TEXTMARKER system does not impose an execution order on the rules based on the fulfilled conditions and/or their complexity. The development of bigger sets of rules has shown that a dynamic execution order holds almost no advantages over imperative program execution order. Additionally, the linear processing allows a comprehensible usage of filtering rules.
- **Variables:** The usage of variables can significantly improve the expressiveness of a simple language. It is already possible in the TEXTMARKER system to e.g., count the number of certain annotation types and evaluate the result in a condition. But practice has shown that the additional concept of variables and expressions on variables is very useful and helps to solve complex rule engineering problems in an elegant and simple way.
- **Conditioned loop blocks:** Besides variables and operations another construct known by scripting and programming languages are conditioned statements and loops. In the TEXTMARKER system we combine both constructs to the concept of the conditioned loop blocks. These complex statements contain an identifier, a rule and a list of rules, declarations or even other conditioned loop blocks. The rule defines both the condition statement and the loop statement: The annotations of the rule match on adequate text fragments. The conditions of the rule determine if the contained rules may be executed. Yet the rule can match several times and therefore defines a list of text fragments, on which the contained rules are applied.

  ```
  BLOCK('ID') headlinedParagraph
      {CONTAINS,relevantAnnotation} ( ...
      rules, declarations or blocks ...
  )
  ```

  In this short and simplified example, a block of rules, declarations or blocks are only executed if an annotation of the type *headlinedParagraph* is found and if that annotation contains at least one annotation of the type *relevantAnnotation* (condition statement). The statements in the block will be applied on all found text fragments the rule matched and only on them. More precisely, if the rule has matched on five *headlinedParagraph* annotations, the contained statements will be executed five times overall, one time for each matched annotations. This additional block structure can therefore increase the performance of the TEXTMARKER system, because the considered text area can be restricted.
- **Method calls:** Another common feature is usage of methods or procedures. For this purpose we are using the conditioned loop blocks again. The identifier is used to call the block like a method by the action of a rule. If the calling rule is not part of the same block or rule file, additional identifiers, respectively file names must be used to reference the complete namespace of the block. Introducing method calls is enabling the TEXTMARKER system to utilize rule libraries and further increases its modularity and reuse.

This effort for extending the TEXTMARKER language towards a scripting language was one of the reasons to replace the existing and successful development environment [von Schoen, 2006]. The new implementation is built on the Dynamic Language Toolkit[2] in order to support the described scripting functionality and UIMA integration in the development process. Figure 2 shows an example of a simple scripting project for layout processing with various editors and views.

## 3 UIMA integration

The Unstructured Information Management Architecture (UIMA) [Ferrucci and Lally, 2004] is a flexible and extensible architecture for the analysis and processing of unstructured data, especially text. Like many other architectures (e.g., [Callmeier *et al.*, 2004]) it supports the interoperability of annotation tools, which drew more and more attention lately. The annotations, the artifact, respectively the text document and its meta data are represented by the Common Analysis Structure (CAS). Different components defined by descriptor files can process this CAS in a pipeline architecture. Whereby components like Analysis Engines add new annotations, CAS Consumer process the contained information further. Additionally, UIMA supports multiple CAS and different views on a CAS. A major point for the interoperability is the concept of the type system that defines the different types used by a component. Types themselves contain a set of features, more precisely references to other types or simple values. UIMA provides no default type system, but efforts are already being made to create type systems e.g. for common natural language processing tasks [Hahn *et al.*, 2007].

[2]Dynamic Language Toolkit: http://www.eclipse.org/dltk/

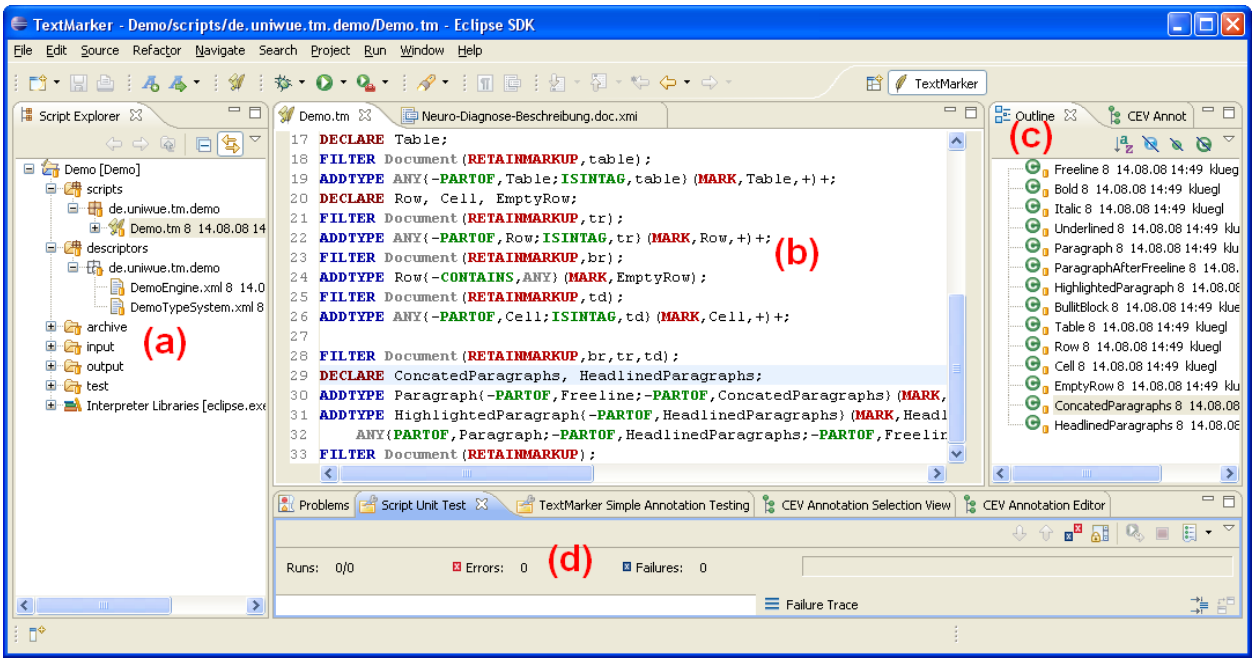


Figure 2: Screenshot of the development environment of the TEXTMARKER system with the project structure (a), the full-featured editor (b), the defined annotations (c) and various tools for testing and visualisation (d).

The TEXTMARKER system creates new annotations and is integrated into UIMA as an analysis engine. Figure 3 shows an exemplary overview of the transformation of rule files to the components of the architecture. During the transformation a build process is initiated given a rule file resulting in two descriptors. In our example, the rule file *A* is the central unit of the information extraction application. It uses several conditioned loop blocks that are defined in the separate rule files *B* and *C*. The rule file *B* calls the rule file *D*. The output of the build process are two kinds of descriptors: The type system descriptor and the analysis engine descriptor, i.e. the TEXTMARKER engine. The details about the generated features of the descriptors are specified below.

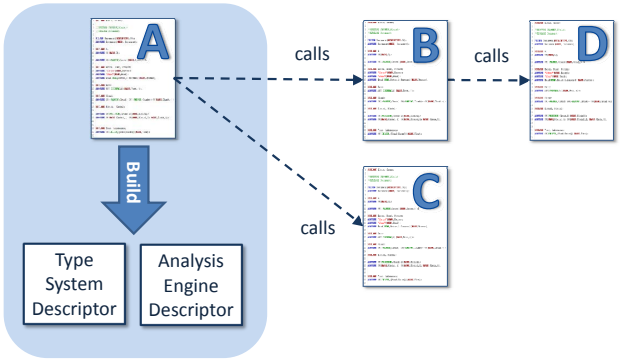


Figure 3: Overview over the TextMarker build process

- **Implementation of the annotator:** The analysis engine descriptor in UIMA needs an implementation of an annotator. The build process of the TEXTMARKER system uses a generic implementation of an annotator, for which the functionality is defined by the given set of rule files. Additionally, the annotator calls a scanner to create the basic annotation, if this feature of the analysis engine is enabled. The scanner is obsolete if more than one TEXTMARKER analysis engine is used by an aggregate annotator, since the initial feature selection only has to be performed once.
- **Type system:** The type system of the TEXTMARKER engine is created based on the defined types of annotations of all of the used rule files. Furthermore the rule files can import external type systems that are then included in the generated type system descriptor. The UIMA capabilities of the TEXTMARKER engine are determined by types that are provided by the different rule files.
- **External resources:** In the example above, the rule scripts *A*, *B*, *C* and *D* provide the functionality of the TEXTMARKER engine and have to be accessible by the annotator in order to complete its initialization. For this purpose we use both the external resources feature and the parameter feature of UIMA to realize an access to the rule scripts. For external dictionaries or lexica used by a rule script an additional external resource is generated.
- **Debugging information:** Debugging functionality is a central and important feature for a scripting language and even more for a rule-based language. The TEXTMARKER engine can be parameterized for enabling diagnostic debugging output. If activated, information about the amount of successful and tried applications of each rule or block and the annotations tried to match are stored in the CAS. Further detailed information about the conditions of their rule elements is also added.
- **Output document:** TEXTMARKER can also modify the input document. The TEXTMARKER annotator uses the view functionality of UIMA and creates an output view. Other analysis engines or CAS consumers can then process the changed document further.

## 4 Conclusions

Information extraction is part of a widespread and active research area that originates a multiplicity of new systems, tools and approaches. Initial development of the TEXTMARKER system was influenced by the LAPIS system (*Lightweight Architecture for Processing Information Structure*)[Kuhlins and Tredwell, 2003] and the LIXTO SUITE [Baumgartner *et al.*, 2001b]. The LAPIS system executes self-explanatory script-like edit operations on the input document. Providing a graphical user interface with an integrated browser, this system allows to revise the extraction results in a HTML view. But its original purpose as innovative text editor lacks some essential concepts like the definition of new types and the representation of uncertainty that is necessary for the effective text extraction. The LIXTO SUITE with its *Lixto Visual Wrapper* provides a graphical user interface for a semiautomatic generation of wrappers. The supervised learning approach uses manual annotations and decisions of the user to learn and refine rules of the ELOG language [Baumgartner *et al.*, 2001a]. The visual programming approach seems to prefer simple conditions instead of complex ones that would increase the robustness of the wrapper. The TEXTMARKER language with its additional conditions and actions uses a new notion of matching rules, but contains also features similar to other languages. E.g., the WHISK system [Soderland, 1999] learns matching rules similarly to regular expressions with predefined types of annotations. New annotations are created by separate instructions based on capturing groups of the rule. The MINORTHIRD toolkit [Cohen, 2004] contains an annotation language called MIXUP that uses the notion of cascaded finite state transducers.

This paper presents the integration of the rule-based IE component TEXTMARKER into the UIMA framework. We have given an conceptual overview on the TEXTMARKER system including the core concepts of the system, the syntax and semantics, and the special features of the TEXTMARKER language. Additionally, the UIMA integration of TEXTMARKER was discussed in detail.

The TEXTMARKER system and its integration into UIMA is still work in progress. The described functionality is almost completely implemented and the TEXTMARKER system with UIMA is already used successfully in real world applications. We are still gathering experience to improve the language and the level of its expressiveness in order to create an easy to use and scalable system. The special features of the TEXTMARKER system provide the possibility to create robust, complex and easy to adapt information extraction systems. With the integration in UIMA the TEXTMARKER system can benefit from the fast growing number of available UIMA components. However, with TEXTMARKER, the UIMA community also receives a new powerful analysis component. Besides its original purpose, the information extraction, TEXTMARKER can be used by other components for various task: E.g., for the knowledge based feature selection, to solve the type system alignment problem or to modify the input document.

For future work we plan to consider more elaborate scripting solutions. Additionally, a library of rule-scripts will further improve the rapid development of TEXTMARKER scripts. Another interesting option is to integrate machine-learning techniques with TEXTMARKER for a more semi-automatic development process and to provide solutions for test-driven development of rule-scripts.

## Acknowledgments

This work has been partially supported by the German Research Council (DFG) under grant Pu 129/8-2.

## References

[Appelt, 1999] Douglas E. Appelt. Introduction to Information Extraction. *AI Commun.*, 12(3):161–172, 1999.

[Baumgartner *et al.*, 2001a] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. The Elog Web Extraction Language. In *LPAR '01: Proceedings of the Artificial Intelligence on Logic for Programming*, pages 548–560, London, UK, 2001. Springer-Verlag.

[Baumgartner *et al.*, 2001b] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web Information Extraction with Lixto. In *The VLDB Journal*, pages 119–128, 2001.

[Callmeier *et al.*, 2004] Ulrich Callmeier, Andreas Eisele, Ulrich Schäfer, and Melanie Siegel. The Deep Thought Core Architecture Framework. In *Proceedings of LREC 04*, Lisbon, Portugal, 2004.

[Cohen, 2004] William W. Cohen. Minorthird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data. http://minorthird.sourceforge.net, 2004.

[Ferrucci and Lally, 2004] David Ferrucci and Adam Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Nat. Lang. Eng.*, 10(3-4):327–348, 2004.

[Hahn *et al.*, 2007] Udo Hahn, Ekaterina Buyko, Katrin Tomanek, Scott Piao, John McNaught, Yoshimasa Tsuruoka, and Sophia Ananiadou. An annotation type system for a data-driven NLP pipeline. In *The LAW at ACL 2007: Proceedings of the Linguistic Annotation Workshop*, pages 33–40, Prague, Czech Republic, 2007.

[Kuhlins and Tredwell, 2003] Stefan Kuhlins and Ross Tredwell. Toolkits for Generating Wrappers – A Survey of Software Toolkits for Automated Data Extraction from Web Sites. In Mehmet Aksit, Mira Mezini, and Rainer Unland, editors, *Objects, Components, Architectures, Services, and Applications for a Networked World*, volume 2591 of *Lecture Notes in Computer Science (LNCS)*, pages 184–198, Berlin, October 2003. International Conference NetObjectDays, NODe 2002, Erfurt, Germany, October 7–10, 2002, Springer.

[Puppe, 2000] Frank Puppe. Knowledge Formalization Patterns. In *Proc. PKAW 2000*, Sydney, Australia, 2000.

[Soderland, 1999] Stephen Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Mach. Learn.*, 34(1-3):233–272, 1999.

[von Schoen, 2006] Patrick von Schoen. Textmarker: Automatische Aufbereitung von Arztbriefen für Trainingsfälle mittels Anonymisierung, Strukturerkennung und Teminologie-Matching. Master's thesis, University of Wuerzburg, 2006.