TextMarker: A Tool for Rule-Based Information Extraction

Peter Kluegl, Martin Atzmueller, and Frank Puppe

University of Würzburg,
Department of Computer Science VI
Am Hubland, 97074 Würzburg, Germany
{pkluegl, atzmueller, puppe}@informatik.uni-wuerzburg.de

Abstract. This paper presents TEXTMARKER– a powerful toolkit for rule-based information extraction. TEXTMARKER is based on UIMA and provides versatile information processing and advanced extraction techniques. We thoroughly describe the system and its capabilities for human-like information processing and rapid prototyping of information extraction applications.

1 Introduction

Due to the abundance of unstructured and semi-structured information and data in the form of textual documents, methods for information extraction are key techniques in the information age. There already exist a variety of methods for information extraction. These can roughly be divided into machine learning and knowledge engineering approaches [1]: Often, machine learning approaches are applied for information extraction. However, in our experience rule-based techniques provide a viable alternative especially since these allow for rapid-prototyping capabilities, that is, by starting with a minimal rule set that can be extended as needed. Furthermore, modeling human-like information extraction and processing can be directly supported.

The model for the extraction of information in the hybrid knowledge engineering approach is either handcrafted or learned by covering algorithms and similar learning methods. For the representation of the white-box knowledge, often sequence labeling or classification approaches are used, e.g., finite state transducers, concept patterns, lambda expressions, logic programs or proprietary rule languages. Appelt [2] gives some reasons to prefer the knowledge engineering approach to the established black-box machine learning approaches that often use knowledge engineering for the feature extraction themselves. Additionally, the time spent on the annotation of a training corpus has to be compared to the effort of writing handcrafted rules. In some cases, there is even no machine learning method available that satisfies the required expressiveness for an optimal model.

The Unstructured Information Management Architecture (UIMA) [3] is a flexible and extensible architecture for the analysis and processing of unstructured data, especially text. The feature structures and the artifact (the text document and its annotations, respectively), are represented by the common analysis structure (CAS). Different components defined by descriptor files can process a CAS in a pipeline architecture.

Whereas 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 of UIMA concerns the concept of a type system that defines the different types used by a component. Types themselves define the concept of a feature structure and contain a set of features, more precisely references to other feature structures or simple values.

In this paper, we describe the TEXTMARKER system as a UIMA-based toolkit for rule-based information extraction. TEXTMARKER applies a knowledge engineering approach for acquiring rule sets and can be complemented by machine learning techniques. Due to its intuitive and extensible rule formalization language that also provides scripting-language like features, TEXTMARKER provides for a powerful toolkit for rule-based information extraction and processing. The knowledge formalization mechanism also supports meta-level information extraction by improving and applying its extraction rules incrementally. Technically, the system is firmly grounded on UIMA.

The rest of the paper is structured as follows: Section 2 provides a detailed introduction to the TEXTMARKER system and gives an overview on current research directions. Next, we describe the integration and combination of TEXTMARKER and UIMA in Section 3. After that, Section 4 provides an overview on applications using the TEXTMARKER system. Finally, Section 5 concludes the paper with a discussion of the presented work and proposes several directions for future work.

2 The TEXTMARKER System

The TEXTMARKER system[1] is a rule-based tool for the processing of unstructured information, especially for information extraction. It aims at supporting the knowledge engineer with respect to the rapid prototyping of information extraction applications and in providing the necessary elements for modeling human extraction knowledge and processes. The development environment is essential for the successful usage of a rule or scripting language and is therefore continually being improved. It is based on the DLTK framework[2] in order to provide a full-featured editor for the knowledge engineer. Components for the explanation provide statistics and related information, e.g., about how often the blocks and rules tried to apply and match, how often they succeeded, and how their conditions evaluated. In the following we provide a short introduction to the core concepts and language elements of TEXTMARKER. A more detailed description can be found in the TEXTMARKER project wiki[3].

2.1 The TEXTMARKER language

A rule file or *Script* in the language of the TEXTMARKER system mainly consists of three major parts (cf. table 1): A declaration of the package of the script (*PackageDecl*), that determines the namespace of newly defined types. *Import* elements make external components available, for example by referencing UIMA descriptors of Analysis

[1] The source code is available at https://sourceforge.net/projects/textmarker/

[2] http://www.eclipse.org/dltk/

[3] http://tmwiki.informatik.uni-wuerzburg.de/

Engines and Type Systems or other rule sets of the TEXTMARKER language. Five different kinds of a *Statement* can form the body of the script file. Whereas the *TypeDecl* creates new types of feature structures, *VariableDecl* defines a new variable for types, strings, booleans or numbers. The *ResourceDecl* loads external trie structures, word lists or tables that can be used by conditions and actions. The *Block* element groups other statements and provides some functionality commonly known in scripting languages.

A *Rule* consists of a list of rule elements that are made up of three parts: The mandatory matching condition of a rule is given by a *TypeExpr* or a *StringExpr* and creates a connection to the document. The optional *QuantifierPart* defines greedy or reluctant repetitions of the rule element, similar to regular expressions. Then, additional conditions and actions in the *ConditionActionPart* add further requirements and consequences to the rule element. Both elements, a *Condition* and an *Action*, require commonly several arguments, respectively expressions of types, numbers, strings or booleans or specialized constructs like assignments. An expression itself can be a terminal element, e.g., a number, variables containing a terminal element, a function returning that kind of element or combinations of them, e.g., a multiplication of two number expressions. The TEXTMARKER language currently provides 24 different conditions and 19 different actions and their number is constantly growing. The list of available conditions and actions can be extended by third parties in order to customize the TEXTMARKER system for specialized domains. In addition to elements that facilitate the rule engineering, external libraries or databases can enrich the TEXTMARKER language.

Script	→ PackageDecl Import* Statement*
Import	→ ('TYPESYSTEM' | 'SCRIPT' | 'ENGINE') Identifier ';'
Statement	→ TypeDecl | ResourceDecl | VariableDecl | Block | Rule
TypeDecl	→ 'DECLARE' (AnnotationType)? Identifier (',' Identifier)*
	| 'DECLARE' AnnotationType Identifier ('(' FeatureDecl ')')?
ResourceDecl	→ ('LIST'|'TABLE') Identfier ';'
VariableDecl	→ ('TYPE'|'STRING'|'INT'|'DOUBLE'|'BOOLEAN') Identifier ';'
Block	→ 'BLOCK' '(' Identifier ')' RuleElementWithType '{' Statement* '}'
Rule	→ RuleElement+ ';'
RuleElement	→ (TypeExpr | StringExpr) QuantifierPart? ConditionActionPart?
QuantifierPart	→ '*' | '*?' | '+' | '+?' | '?' | '??' | '[' NumberExpr ',' NumberExpr ']' ('?')?
ConditionActionPart	→ '{' (Condition (',' Condition)*)? ('->' Action (',' Action)*)? '}'
Condition	→ ConditionName ('(' Argument (',' Argument)* ')')?
Action	→ ActionName ('(' Argument (',' Argument)* ')')?
Argument	→ TypeExpr | NumberExpr | StringExpr | BooleanExpr | ...

Table 1. Extract of the TEXTMARKER language definition in Backus-Naur-Form.

The characteristics of the TEXTMARKER language are illustrated with two simple examples. In the first example, a rule with three rule elements is given that processes dates in a certain format, e.g., "Dec. 2004", "July 85" or "11.2008". The first rule element matches on a basic annotation of the type *ANY* (any token), if its covered text is

contained in a dictionary named *Month.twl*. An optional *PERIOD* can follow the word. Then, a *NUM* (number) annotation has to come next, that has between two and four digits. If all three rule elements matched, then a new *Month* annotation for the text matched by the first rule element, a *Year* annotation for the text matched by the last rule element and a *Date* annotation for the completely matched text are created.

```
ANY{INLIST(Months.twl) -> MARK(Month), MARK(Date,1,3)} PERIOD?
  NUM{REGEXP(".{2,4}") -> MARK(Year))};
```

The rule in the second example creates a new relation concerning an employment. The single rule element matches on all *Sentence* annotations that contain an annotation of the type *EmploymentIndicator*. Then, a feature structure of the type *EmplRelation* is created and the values of its features *EmployeeRef* and *EmployerRef* are assigned to an annotation of the type *Employee* and *Employer* located within the matched annotation. The sentence "Peter works for Frank", for example, has to be already annotated with the employee "Peter", the employer "Frank" and the employment indicator "works for" that determines the role of persons in that sentence.

```
Sentence{CONTAINS(EmploymentIndicator) -> CREATE(EmplRelation,
  "EmployeeRef" = Employee, "EmployerRef" = Employer)};
```

2.2 Rule Inference

The inference of the TEXTMARKER system relies on a complete, disjunctive partition of the document. A basic (minimal) annotation for each element of the partition is assigned to a type of a hierarchy. These basic annotations are enriched for performance reasons with information about annotations that start at the same offset or overlap with the basic annotation. Normally, a scanner creates a basic annotation for each token, punctuation or whitespace, but can also be replaced with a different annotation seeding strategy. Unlike other rule-based information extraction language, the rules are executed in an imperative way. Experience has shown that the dependencies between rules, e.g., the same annotation types in the action and in the condition of a different rule, often form tree-like and not graph-like structures. Therefore, the sequencing and imperative processing did not cause disadvantages, but instead obvious advantages, e.g., the improved understandability of large rule sets. Algorithm 1 summarizes the rule inference of the TEXTMARKER system. The rule elements can of course match on all kinds of annotations. Therefore the determination of the next basic annotation returns the first basic annotation after the last basic annotation of the complete, matched annotation.

2.3 Special Features

The TEXTMARKER language features some special characteristics that are not found in other information extraction systems. The expressiveness of the TEXTMARKER language is increased by elements commonly known in scripting languages. Conditions and actions can refer to expression, respectively variables. They either evaluate or modify the value of the expression. As a consequence, a rule can alter the behavior of other

Algorithm 1 Rule Inference Algorithm

```
collect all basic annotations that fulfill the first matching condition
for all collected basic annotations do
  for all rule elements of current rule do
    if quantifier wants to match then
      match the conditions of the rule element on the current basic annotation
      determine the next basic annotation after the current match
      if quantifier wants to continue then
        if there is a next basic annotation then
          continue with the current rule element and the next basic annotation
        else if rule element did not match then
          reset the next basic annotation to the current one
      set the current basic annotation to the next one
      if some rule elements did not match then
        stop and continue with the next collected basic annotation
      else if there is no current basic annotation and the quantifier wants to continue then
        set the current basic annotation to the previous one
  if all rule elements matched then
    execute the actions of all rule elements
```

rules, e.g., by changing the type of the created annotations. The block element introduces the functionality of procedures, conditioned statements and loops. The identifier of the block element defines the name of the procedure and is used in the invocation by other rules. The single rule element in the definition of the block element creates a local view of the inner statements of the document. If the type expression of the rule element refers to a type that occurs several times in the current view on the documents, then the inner statements are executed on each text fragment defined by these annotations. The conditions of the rule elements add additional requirements, that need to be fulfilled before the inner statements are processed. Therefore a match on the complete document with additional conditions equals an if statement.

Some actions can modify the view on the document by filtering or retaining certain types of annotations or elements of the markup. For reasons of convenience, handcrafted rules are often based on different assumptions. Therefore unintended token classes, e.g., markup or whitespace, or arbitrary types of annotations can be removed from the view on the document. The knowledge engineer is able to retain the important features and increase the robustness of the extraction process.

Sometimes it is rather difficult to capture all relevant features for a type in a single rule. In addition to the creation of annotations, it is possible to add a positive or negative score: The features can thus be distributed among several rules that each weight their impact on the type with a heuristic score. The annotation is not created until enough rules have fired and the heuristic value has exceeded a defined threshold. The set of rules is even more robust, since the absence of a few features can be neglected.

Beside its primary task to extract information, the input document can also be modified, e.g., for anonymization. For this purpose some actions are able to delete, replace or color the text of the matched text fragment. The actual change will be performed by another analysis engine that creates a new view containing the modified document.

2.4 Research Directions

The formalization of matching rules is usually difficult and time consuming. Often, machine learning methods can support the knowledge engineer in a semi-automatic process: Annotated documents form the input for rule learning methods complementing the handcrafted rules. Since the knowledge engineer usually has deep insights in the domain and the learning task, appropriate acquisition methods and their parameters can be selected. There are several options for utilizing the learned rules: The acquired rules can be modified and transferred, if the quality of the proposed rules is good enough. If this is not the case, then the settings of the learning task can be adapted by annotating more examples or by changing the applied methods and their parameters. However, the process can also be restarted by continuing with the next concept. Our integrated framework for this process [4] currently contains prototypes of four learning methods, for example LP^2 [5], and is extended with well known and with more specialized methods in future. Since the methods are used in a semi-automatic process, the accuracy of extraction process is not as important as the comprehensibility, the extensibility, the capabilities of system integration, the usage of features and the overall result.

According to the common process model in information extraction, features are extracted from the input document and are used by a model to identify information. But using already extracted information for further information extraction can often account for missing or ambiguous features and increase the accuracy in domains with repetitive structure. If the document was written, for example, by a single author, often the same layout for repetitive structures is used. If the corpus contains documents by different authors and these authors used different layout styles, then the relation between the features and information is contradictory. By identifying a confident information and analyzing its features, meta-features can be created that describe the relation of the information and its features for the current document. These meta-features and the transfer knowledge that projects the meta-features to other text fragments form a dynamic layer of the model. Our meta-level information extraction approach [6] engineers parts of the human processing of documents and is able to considerably increase the accuracy of an information extraction application.

2.5 Related Systems

The JAPE [7] system also applies patterns on annotations, uses a textual knowledge representation, but utilizes finite state machines for rule inference; the integration of additional java code is possible. While JAPE is not based on UIMA, an integration is enabled using a UIMA-bridge.

LANGUAGEWARE[4] is a comprehensive linguistic platform for semantic analysis that is also embedded into UIMA. It provides an integrated development environment with real-time testing capabilities, and several configurable components, e.g., for dictionary lookup, language identification, syntactic and semantic analysis, or entity and relation extraction. In contrast to TEXTMARKER the rule construction is performed using a drag-and-drop paradigm. The rule inference is strongly based on conceptual text

[4] http://www.alphaworks.ibm.com/tech/lrw

structures like sentences and therefore especially useful for the processing of unstructered documents. TEXTMARKER applies a different paradigm, i.e., similar to a scripting language implemented using rules. Therefore, TEXTMARKER directly supports several tasks directly 'out of the box' that would otherwise be implemented using separate UIMA components.

3 TEXTMARKER and UIMA

The development environment of the TEXTMARKER system provides a build process for the automatic creation of UIMA descriptors. For each rule file, an analysis engine descriptor and a type system descriptor are created that include all dependencies and types of referenced rule files. For the analysis engine a generic descriptor is extended that allows to configure all generated descriptors in the project. The availability of generated descriptors eases the integration of TEXTMARKER components in other UIMA pipelines for various tasks, like feature construction, document modification or information extraction. UIMA elements can directly be processed by the given rules, for example the values of features can be transferred between compatible feature structures. Arbitrary UIMA type systems and analysis engines can be used directly in a rule file by importing the descriptors and executed by an action (in the case of an analysis engine). In doing so, a new document is created with the current filtering settings: The HTML elements of semi-structured documents, for example, are removed before a part-of-speech tagger is executed on that dynamic document. Then, the newly created annotations with their offsets are transferred to the original document and can be used by TEXTMARKER rules. There is ongoing work to integrate several component repositories like DKPRO [8] and advanced machine learning toolkits like CLEARTK [9] directly into the TEXTMARKER development environment in order to provide a simple usage of linguistic approaches and arbitrary components with the complete expressiveness of the system.

4 Experiences

Although the TEXTMARKER system is still in an early project state, it was already successfully applied in several projects, especially for semi-structured documents. For the task of creating structured data of work experiences in curricula vitae, more than 10000 documents with heterogenous layouts and structures were successfully processed. Feature structures containing the complete description, the exact start date and end date, the employer and title amongst other annotations are extracted. The TEXTMARKER system uses large dictionaries together with advanced approaches to reproduce the human perception of text fragments. In order to apply data mining on medical discharge letters, the TEXTMARKER first anonymizes and then partitions the document in different fragments for diagnoses, therapies, oberservations and so forth. These sections are further processed for the acquisition of structured data. The CASETRAIN[5] project is also using the TEXTMARKER system for its authoring component. Structured documents for the creation of e-learning cases are parsed and an error feedback for the authors is created.

[5] http://casetrain.uni-wuerzburg.de

5 Conclusions

In this paper, we have presented the TEXTMARKER system as a UIMA-based tool for rule-based information extraction. Furthermore, we also have discussed the tight integration of TEXTMARKER within UIMA, and we have outlined several current research directions for TEXTMARKER that provide for further advanced information processing and extraction techniques. With its intuitive and extensible rule formalization language, TEXTMARKER provides for a powerful toolkit for the UIMA community, and is not limited to certain domains, but open for various applications.

For future work, we aim to perform several improvements, e.g., embedding more UIMA structures into the language (lists and arrays), and to add more language elements, e.g., new expressions, conditions and actions. Additionally, we plan to further improve the development environment. One of the most important goals is the extension of the rapid-prototyping and rule learning capabilities that will also be utilized for the (automatic) acquisition of meta knowledge. We also aim for a combination with other approaches, e.g., advanced machine learning techniques or text mining methods. For example, textual subgroup mining [10] is an interesting complement, e.g., for supporting rule formalization.

Acknowledgements

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

References

1. Turmo, J., Ageno, A., Català, N.: Adaptive Information Extraction. ACM Comput. Surv. **38**(2) (2006) 4
2. Appelt, D.E.: Introduction to Information Extraction. AI Commun. **12**(3) (1999) 161–172
3. Ferrucci, D., Lally, A.: UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. Nat. Lang. Eng. **10** (2004) 327–348
4. Kluegl, P., Atzmueller, M., Puppe, F.: A Framework for Semi-Automatic Development of Rule-based Information Extraction Applications. In: Proc. LWA 2009 (KDML - Special Track on Knowledge Discovery and Machine Learning). (2009) (accepted).
5. Ciravegna, F.: $(LP)^2$, Rule Induction for Information Extraction Using Linguistic Constraints. Technical Report CS-03-07, University of Sheffield, Sheffield (2003)
6. Kluegl, P., Atzmueller, M., Puppe, F.: Meta-Level Information Extraction. In: The 32nd Annual Conference on Artificial Intelligence, Berlin, Springer (September 2009) (accepted).
7. Cunningham, H., Maynard, D., Tablan, V.: JAPE: A Java Annotation Patterns Engine (Second Edition). Research Memorandum CS–00–10, University of Sheffield (November 2000)
8. Gurevych, I., Müller, M.C.: Information Extraction with the Darmstadt Knowledge Processing Software Repository (Extended Abstract). In: Proceedings of the Workshop on Linguistic Processing Pipelines, Darmstadt, Germany (Jul 2008)
9. Ogren, P.V., Wetzler, P.G., Bethard, S.: ClearTK: A UIMA Toolkit for Statistical Natural Language Processing. In: UIMA for NLP workshop at LREC 08. (2008)
10. Atzmueller, M., Nalepa, G.J.: A Textual Subgroup Mining Approach for Rapid ARD+ Model Capture. In: Proc. 22nd Intl. Florida AI Research Soc. Conf. (FLAIRS), AAAI Press (2009)