Test-Driven Development of Complex Information Extraction Systems using TEXTMARKER

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. Information extraction is concerned with the location of specific items in textual documents. Common process models for this task use ad-hoc testing methods against a gold standard. This paper presents an approach for the test-driven development of complex information extraction systems. We propose a process model for test-driven information extraction, and discuss its implementation using the rule-based scripting language TEXTMARKER in detail. TEXTMARKER and the test-driven approach are demonstrated by two real-world case studies in technical and medical domains.

1 Introduction

There are two main paradigms for information extraction approaches: Either the systems provide automatic extraction techniques, i.e., based on machine-learning techniques, or they are based on manually acquired knowledge, mainly considering rule-based knowledge. The first approach allows an easy application of the system since no knowledge acquisition phase is required, and the extraction knowledge can be automatically learned from annotated examples. However, the latter often performs better for rather complex domains. There are several reasons to prefer a knowledge engineering approach to a machine learning approach [1]: The knowledge can usually be captured and extended quite easily by a domain specialist and therefore provides flexible techniques if the requirements change, for extensions, and for exceptions for certain documents.

In this paper, we present an approach for the test-driven development of complex text extraction systems using the rule-based scripting language TEXTMARKER: We discuss a process model for the test-driven extraction process, and discuss the contained steps in detail. Furthermore, we describe the TEXTMARKER system for rule-based information extraction. After that, we provide two case studies for demonstrating and discussing the presented approach in detail.

The rest of the paper is structured as follows: Section 2 presents the process model for the test-driven development of complex text extraction systems using TEXTMARKER. Section 3 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 4 discusses two real-world case-studies for the presented approach. Finally, Section 5 concludes with a discussion of the presented work, and provides several interesting options for future work.

2 Test-Driven Process Model

In the following section, we describe the process model for test-driven development of complex text extraction systems using TEXTMARKER. We distinguish two roles, the knowledge engineer and the domain specialist. The latter is concerned with the annotation, selection, and formalization of documents/test cases, whereas the former performs the development and incremental refinement of the rule base for text extraction. The process is shown in Figure 1, and discussed below.

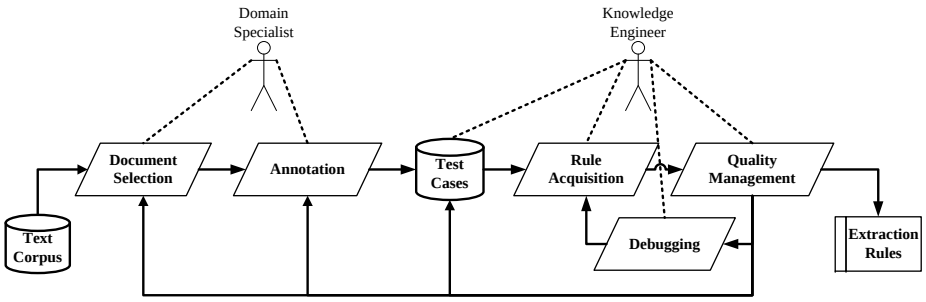

Fig. 1. Process Model: Semi-Automatic Rule-Based Instance Generation from Texts

- **Document Selection**: The document selection steps provides a subset of the cases contained in the text corpus that are applied as test cases later. Synthetic cases can also be added (as test cases), therefore new cases can also be easily integrated for coping with new requirements. For document selection also cluster techniques can be applied in order to select a limited, heterogeneous and diverse set of cases.
- **Annotation**: During the annotation step, the domain specialist selects text fragments and assigns pre-specified types (i.e., annotations) to these. Each added annotation is either an input type (pre-specified information) or an output type (expected result). Input annotations are used for testing specific features/rules in their context (unit-tests). Output types are the *expected* types of the test. Visual editors allow the user to annotate the text interactively.
- **Rule Acquisition**: During the rule acquisition step new extraction rules are formalized, tuned and refined according to the given test corpus based on the results of the subsequent quality management and debugging steps.
- **Quality Management**: In the quality management step, the knowledge engineer assesses the correctness and completeness of the system based on the set of formalized test cases. Also, erroneous test cases that were marked during the quality management step can be incrementally corrected.
- **Debugging**: The debugging step is mainly concerned with debugging specific rules. The information about rule applications and their contexts can significantly help for the improvement of the rule base and also for correcting test cases.

3 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 current TEXTMARKER implementation is based on a prototype described in [2] that supports a subset of the TEXTMARKER language described below. The present TEXTMARKER system is currently being extended towards an integration as a UIMA (*Unstructured Information Management Architecture*) component [3]. 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 a input format, since most word processing documents can be easily converted to HTML.

In the following sections we first give a conceptual overview on the TEXTMARKER language by introducing its core concepts. After that, we discuss the syntax and the semantics of the TEXTMARKER language in detail, and provide some illustrating examples. Next, we present special characteristics of the language that distinguishes the TEXTMARKER system from other common rule based information extraction systems.

3.1 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 manually created 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 2 shows an excerpt of a basic annotation taxonomy: For example, *CW* describes all tokens, that contain a single word starting with a capital letter, *MARKUP* corresponds to HTML/XML tags, *ANY* combines all symbols that are not classified as *MARKUP* 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.

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

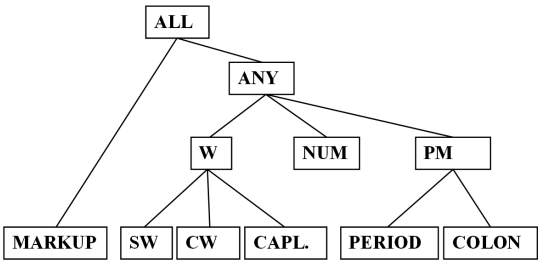

Fig. 2. Part of a taxonomy for basic types. (W=Word, NUM=Number, PM=Punctuations, SW=Word without capitals, CW=Word starting with a capital letter)

3.2 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 3.2 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 considers 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).

```
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.

3.3 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 four types of input in this example (see [2]):

```
DEFAULT 'Dr' PERIOD CW CW;

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

Heuristic extraction using scoring rules Diagnostic scores [4] 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 indicated 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, rules, declarations or blocks of rules are only executed if an annotation of the type *headlinedParagraph* is located in the text 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 and the rules do not need to be applied on the complete document.
- **Method calls:** Another common feature is the declaration and the reference 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.

These efforts for extending the TEXTMARKER language towards a scripting language was one of the reasons to replace the existing and successful development environment [2]. The new implementation is built on the Dynamic Language Toolkit[2] in order to support the described scripting functionality in the development process.

In test driven development automatic test cases are ideally written for small, atomic units [5]. The smallest unit of a TEXTMARKER scripting file is a single rule. However, with the integration in UIMA, the interfaces of a UIMA component and the information structure [6] are especially suitable for the specification of test cases. The TEXTMARKER system is integrated in UIMA as a component, i.e. an *Analysis Engine* that provides the functionality of including several scripting files. If the functionality of a large rule set is split into several modular scripting files with internal block structure, then it is still possible to create small and self-contained *testing units*. Therefore the test cases are specified independently of the TEXTMARKER implementation by using the UIMA interfaces, but they can still refer to specific parts of functionality, especially single block definitions of a TEXTMARKER scripting file.

4 Case Studies

In the following sections we describe two real-world case studies applying parts of the presented approach. The first case study is concerned with high-level information extraction in a technical domain. The second case study considers the generation of structured data records given semi-structured medical discharge letters.

4.1 High-Level Information Extraction

The case study is about a high level information extraction, automatic content segmentation and extraction task. Unfortunately, we can only describe the case study in a very general way due to non-disclosure terms. As a general setting, word processing documents in common file formats[3] like Microsoft Word or OpenOffice are mined for information specific to certain projects, with temporal margins, e.g., similar to curricula vitae. The input documents feature an extremely heterogeneous layout and are each written by a different person. Interesting text fragments may relate from plain text to structured tables or even combinations or parts of them. Additionally, the layout is not sufficient enough for a correct classification, since also domain dependent semantics may change the relevance of a fragment. The output of a document are a set of templates that contain exact temporal information, the exact text fragment related to the template and various domain specific information, e.g., a project name or company names in our curriculum vitae example.

Although the application is still under development, it already involves 479 rules and several domain specific dictionaries with up to 80000 entries. Basically, the TEXTMARKER system tries to imitate the human perception of text blocks. For this purpose interesting named entities, e.g., temporary information, are recognized. Then, the application locates text structures of different types of complexity and size, e.g., a headlined

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

[3] The input documents are converted to HTML

paragraph or a row of a table. If one of these text fragments or a set of text fragments of the same type contains a significant pattern of interesting named entities, then they are marked as a relevant block of text. Finally, additional rules are used to detect the domain specific information which is also used to refine the found segments. In the current state the TEXTMARKER application was evaluated on correct text fragments and temporal data only. It achieved an F1 measure of 89% tested on 58 randomly selected documents with 783 relevant text fragments. These results seem to indicate potential for further improvements, however, in order to obtain more reliable results we need to perform more evaluations together with our project partners first.

The development of the application used a process model similar to the common model with ad-hoc testing. Normally, an information extraction application is tested automatically for quality assurance. But due to the characteristics of the high level information extraction task, it is often not suitable to utilize complete annotated documents for back testing. Therefore a semi-automatic approach with several supporting tools was used. The applied process works as follows: At the beginning a new application or a new requirement is defined by the domain specialist. He or she manually selects a representative set of documents and creates a test corpus. The knowledge engineer develops new rules using the test corpus and informal specifications. The domain specialist tests the new rules with the test documents for their functionality. Then he or she creates a feedback document, a documentation of the errors with examples. Furthermore, the new rules are additionally tested on a new test corpus with randomly selected documents. The feedback document is extended with the new reported errors. The knowledge engineer writes new rules to correct the documented errors. If the functionality or the quality of the rules is not sufficient enough, the process is iterated: Either new features are added or the rule set has to be improved further. In both possibilities the knowledge engineer receives a new representative corpus for testing.

The experience with this application motivated the development of the presented test-driven process model. The process has already been partially implemented, and especially the controlled formalization of test cases, the isolated specification of new features and the automatic back testing of different kinds of test cases provide distinct advantages over the current ad-hoc testing process model.

4.2 Diagnostic Case Extraction from Textual Discharge Letters

The second case study considers the generation of cases from semi-structured medical discharge letters. These letters are written by the physicians when a patient has been diagnosed and leaves after a hospital stay. The letters are typically written by the responsible physicians themselves and are stored as MS Office (Word) documents. These contain the observations, for example, the history of the patient, results from certain examinations, measurements of laboratory parameters, and finally the inferred diagnoses of the patient. Figure 3 shows an example of a partial (anonymized) discharge letter with the diagnosis, anamnesis, and some laboratory values. The available electronic discharge letters provide the basis for various purposes, for example, for quality control with respect to a hospital information system, for medical evaluations, or for creating case-based training sessions.

Diagnosen:
Leberzirrhose ethyltoxisch CHILD C
Therapierefraktärer Aszites (chylös)
Indikation zur TIPSS-Anlage
Indikation zur Lebertransplantation
Z. n. Ösophagusvarizenblutung
Z. n. spontan bakterielle Peritonitis
Z. n. Prostataektomie bei Prostata-Karzinom.
Hyperplastischer Magenpolyp
Z. n. Polypektomie eines Colon-Polypens an der Bauhin'schen Klappe.

Anamnese:
Herr X wurde uns mit therapierefraktärem Aszites bei Leberzirrhose vorgestellt. Auch unter gesteigerter Diuretika-Dosierung waren Aszitespunktionen in kurzen Abständen notwendig. Herr X wurde nun zur erneuter Aszitespunktion und Neueinstellung der medikamentösen Therapie stationär aufgenommen. Des Weiteren sollte eine Gastroskopie bei bekannten hyperplastischen Magenpolypen durchgeführt werden. Der Patient berichtete über eine Gewichtszunahme von 6 kg innerhalb von einer Woche. Die Trinkmenge läge z. Z. bei 2,5 – 3 Liter pro Tag. Wegen zahlreicher Nebenwirkungen wurde die aktuelle Medikation in Rücksprache mit unterer gastroenterologischen Ambulanz abgesetzt. Zuletzt nahm der Patient an Torasemid 20 mg und Spironolacton 200 mg täglich ein.

Labor:
(XX.XX.20XX 10:20:00)
Klinische Chemie: ***Eisen: 38 [59 - 158] µg/dl;***
Gerinnung: ***Thromboplastinzeit n. Quick: 44 [70 - 130] %; Ratio int. norm.: 1.63 [0.85 - 1.18] ; PTT: 64.1 [23 - 36] s; Antithrombin III: 27 [75 - 125] %;*** Fibrinogen (Clauss): 2.6 [1.8 - 3.5] g/l; ***Faktor II: 27 [70 - 120] %; Faktor V: 27 [70 - 140] %; D-Dimere (immunol.): 0.420 [0 - 0.190] mg/l;***
Serumproteine und Tumormarker: Ferritin: 30 [30 - 400] µg/l; ***Transferrin: 169 [200 - 380] mg/dl; Transferrinsättigung: 15.9 [16 -***

Fig. 3. Example of a partial discharge letter (in german): The screenshot shows the diagnoses, anamnesis, and laboratory examination part ("Diagnosen, Anamnese, Labor"). Then, the segments corresponding to these need to be extracted, and post-processed for data extraction.

The text corpus is made up of a set of discharge letters for a set of patients. The goal is to process these and to extract the relevant information (observations, diagnoses) from the discharge letters. We started with a training corpus of 43 discharge letters. For extracting the relevant information, we developed a set of rules that take the structure of the document into account. A discharge letter needs to follow a certain standard structure: The document is started by the salutation, the diagnosis part, the history of the patient, textual paragraphs describing the results of various examinations like computer tomography (CT), and the result of laboratory examinations, i.e., the measured parameters. For applying the TEXTMARKER system, we can therefore focus on these building blocks of the document. Therefore, the domain specialist provided this information and annotated several documents concerning the important text blocks, and the respective concepts. Each of the documents contained in the test corpus was annotated with the concepts that are mentioned in the document. In this way, we developed a set of rules for extracting segments of the letter first, for example, considering the diagnosis block. After that, those segments were split up further, for example, considering the fact that individual diagnoses are almost always contained in separate lines.

The corpus is still being extended, and new diagnoses and observations are being added to the set of important concepts. Therefore, this provides for an ideal option for further applying and testing the presented approach. The new concepts can be integrated and captured with new rules, and their application can be debugged in context using the new framework.

5 Conclusions

Information extraction is part of a widespread and still growing scientific community that originates a multiplicity of new systems, tools and approaches. The initial development of the TEXTMARKER system was influenced by the LAPIS system [7] and the LIXTO SUITE [8] with its LIXTO VISUAL WRAPPER.

Test-driven development is a well known and successfully applied development strategy. It is often combined with agile methods like extreme programming and is supported by an automatic testing framework. Test-driven development is not only a test first approach for quality management, but also for the analysis and design process [5].

Various studies have shown that test-driven development reduces the defect rate and detects defects earlier. Maximilien et al. [9] have shown a reduction of defect by 50 percent compared to an ad-hoc unit testing approach. Baumeister et al. [10] applied automatic tests and restructuring methods for an agile development of diagnostic knowledge systems. They defined different types of tests, e.g., on correctness, anomalies or robustness, and noticed significant improvements for the evolutionary development.

In the area of text mining and information extraction, ad-hoc testing against a hand annotated *gold standard* is common practice. The tool *CFE* (Common Feature Extraction) [11] is a system for testing, evaluation and machine learning of UIMA based applications. It provides the declarative language *Feature Extraction Specification Language* (FESL) that is interpreted and executed by a generic UIMA component. However, to the best knowledge of the authors, there is no prominent tool that supports a test-driven development of information extraction applications beyond common back testing. The strategy of test-driven development can be used for the development of complex information extraction applications. Yet, the transfer is not straight forward for common rule-based or pattern-based tools. The test specification and the test framework has to incorporate the imprecise nature of the unstructured information domain.

In this paper, we have presented a test-driven approach for the development of complex text extraction systems using TEXTMARKER. We have proposed a process model for the discussed task, and we have introduced the necessary components and features of TEXTMARKER in detail. Additionally, we have discussed two real-world case studies for exemplifying the presented approach.

The test-driven strategy is being integrated in our case studies. We expect a significant improvement in the development in general and especially in defect detection, defect reduction and an accelerated development. The process model described in this paper has some prominent features that are not found in the common development strategy. The presented process model supports an incremental development with minimal initial test cases. Real world test cases that define the common requirements can be combined with synthetic test cases for specific features and quality management. Furthermore, the proposed test-driven development process contains short iterations of different steps and provides a flexible way to create complex information extraction applications. The debugging step combines the advantages of the rule-based approach of the language, the powerful integrated development environment and the information contained in the test cases. Therefore detailed debugging information about the rule applications, the matched text and the conditions of each rule elements can explain each occurred error. The common *red-green* metaphor of the automatic testing frameworks is

therefore extended, because the test information can be displayed in combination with debugging information directly in the textual document of the test case.

In the future, we plan to consider one major part of test-driven development that was not addressed yet: Refactoring techniques for TEXTMARKER scripts in order to further enhance the user experience and applicability of the presented approach. Additionally, we aim to integrate machine learning techniques, e.g., knowledge-intensive subgroup discovery methods [12], for a more semi-automatic development approach.

Acknowledgements

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

References

1. Appelt, D.E.: Introduction to Information Extraction. AI Commun. **12**(3) (1999) 161–172
2. von Schoen, P.: Textmarker: Automatische Aufbereitung von Arztbriefen für Trainingsfälle mittels Anonymisierung, Strukturerkennung und Teminologie-Matching [TextMarker: Automatic Refinement of Discharge Letters for Training Cases using Anonymization, Structure- and Terminology-Matching]. Master's thesis, University of Wuerzburg (2006)
3. Ferrucci, D., Lally, A.: UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. Nat. Lang. Eng. **10**(3-4) (2004) 327–348
4. Puppe, F.: Knowledge Formalization Patterns. In: Proc. PKAW 2000, Sydney, Australia (2000)
5. Janzen, D., Saiedian, H.: Test-Driven Development: Concepts, Taxonomy, and Future Direction. Computer **38**(9) (2005) 43–50
6. Götz, T., Suhre, O.: Design and Implementation of the UIMA Common Analysis System. IBM Syst. J. **43**(3) (2004) 476–489
7. Kuhlins, S., Tredwell, R.: Toolkits for Generating Wrappers – A Survey of Software Toolkits for Automated Data Extraction from Web Sites. In Aksit, M., Mezini, M., Unland, R., eds.: Objects, Components, Architectures, Services, and Applications for a Networked World. Volume 2591 of Lecture Notes in Computer Science (LNCS)., Berlin, International Conference NetObjectDays, NODe 2002, Erfurt, Germany, 2002, Springer (2003) 184–198
8. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: The VLDB Journal. (2001) 119–128
9. Maximilien, E.M., Williams, L.: Assessing Test-Driven Development at IBM. In: ICSE '03: Proceedings of the 25th International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2003) 564–569
10. Baumeister, J., Seipel, D., Puppe, F.: Using Automated Tests and Restructuring Methods for an Agile Development of Diagnostic Knowledge Systems. In: FLAIRS'04: Proc. 17th Intl. Florida Artificial Intelligence Research Society Conference. (2004) 319–324
11. Sominsky, I., Coden, A., Tanenblatt, M.: CFE - a System for Testing, Evaluation and Machine Learning of UIMA based Applications. In: LREC '08: The sixth international Conference on Language Resources and Evaluation. Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP. (2008)
12. Atzmueller, M., Puppe, F., Buscher, H.P.: Exploiting Background Knowledge for Knowledge-Intensive Subgroup Discovery. In: Proc. 19th Intl. Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, Scotland (2005) 647–652