Integrated Tools for Query-driven Development of Light-weight Ontologies and Information Extraction Components

Martin Toepfer[1] **Georg Fette**[1] **Philip-Daniel Beck**[1] **Peter Kluegl**[12] **Frank Puppe**[1]

[1]Department of Computer Science VI
University of Würzburg, Am Hubland
Würzburg, Germany
first.last@uni-wuerzburg.de

[2]Comprehensive Heart Failure Center
University of Würzburg, Straubmühlweg 2a
Würzburg, Germany
pkluegl@uni-wuerzburg.de

Abstract

This paper reports on a user-friendly terminology and information extraction development environment that integrates into existing infrastructure for natural language processing and aims to close a gap in the UIMA community. The tool supports domain experts in data-driven and manual terminology refinement and refactoring. It can propose new concepts and simple relations and includes an information extraction algorithm that considers the context of terms for disambiguation. With its tight integration of easy-to-use and technical tools for component development and resource management, the system is especially designed to shorten times necessary for domain adaptation of such text processing components. Search support provided by the tool fosters this aspect and is helpful for building natural language processing modules in general. Specialized queries are included to speed up several tasks, for example, the detection of new terms and concepts, or simple quality estimation without gold standard documents. The development environment is modular and extensible by using Eclipse and the Apache UIMA framework. This paper describes the system's architecture and features with a focus on search support. Notably, this paper proposes a generic middleware component for queries in a UIMA based workbench.

1 Introduction

According to general understanding, a specification of relevant concepts, relations, and their types is required to build Information Extraction (IE) components. Named Entity Recognition (NER) systems in the newspaper domain, for example, try to detect concepts like persons, organizations, or locations. Regarding clinical text, it has been shown that lookup-based approaches (Tanenblatt et al., 2010) can achieve high precision and recall if a terminology exists that maps terms to their meanings. However, this approach is not directly applicable if such resources are not available for a certain language or subdomain, or if the domain and its terminology are changing. Unsupervised methods can help to find and to group the relevant terms of a domain, for example, into concept hierarchies (Cimiano et al., 2005). Nevertheless, automatically derived terminologies are not perfect, and there are many applications that require high precision knowledge resources and representations, for instance, to build up a clinical data warehouse. In this case, automatically generated ontologies have to be refined by domain experts like clinicians, which imposes special requirements on the usability of the tools.

There have been several efforts to support ontology extraction and refinement (Cimiano and Völker, 2005), predominantly with text processing based on the GATE (Cunningham et al., 2011) infrastructure. In the Apache UIMA (Ferrucci and Lally, 2004) community[1], several tools exist that ease system development and management. Much work has, for instance, been spent on pipeline management, rule development (Kluegl et al., 2009), or evaluation. Terminology and ontology development support, however, have not gained as much attention in the context of this framework by now. This is surprising since the integration of tools for terminology development and especially terminology generation and information extraction into existing infrastructure for text processing is promising. Actually, the approach

[1] http://uima.apache.org/

Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT, pages 83–92,
Dublin, Ireland, August 23rd 2014.

taken in this paper regards terminology creation and information extraction as two related tasks. The proposed system aims to assist users in the development of components that extract information with lexical resources gathered during the specification of the concepts of the domain.

This paper introduces *Edtgar*: a user-friendly integrated terminology development environment. It provides many features that help domain experts to construct and refine light-weight ontologies driven by flexible corpus queries. In this work, "light-weight ontology" means that we focus on simple relations, as well as restricted inference. We call the knowledge representation "terminology" since the tool aims to manage lexical information for information extraction. The major components of the system are a terminology editor, a plug-in concept for terminology extraction and an information extraction API, as well as support for corpus queries. Special views show extraction statistics, provide semi-automatic annotation of gold standard documents, as well as evaluation and deployment support. The tool comprises an implementation for terminology induction and an information extraction algorithm that considers contexts. In order to keep the system modular and extensible, it integrates into Eclipse[2] and uses the Apache UIMA framework. Apache UIMA provides a well-established framework for text processing, hence, a variety of natural language processing components can easily be integrated, for example, by accessing component repositories like DKPro Core[3]. At the technical level, the default processing components of the proposed system use a combination of Apache UIMA Ruta[4] scripts and custom analysis engines implemented in Java. As a consequence, the tight integration of the terminology development tools into Apache UIMA's Eclipse Tools and Apache UIMA Ruta's rule engine and workbench allows technical engineers to use several existing features.

The structure of the paper is as follows: Section 2 gives a brief overview of ontology development systems and tools. Section 3 and 4 introduce the tool and its support for corpus queries. Results of a case study are given in Section 5. Finally, we conclude in Section 6.

2 Related Work

Most of all, our work relates to environments and frameworks for ontology learning, editing, and refinement. We first give a brief overview of such systems with a focus on open source and research related systems[5]. Afterwards, we discuss some popular query tools that come into question for integration into natural language processing environments.

OntoLT (Buitelaar et al., 2004) is a plugin for the ontology editor Protégé. It aims to derive ontological concepts and relations from plain text by defining XPath expressions over linguistic structures, e.g., subject object relations under constraints like specific lemmas. Manual annotation of ontology concept mentions can be performed with the Protégé plugin Knowtator[6].

Very similar to our work is the *NeOn toolkit*[7]. It is an Eclipse based ontology development environment. There are many plugins available that extend NeOn toolkit's functionality. For instance, the GATE Webservice plugin[8] and its TermRaider component automatically generate ontological information. One of the plugins for NeOn is the work by Cimiano and Völker, who proposed *Text2Onto*[9] (Cimiano and Völker, 2005), which is a framework that allows to apply ontology learning and change discovery algorithms. Its central data structure is called probabilistic ontology model (POM). It is not providing statistics but stores values that represent concept or relation extraction certainty. Text2Onto's natural language processing is based on GATE and the rule engine JAPE. Similar to our work, Text2Onto aims to provide an easy-to-use user interface.

Most of the tools mentioned above either use GATE or proprietary data formats for linguistic representations. We believe that there is a need for a tool based on UIMA. Our tool aims to provide an integrated

[2] http://eclipse.org/

[3] https://www.ukp.tu-darmstadt.de/software/dkpro-core/?no_cache=1

[4] https://uima.apache.org/ruta.html

[5] There are related commercial solutions, for instance, http://www.poolparty.biz/

[6] http://knowtator.sourceforge.net/

[7] http://neon-toolkit.org

[8] http://neon-toolkit.org/wiki/Gate_Webservice

[9] https://code.google.com/p/text2onto/

development environment that includes terminology development and allows to use already available features provided by Eclipse plugins related to UIMA. Thereby, we ease development, introspection, debugging, and testing of all kinds of UIMA annotation engines involved in system and terminology development. Hence, the components that extract and refine terminologies based on natural language processing can easily be adapted, for example, segmentation rules written in UIMA Ruta's workbench. The same argumentation applies to information extraction components or linguistic preprocessors like chunkers that are involved in both tasks. Contrary to most other tools, we do not consider complex relations. Instead, we focus on easy-to-understand inference and relations.

In the UIMA community, Fiorelli et al. proposed the computer-aided ontology development architecture (CODA) (Fiorelli et al., 2010) that consists of the tasks: ontology learning, population of ontologies, and linguistic enrichment. Fiorelli et al. describe how an integrated ontology development system based on UIMA could look like, however, their system *UIMAST* concentrates on ontology population aspects. By contrast, this paper also considers construction tasks and describes tooling for editing and refining light-weight ontologies either manually or based on document collections.

Table 1 compares qualitative features of different query tools which are described below. Since semantic search is an active field of research, we can only give a brief overview focused on popular tools for UIMA and GATE. We put special emphasis on tools that can be easily integrated into an Eclipse-based environment.

Tool	Framework	Index	Syntax	IDE Integration
Ruta Query View	UIMA	no[a]	expert	Eclipse
Lucas / Lucene	UIMA	yes	user-friendly	-
GATE Mímir	GATE	yes	medium	-
GATE Annic	GATE	yes	medium	GATE Developer[b]

[a] uses only UIMA's annotation index

[b] `https://gate.ac.uk/family/developer.html`

Table 1: Qualitative comparison of query tools.

The UIMA Ruta workbench contains a *query view* that enables to search in document collections with arbitrary queries formulated as rules. For instance, `Segment{-CONTAINS(CONCEPT)}` matches segment annotations that do not contain any concept annotation. With regard to usability for domain experts, this tool has a drawback: users have to be familiar with Ruta's syntax which is in general too complex for terminology engineers. They should not have to learn a programming language to pose a query on a corpus. Another drawback of the query view is that it has no option to group search results, e.g. by their covered text. The Ruta query view is not designed for fast results on very large corpora. It iteratively processes all documents of a folder that match a user-defined filename filter, thus, queries do not run as fast as with index structures for the whole corpus. A combination of rule-based query formulation and the new middleware (Section 4) would be useful for the community.

Apache Lucene [10] is a popular search engine. Existing mapping tools like the UIMA Lucene indexer *Lucas*[11] show how UIMA pipeline results can be indexed with Lucene. This solution is attractive since Lucene's query syntax allows complex query patterns but still remains easy-to-use. For example, `valve -pulmo*` searches for documents that contain the term "valve" but do not contain terms beginning with "pulmo". Inexperienced users have a higher chance to understand the syntax because it is more similar to the one of web search engines. Lucene itself does not provide a user interface but there are tools like *Apache Solr*, or *Apache Stanbol* which is a semantic search project based on Apache Solr.

Our requirements are similar in certain aspects to *Gate mímir* [12], which is an indexing and retrieval tool for Gate. It allows to find text passages that match certain annotation or text patterns. For example,

[10] `http://lucene.apache.org/core/`

[11] `http://svn.apache.org/repos/asf/uima/addons/trunk/Lucas/`

[12] `http://gate.ac.uk/mimir/`

`transistor IN {Abstract}`[13] searches for abstracts regarding transistors. In combination with GATE's rule engine Jape[14], similar functionality can be achieved for terminology development. A query tool for document processing and retrieval based on Apache Lucene and GATE is Annic[15] (ANNotations In Context) (Aswani et al., 2005). It provides a viewer for nested annotation structures and features.

3 Edtgar: Terminology Development Environment

The system has been developed as part of a medical information extraction project that populates the data warehouse of a German hospital. Documents in clinical domains differ from the kind of text that is widely used for ontology development, hence common approaches to learn ontologies, for example, with lexical patterns, have relatively low entity acceptance rates on clinical documents (Liu et al., 2011). As a consequence, ontology learning and enrichment methods and information extraction components have to be adapted to the domain. We address this adaptation step integrating new editors, views and application logic into existing tooling for working with UIMA documents. Some features of our system are especially useful for processing clinical text but they also work for similar domains, such as advertisements, product descriptions, or semi-structured product reviews.

In order to assist users during development, we qualitatively analyzed workflows in a project with clinical reports. As a result, we identified the following steps:

1. Linguistic preprocessing: A technical engineer chooses a component for preprocessing steps like tokenization, sentence detection, part-of-speech tagging, chunking, or parsing. In our projects, a rule engineer typically adapts general rules to fit to a special domain. He modifies tokenization rules and lists of abbreviations, and specifies segmentation rules that detect relevant parts of the document, annotate sections, subsections, sentences, and segments.

2. Initial terminology generation: an algorithm automatically derives a terminology based on a corpus of plain text documents.

3. Terminology refinement: a domain expert changes the terminology until it meets subjective or objective quality criteria:

    (a) Automatically extract information according to the current state of the terminology.
    (b) Inspect extraction results.
    (c) Refine terminology: edit/add new concepts, add/update variants of existing concepts.

4. Annotation of gold standard documents. Evaluation of the information extraction component and the coverage of the terminology.

Further analysis showed different aspects of improvement that conform with common expectations:

1. Search: terminology engineers frequently need to pose different kinds of search patterns on document collections. Some of them are related to quality estimation without a gold standard.

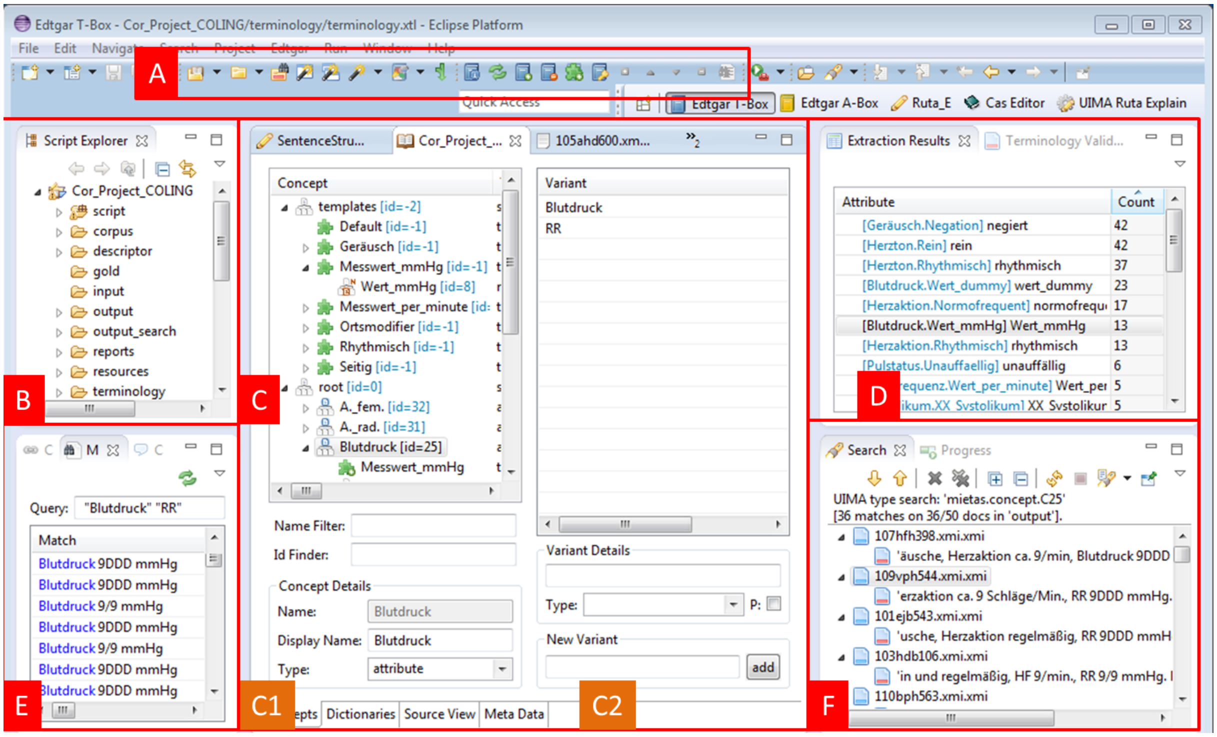
2. Redundancy: modeling concepts independently of each other causes false negatives. Some concepts have highly redundant aspects that should be managed in a central way.

In the following, we first sketch the terminology model and then briefly report on the terminology induction, validation, and information extraction components. In this work, we put special emphasis on analysing search tools that can be used in a UIMA-based workbench. Our approach to pose queries in the workbench is discussed in Section 4.

[13] from: `http://services.gate.ac.uk/mimir/query-session-examples.pdf`

[14] `http://gate.ac.uk/sale/tao/splitch8.html`

[15] `http://gate.ac.uk/sale/tao/splitch9.html`



Variant:
Blutdruck
RR

Name Filter:
Id Finder:

Concept Details
Name: Blutdruck
Display Name: Blutdruck
Type: attribute

Type: P: □

Attribute	Count
[Geräusch.Negation] negiert	42
[Herzton.Rein] rein	42
[Herzton.Rhythmisch] rhythmisch	37
[Blutdruck.Wert_dummy] wert_dummy	23
[Herzaktion.Normofrequent] normofrequ	17
[Blutdruck.Wert_mmHg] Wert_mmHg	13
[Herzaktion.Rhythmisch] rhythmisch	13
[Pulstatus.Unauffaellig] unauffällig	6
requenz.Wert_per_minute] Wert_per	5
ikum.XX_Systolikum] XX_Systolikur	5

Query: "Blutdruck" "RR"

Match
Blutdruck 9DDD mmHg
Blutdruck 9DDD mmHg
Blutdruck 9/9 mmHg
Blutdruck 9DDD mmHg
Blutdruck 9/9 mmHg
Blutdruck 9DDD mmHg
Blutdruck 9DDD mmHg

UIMA type search: 'mietas.concept.C25'
[36 matches on 36/50 docs in 'output'].

```
107hfh398.xmi.xmi
    'äusche, Herzaktion ca. 9/min, Blutdruck 9DDD
109vph544.xmi.xmi
    'erzaktion ca. 9 Schläge/Min., RR 9DDD mmHg.
101ejb543.xmi.xmi
    'usche, Herzaktion regelmäßig, RR 9DDD mmH
103hdb106.xmi.xmi
    'in und regelmäßig, HF 9/min., RR 9/9 mmHg. I
110bph563.xmi.xmi
```

Figure 1: Edtgar T-Box Perspective: (A) Toolbar; (B) Project Structure; (C) Terminology Editor with (C1) Concept Tree, (C2) Variants; (D) Information Extraction Statistics; (E) Search View (Lucene); (F) Search Results (Filesystem: TreeViewer mode)

3.1 Terminology Model

In order to describe the information contained in documents of a domain, we follow an attribute-value model extended with concepts of type *entity* (*object*) that disambiguate different meanings of attributes.

The central aspects of our knowledge representation are given by a tuple

$$O = (T, C, D, V, R_{T\times C}, R_{T\times T}, R_{C\times V}, R_{V\times D})$$

where T, C, D, V contain templates, concepts, dictionaries, and variants, respectively. The relation $R_{T\times C}$ defines which concepts use certain templates, $R_{T\times T}$ models inter-template references. *Concepts* are the main elements of the terminology. There are several types of concepts such as objects, attributes, and values. Each concept is expressed by lexical variants (relation $R_{C\times V}$) which can be grouped by dictionaries (relation $R_{V\times D}$). *Variants* can be either simple strings or regular expressions. Attributes that belong to the same kind of semantic category typically share the same sets of values. Ontologies model this kind of relation between concepts typically by inheritance (subclass-of) or instantiation. In our terminology model, users can use *templates* to group concepts and even templates into semantic classes that share all values that are part of the template. As a consequence, the template mechanism avoids redundancy and errors in the terminology. Templates can also be used just to tag concepts. To allow users to store domain specific information, concepts can have arbitrary properties (key-value-pairs). All information of the model is de-/serializable as a single easy-to-read XML file.

3.2 Terminology Induction

Developing a terminology from scratch is costly, but some domains allow that a considerable amount of attributes and values can be automatically found. Algorithms that induce terminologies and relations can be easily plugged into the workbench through certain APIs. Per default, a simple rule-based approach based on part-of-speech tags is used. It basically finds different kinds of patterns for attributes (nouns) and values (adjectives). The algorithm uses the lemmas of lexical items to group concepts and

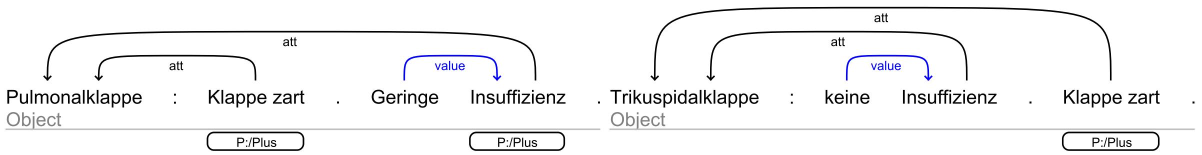


Figure 2: Ambiguous attributes: the attribute insufficiency (German: *Insuffizienz*) is both valid for the entities pulmonary valve (German: *Pulmonalklappe*) and tricuspid valve (German: *Trikuspidalklappe*). Insufficiency itself can either be negated (German: *keine*) or minor (German: *gering*). Pulmonary valve and tricuspid valve are both regular in this example (German: *Klappe zart*).

creates different variants of the concepts respectively. Probabilities that tell users how certain a concept extraction is can be optionally shown if they are provided by the algorithm.

3.3 Terminology Validation

Terminologies can become quite large, for example, in medical domains, which makes it difficult to manage them manually. For instance, it is important to avoid redundancy because keeping redundant concepts or variants in sync gets difficult. We provide different types of validators that check certain aspects of improvement and show errors and warnings. For example, missing dictionary references cause insufficient synonym lists and false negative extractions. We have a validator that creates warnings if a dictionary should be referenced instead of using only some part of this dictionary in a certain concept. There are several other validators, for instance, to detect missing template references. New validators can easily be integrated into the framework.

3.4 Information Extraction

Similar to its terminology induction module, our system has a plug-in mechanism for information extraction algorithms. By default, it contains an information extraction algorithm which allows for context-sensitive disambiguation of terms with multiple meanings. It can, for example, resolve the correct interpretation for ambiguous terms like “Klappe zart” or “Insuffizienz” as shown in Figure 2.

At first, the terminology is transformed into appropriate data structures for inference. The processing pipeline begins with finding lexemes by matching the text against regular expressions and simple strings. The next annotator segments the document into hierarchically ordered parts such as sections, subsections, sentences, segments, and tokens. This component is implemented with Ruta rules which enables rule engineers to adapt this stage to different document types and styles easily. The following stage is implemented as a Java analysis engine. At first, it finds and annotates objects, i.e., entities that are close to the root of the terminology and belong to the object type in the terminology. These concepts are typically expressed by unique variants and should not need any disambiguation. Afterwards, the algorithm tries to assign unresolved attribute entities to objects, or directly annotates special entities. Finally, the algorithm performs *light-weight inference*: first, value extractions are removed when the corresponding attribute has been negated. Second, presence annotations are added if an attribute entity requires a status value and is not negated in the sentence.

3.5 Knowledge-driven Evaluation and Status Reports

Similar to quality assurance or quality assessment in factories, users can specify assertions for text processing tasks where system behavior is expected to conform to these assertions (Wittek et al., 2013). Such expectations allow to compute quality estimates even without annotated documents. To this end, we provide special annotation types for these cases that can be categorized to distinguish different tasks or types of misclassifications. By now, knowledge-driven evaluation is realized by the contributed search commands (see Section 4). They allow to find, count, and group constraint violations which helps to estimate the quality of the text processing component and to understand the errors it makes. For example, the user can expect that all nouns should either be assigned to a concept annotation or listed in a blacklist. Elaboration of this aspect of the tool is planned for future releases.

3.6 Edtgar Workbench Overview

Edtgar's graphical user interface (see Figure 1) provides two perspectives and several views to assist the user. The heart of the tool is the *terminology editor* that allows to create or modify concepts (attributes and values, etc.), manage variants and dictionaries. If the main terminology of a project is opened, users can set the active corpus for the project, or trigger several actions. They can start terminology induction/enrichment, information extraction, or run different types of specialized or general searches on the active corpus, for example, by pressing the corresponding buttons in the toolbar. The tool also provides several other features that we do not discuss here, e.g., support for semi-automatic gold standard annotation, evaluation, documentation, and much more.

3.7 Typesystem Handling

Inspecting the results of processed documents is a visual process. Representing each concept type by a distinct annotation type has a technical advantage because occurrences of a certain type of concept can be highlighted in the UIMA CAS editor with a different color. During terminology induction, however, the terminology and the corresponding annotation types do not exist in the typesystem of the processed document. As a result, the presented framework uses a hybrid concept representation. Engines can use generic annotation types with an integer id feature to create proposal annotations for terminology generation and enrichment. These annotations are subsequently mapped to type based representations when the terminology and its typesystem have been completely defined. As a natural side-effect of terminology development, identifiers of concepts may change, for instance, if concepts are rejected or merged. The terminology editor keeps identifiers stable as long as possible since both representation schemes have problems with invalid IDs. An advantage of the ID feature based approach is that it is able to retain invalid references whereas a type-based approach looses information when documents are loaded leniently.

Besides concept annotations, the system provides framework types for different purposes. For instance, *IgnoredRegion*, *UnhandledSegment*, or *Sealed*. They allow to configure irrelevant text detection for each subdomain, enable users to find new terms, or that have been manually inspected and contain gold standard annotations, respectively.

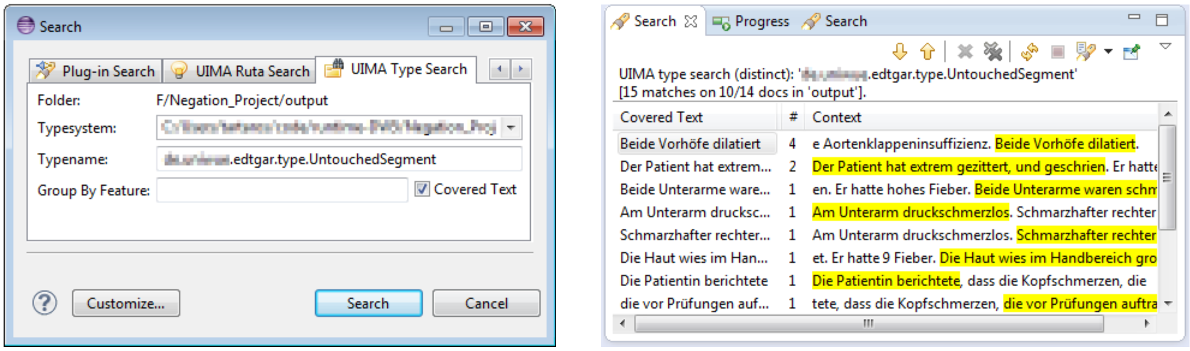
4 Search Tools for the Workbench

An important aspect of terminology development is *search support*. It should assist users with predefined queries, e.g., to find new concepts, synonyms of concepts, or different meanings of concepts. Technical users, however, must be able to perform searches with arbitrary complex patterns. Several tasks can be regarded as corpus queries, for instance, finding measurement units or abbreviations.

In order to support domain experts, we identified three main types of queries that fit to their workflows. First of all, *Unhandled Segments Search* is a kind of false negative predictor (cf. Section 3.5). It lists all segments in the corpus that have not yet been covered either by terminological concepts or exclude patterns. This is in particular useful in clinical domains where nearly every noun phrase contains relevant information. It can, however, easily be adapted to other domains. Second, *querying specific types of extracted information* is necessary, for instance, to inspect extraction results and to judge their quality. It allows to create partial gold standards and it helps to decide if attributes have ambiguous meanings dependent on their context. Third, *constrained search* supports the terminology engineer when editing concepts or creating new entries: users often search for attributes to find their value candidates and need to see documents containing synonyms of the attribute but leaving known value terms out. Finally, if domain experts are not familiar with regular expressions, they benefit from *fast lexical search* to test their patterns.

Query Tools in Edtgar

Most of the queries mentioned above can be implemented as searches for certain annotation types (*UIMA Type Search*). For instance, segments that do not contain any extracted information (*Unhandled Segments Search*) can be annotated with a special type, and concept mentions can be annotated with types based on



Folder: F/Negation_Project/output
Typesystem: [redacted]
Typename: [redacted].edtgar.type.UntouchedSegment
Group By Feature: ☑ Covered Text

UIMA type search (distinct): '[redacted].edtgar.type.UntouchedSegment'
[15 matches on 10/14 docs in 'output'].

Covered Text	#	Context
Beide Vorhöfe dilatiert	4	e Aortenklappeninsuffizienz. Beide Vorhöfe dilatiert.
Der Patient hat extrem...	2	Der Patient hat extrem gezittert, und geschrien. Er hatte
Beide Unterarme ware...	1	en. Er hatte hohes Fieber. Beide Unterarme waren schm
Am Unterarm drucksc...	1	Am Unterarm druckschmerzlos. Schmarzhafter rechter
Schmarzhafter rechter...	1	Am Unterarm druckschmerzlos. Schmarzhafter rechter
Die Haut wies im Han...	1	et. Er hatte 9 Fieber. Die Haut wies im Handbereich gro
Die Patientin berichtete	1	Die Patientin berichtete, dass die Kopfschmerzen, die
die vor Prüfungen auf...	1	tete, dass die Kopfschmerzen, die vor Prüfungen auftra

(a) Search Page for UIMA annotation types: results can be grouped by a certain feature or their covered text.

(b) Search Results Page (TableViewer mode): it shows distinct unhandled segments ordered by frequency

Figure 3: Generic Search Components

concept identifiers, e.g., `mietas.concept.C24` for the concept with the ID 24. The system provides two types of query commands that support annotation type searches.

The first query command uses a separate index over a whole corpus and provides very fast retrieval. It is based on Apache Lucene, thus Lucene's user-friendly query syntax can be used. For instance, querying "insufficiency AND aortic -moderate" retrieves sentences that contain "insufficiency" and "aortic" but not "moderate". The interface to Lucene-based queries can be seen in Figure 1 (E). The index is based on sentences and contains special fields for extracted concept annotations. For instance, "concept:24" shows sentences where the concept with the ID 24 has been extracted. Indexing can be triggered manually.

The second query command iterates over files in the filesystem and uses UIMA's index structures to find annotations of requested types. It integrates into the search framework of Eclipse. As a consequence, users can easily expand or collapse search results, browse search results, or iteratively open respective documents in a shared editor instance. The component implements a tree view (confer Figure 1 (F)) and a table view (confer Figure 3b) to show results. The programmatic search command needs a type name, a type system, and a folder. As a special feature, search results can be grouped based on their covered text or the string value of a feature of the annotations. Grouping allows to show absolute counts for each group. It helps users to find and rank relevant candidate terms and phrases. It can, however, also be used in other use cases, for example, to list all distinct person mentions in a text collection. Results can be exported as an HTML report. Figure 3a shows the generic search page for posing UIMA based queries. The folder can be selected in the script explorer (see Fig. 1 (B)). The dialog provides a combo box to choose the type system and auto-completion for the type name and group-by text widgets.

Technical users can already use the middleware in combination with rule-based queries when they create rule scripts just for this purpose. These scripts define queries and types for results. Subsequently, users apply one of the type-based search commands.

Only one of the task specific query types in the terminology development environment is not implemented as a Lucene search or an annotation type based search command: to support quick testing of regular expressions, the tool accesses the text search command of Eclipse which allows very fast search and rapid feedback.

5 Case Study

The main application of the tool is terminology generation and subsequent information extraction for a wide range of medical reports. We evaluated it in a small subdomain of the physical examination concerning the heart (cor). We gathered 200 anonymized documents from a hospital information system and divided the corpus into a training and a test set (150:50). For terminology generation, we first extracted all nouns from the training set as attributes. Then we merged similar entries and deleted irrelevant candidates. For each attribute we generated and manually adapted relevant value candidates assigning templates wherever possible. For both tasks, we applied multiple searches and used tool support for fast inspection of the relevant segments in the documents. The final terminology was used for information extraction on the test set. We measured the time necessary for terminology adaption and the precision and recall of the information extraction on the test set and additionally estimated the recall with a query as described in Section 4. The gold standard for information extraction in this feasibility study was defined by ourselves. From the training set, we extracted 20 attributes and 44 boolean and 6 numerical values. Manual correction took about 4 hours. Microaveraged precision and recall of the information extraction were 99% and 90% on the test set. The estimated recall on the test set was 84%. Roughly one half of the errors was due to unknown terminology in the test documents. The other half was mainly induced by missing variants of already known concepts. With a larger training set, these kinds of errors can be considerably reduced.

6 Summary

Ontology development and query-driven workflows have not gained as much attention in the UIMA community as, for example, pipeline management, rule development, or evaluation. Especially if ontologies are developed in order to build information extraction systems, it is desirable to have a workbench environment that integrates both tools for ontology development and information extraction. The tool suggested in this paper aims to fill this gap. It supports the creation of light-weight ontologies for information extraction, that is, it helps to find attributes and their values, and to encode simple relations between concepts. It integrates into Eclipse and lowers the bridge between different frameworks and tools. Notably, we assist users in query-driven workflows, which includes a simple way to assess quality without manually annotated documents. We plan to release the tool under an open source license.

Acknowledgments

This work was supported by the Competence Network Heart Failure, funded by the German Federal Ministry of Education and Research (BMBF01 EO1004).

References

N. Aswani, V. Tablan, K. Bontcheva, and H. Cunningham. 2005. Indexing and Querying Linguistic Metadata and Document Content. In *Proceedings of Fifth International Conference on Recent Advances in Natural Language Processing (RANLP2005)*, Borovets, Bulgaria.

Paul Buitelaar, Daniel Olejnik, and Michael Sintek. 2004. A Protégé Plug-In for Ontology Extraction from Text Based on Linguistic Analysis. In ChristophJ. Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *The Semantic Web: Research and Applications*, volume 3053 of *Lecture Notes in Computer Science*, pages 31–44. Springer Berlin Heidelberg.

Philipp Cimiano and Johanna Völker. 2005. Text2Onto: A Framework for Ontology Learning and Data-driven Change Discovery. In *Proceedings of the 10th International Conference on Natural Language Processing and Information Systems*, NLDB'05, pages 227–238, Berlin, Heidelberg. Springer-Verlag.

Philipp Cimiano, Andreas Hotho, and Steffen Staab. 2005. Learning Concept Hierarchies from Text Corpora Using Formal Concept Analysis. *J. Artif. Int. Res.*, 24(1):305–339, August.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. 2011. *Text Processing with GATE (Version 6).*

David Ferrucci and Adam Lally. 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3/4):327–348.

Manuel Fiorelli, Maria Teresa Pazienza, Steve Petruzza, Armando Stellato, and Andrea Turbati. 2010. Computer-aided Ontology Development: an integrated environment. In René Witte, Hamish Cunningham, Jon Patrick, Elena Beisswanger, Ekaterina Buyko, Udo Hahn, Karin Verspoor, and Anni R. Coden, editors, *New Challenges for NLP Frameworks (NLPFrameworks 2010)*, pages 28–35, Valletta, Malta, May 22. ELRA.

Peter Kluegl, Martin Atzmueller, and Frank Puppe. 2009. TextMarker: A Tool for Rule-Based Information Extraction. In Christian Chiarcos, Richard Eckart de Castilho, and Manfred Stede, editors, *Proceedings of the Biennial GSCL Conference 2009, 2nd UIMA@GSCL Workshop*, pages 233–240. Gunter Narr Verlag.

K. Liu, W. W. Chapman, G. Savova, C. G. Chute, N. Sioutos, and R. S. Crowley. 2011. Effectiveness of Lexico-syntactic Pattern Matching for Ontology Enrichment with Clinical Documents. *Methods of Information in Medicine*, 50(5):397–407.

Michael Tanenblatt, Anni Coden, and Igor Sominsky. 2010. The ConceptMapper Approach to Named Entity Recognition. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may. European Language Resources Association (ELRA).

Andreas Wittek, Martin Toepfer, Georg Fette, Peter Kluegl, and Frank Puppe. 2013. Constraint-driven Evaluation in UIMA Ruta. In Peter Kluegl, Richard Eckart de Castilho, and Katrin Tomanek, editors, *UIMA@GSCL*, volume 1038 of *CEUR Workshop Proceedings*, pages 58–65. CEUR-WS.org.