Flexible UIMA Components for Information Retrieval Research

**Christof Müller*, Torsten Zesch*,
Mark-Christoph Müller*, Delphine Bernhard*, Kateryna Ignatova*,
Iryna Gurevych* and Max Mühlhäuser†**

* Ubiquitous Knowledge Processing Lab
† Telecooperation Division
Technische Universität Darmstadt, Germany
{mueller|zesch|chmark|delphine|ignatova|gurevych|max}@tk.informatik.tu-darmstadt.de

Abstract

In this paper, we present a suite of flexible UIMA-based components for information retrieval research which have been successfully used (and re-used) in several projects in different application domains. Implementing the whole system as UIMA components is beneficial for configuration management, component reuse, implementation costs, analysis and visualization.

1. Introduction

Existing information retrieval (IR) tools and frameworks like Apache Lucene[1] focus primarily on application building, where fast indexing and retrieval capabilities for large data collections are the driving factor. In IR *research* however, indexing and retrieval speed are not the (only) important factors. For rapidly performing successful IR experiments, it is crucial to

- support an easy integration, combination and configuration of new IR algorithms,
- manage vast numbers of runs of IR experiments resulting from different system configurations,
- provide evaluation methods for retrieval performance, and
- visualize the data, the retrieval process and the results.

Successful research in the field of IR and the development of new IR models involve constant changes to both the algorithm implementations and the preprocessing components, as well as the handling and visualization of (potentially huge amounts of) textual data for analysis purposes. A recent shift in IR towards semantics and NLP methods, as indicated by emerging search engines like Powerset, Hakia, Lexxe, and CognitionSearch,[2] shows the need for integrating more sophisticated preprocessing capabilities into IR frameworks.

In this paper, we present a suite of flexible UIMA-based components for IR research which have been successfully used (and re-used) in several projects in different application domains. The components are part of the DKPro (Darmstadt Knowledge Processing) repository[3], a collection of UIMA-based components for NLP tasks. The focus of this paper is on a description of the IR components in the DKPro repository. Section 2. briefly describes some requirements for research-oriented IR systems. Section 3. outlines a generic IR workflow and how it is realized by our DKPro components. Section 4. describes some of the projects in which they have been successfully applied.

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

[2] http://www.powerset.com, http://www.hakia.com, http://lexxe.com, http://cognitionsearch.com

[3] http://www.ukp.tu-darmstadt.de/software/repository

2. UIMA for Research-Oriented IR

From the above characterization of IR research, some clear requirements for the implementation of IR systems can be deduced, including the ability to process (potentially huge amounts of) unstructured natural language text, and to quickly configure different setups using varying combinations of (pre-)processing and retrieval components.

The modular nature of our components (as brought about by the UIMA architecture) simplifies within-project configuration management (i.e. different system configurations for different experiment runs), and minimizes the effort for cross-project employment (i.e. re-use) of components. The implementation of IR algorithms as UIMA components also offers the possibility to use the results of sophisticated NLP methods in the retrieval process without having to build custom indexing formats. Moreover it enables a thorough analysis of data and results as the visualization component can create combined views of the preprocessing and retrieval process.

3. IR Components in DKPro

The DKPro software repository is a collection of UIMA components for various NLP tasks. Among components for tasks in areas as diverse as topic segmentation, opinion mining, and community mining, it also contains flexible and efficient IR components.[4] The components cover all steps in what can be regarded as a generic IR workflow. Figure 1 provides an overview.

3.1. Collection Reading

This initial step relates to the basic task of importing the test collections (i.e. the documents and the related topics[5]) into the IR system. In UIMA, it is to be performed by instances of *reader* components. In different application domains, document collections come in vastly different formats, and it is in the reader (and only here) that the peculiarities of the respective formats are dealt with. The DKPro repository contains several readers for various formats. A

[4] Currently based on Lucene. Work for supporting further IR toolkits like e.g. Terrier is ongoing.

[5] The *topic* is a natural language statement of a user's information need which is used to create a *query* in an IR system.

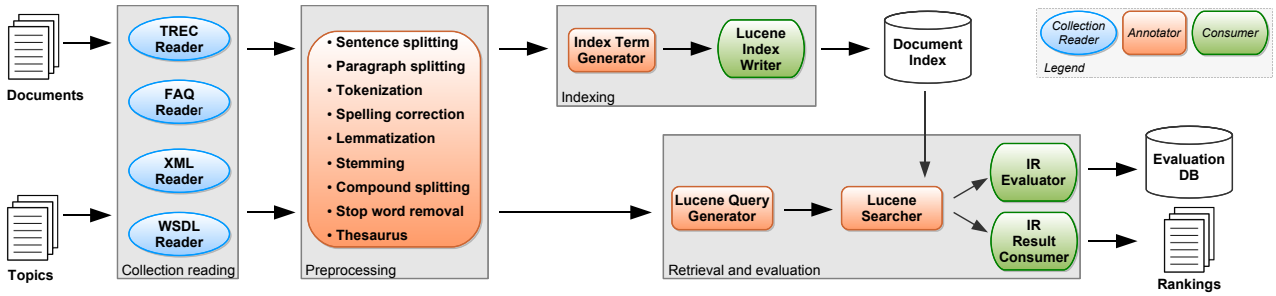

Figure 1: DKPro Components in a Generic IR Workflow

core functionality performed by all readers is the annotation of each processed collection item (i.e. document and topic) with a *DocumentMetaData* annotation. Apart from providing a unique ID for each item, this annotation also contains information like the title of a document or the ID of the collection it belongs to. This information is used in several downstream processing steps, including retrieval and visualization (cf. below). Some of the readers conserve collection-specific formatting information by adding annotations to the document. The `XMLReader` e.g. can be parameterized to create annotations for arbitrary XML elements found in a document. Other readers (like e.g. the `WSDLReader`) use more elaborate analysis to create more specific annotations.

3.2. Preprocessing

IR document collections normally consist of natural language text (but cf. Section 4.3.). Some preprocessing is commonly performed in order to (1) make explicit hidden structure within the texts (e.g. sentence or paragraph splitting or tokenization), (2) normalize their content (e.g. lemmatization, stemming, compound splitting, or spelling correction), or (3) add linguistic meta information (e.g. POS tagging, parsing, or stop word identification).
In UIMA, this is modelled as a task for *annotator* components, which add the new information and the normalized content in the form of annotations. More substantial modifications (like e.g. spelling correction in error-prone user-generated discourse, cf. Section 4.2.1.) can be implemented by having the annotator component actually *modify* the underlying content.[6] This method is used by `SpellingCorrector`. For numerous preprocessing tasks, powerful stand-alone tools are already available in the NLP research community. Where possible, the components in DKPro utilize these. Our `POSTagger` and `Lemmatizer` e.g. are wrappers for the TreeTagger (Schmid, 1994). In a broader sense, preprocessing can also be understood to comprise less generic and more application-specific tasks. For IR, one of these tasks is query expansion, in which related terms are added to the query text. The DKPro repository contains a component which adds related terms (e.g. based on various types of semantic relatedness (Gurevych, 2005)) in the form of annotations. Keeping the original query text and the expansion terms apart by adding the latter in the form of annotations is particularly useful because it allows explicit control over the use of the query expansion feature by downstream components, e.g. for assigning a different weight to expansion terms in the query generation and retrieval process.

3.3. Indexing

The generation of a document and query index is a prerequisite for efficient retrieval. The scope and nature of the index can vary for different collections and different applications. In some settings, all document and query tokens (presumably excluding stop words) have to be indexed, while in other settings only certain parts might be relevant. In DKPro, the `IndexTermGenerator` annotator is responsible for identifying terms to be indexed. Provided that the respective preprocessing has been performed earlier, it can create index terms of entire tokens, lemmata, stems, and/or other arbitrary annotation elements. If the `POSTagger` annotator was applied to the documents and queries to be indexed, index term generation can also be constrained by POS information. The resulting index terms are then written by a *consumer* component to an index file in the format required by the IR engine to be used. Up to now, the DKPro repository contains a `LuceneIndexWriter` and some project specific components, which are described in Section 4.

3.4. Retrieval, Evaluation, and Visualization

In the retrieval step, the previously generated document and query indices and a set of parameter settings (e.g. threshold values to be used) are employed to create actual IR *runs*. A run consists of the application of all queries to a document collection and yields a quantitative evaluation of the overall effectiveness of the applied (pre-)processing pipeline, parameter settings, and retrieval engine for a particular document collection. The retrieval step is broken down into query generation, search, evaluation, and (optionally) visualization. For each of the first three steps, there is a dedicated component in DKPro. The first two (`LuceneQueryGenerator` and `LuceneSearcher`) are particular to the retrieval engine to be used. The third one (`IREvaluator`) is a general-purpose IR evaluation component which computes common IR evaluation measures by wrapping the `trec_eval`[7] tool, but which also offers other evaluation measures like Spearman's rank cor-

[6] Technically, this is implemented by having the annotator create a new *view* containing the altered content.

[7] http://trec.nist.gov/trec_eval

relation coefficient. The `IREvaluator` can optionally store the evaluation results in a relational database. The stored results include not only the overall retrieval results, but also detailed information about individual topics and documents.

In contrast to visualization of IR results in an end-user oriented setting[8], IR *research* is best supported by allowing researchers to trace individual topics and documents through the entire retrieval run, e.g. for error or general performance analysis. For the DKPro IR components, this is supported by a component which allows result visualization and browsing. As browsing is inherently interactive, it is not naturally implemented as a (pipeline-oriented) UIMA component. Therefore, result browsing is implemented as a servlet-based web application which reads evaluation information from the database (created by the `IREvaluator`) and displays it in a web browser. The analysis process which is necessary for understanding and improving the IR model requires data browsing on different information levels:

- *run level:* configuration parameters and overall results;
- *query level:* evaluation results of each query (for selected runs);
- *document level:* relevance scores and relevance assessments of each document (for a certain query and selected runs);
- *process level:* visualization of the retrieval process of a document (for a certain query and selected runs).

The component uses the original documents and topics, the output of the retrieval process and the relevance assessments. In order to provide detailed information on the process level, the component offers the possibility to rerun the processing pipeline for a selected document and query, adding a special consumer to the pipeline which creates an HTML document with preprocessing and retrieval information. In this step, topic and document are passed simultaneously through the pipeline (in the same *CAS* object, but in two separated *views*) and the retrieval components can add additional information that helps to understand the details of the retrieval process.

Especially for research purposes, the tight coupling of preprocessing and retrieval can be beneficial when developing new IR algorithms. Instead of investing time in (re-)adjusting or implementing new indexing formats, the retrieval components can (temporarily) work directly on the annotations created by the preprocessing components.

3.5. Configuration Management

As mentioned above, IR research aims at finding new and improved algorithms and optimized settings for IR parameters. Also, different configurations for preprocessing steps yield multiple indices. In practice, therefore, the processing workflow described above has to be executed very often. The DKPro IR components are complemented with a number of helper components for batch execution of experimental runs. The helper classes provide functionality for programmatically configuring and executing collection processing engines. The configurations can be stored in a relational database which enables the visualization and comparison of IR results in the visualization component.

4. DKPro IR Components in Use

In this section, we give a detailed account of how some of the components in the DKPro repository are employed in several projects in different application domains. Where available, experimental results are also reported.

4.1. Electronic Career Guidance

The task of electronic career guidance is to support school leavers in their search for a profession or a vocational training to take up. In (Gurevych et al., 2007), we describe work in which electronic career guidance is modelled as an IR task. Vocational trainings are represented by documents which were automatically extracted from BERUFEnet, a database created by the German Federal Labour Office. Topics are short essays collected from students in which they describe in their own words what they would like their future job to be like. One special challenge of this task is the large *vocabulary gap* between the language of the (expert-authored) documents from the database and the language of the students. The term *vocabulary gap* relates to the fact that people with different backgrounds or different levels of expertise use (sometimes strikingly) different vocabularies when describing similar things. String-based IR approaches (as represented e.g. by Lucene) are not able to adequately handle this phenomenon. The best results reported in (Gurevych et al., 2007) were therefore produced by a *semantic* information retrieval component, which scores the similarity of documents and queries on the basis of their semantic relatedness. The components come as the annotators `RelatednessScorer` and `SemanticSearcher` and the consumer `SemanticIndexWriter`, and fit seamlessly into the pipeline of the other DKPro components.

4.2. Question Answering

Question Answering (QA) systems aim at giving precise answers to natural language questions. The architecture of traditional QA systems is therefore more complex than IR systems, since they have to include a component which extracts answers from documents. The answer extraction problem can be avoided by leveraging the wealth of information available on the Web in the form of Frequently Asked Questions (FAQ) pages and question-answer services such as Yahoo!Answers[9] or WikiAnswers[10]. When answers are retrieved from question-answer repositories, the QA task can be redefined as an IR task where topics are natural language questions and documents are the question-answer pairs. There are actually two ways to address this task: by identifying paraphrases of the input question in a question-answer repository (Section 4.2.1.), or by retrieving the most similar question-answer pair from an FAQ (Section 4.2.2.).

[8] http://people.lis.uiuc.edu/~twidale/irinterfaces/2classics.html, http://people.ischool.berkeley.edu/~hearst/tb-overview.html

[9] http://answers.yahoo.com

[10] http://wiki.answers.com

4.2.1. Question Paraphrase Identification

The objective of this task is to retrieve those questions in the question-answer repository which are most similar to the input question. A first difficulty lies in the fact that most online question-answer services record real user questions, which may be ill-formulated or may contain spelling errors. Prior to indexing, therefore, we apply the `SpellingCorrector` annotator. In order to perform the matching of an input question to the most similar question in a question-answer pair, we have implemented several text similarity measures based on the work by Tomuro & Lytinen (2004) and Zhao et al. (2007), among others. These measures include matching coefficient, word overlap coefficient, edit distance and term vector cosine similarity. Two UIMA annotators are in charge of computing the similarity values and ranking the results for each input question. These annotators replace the `LuceneQueryGenerator` and `LuceneSearcher` components in the generic retrieval step described above. Since the similarity measure to be used in a given experiment is a component's parameter, the available measures can be easily tested and new text similarity measures can be conveniently added.

4.2.2. FAQ Mining

Based on the work by Jijkoun & de Rijke (2005), we aim at answering users' questions by retrieving relevant question-answer pairs found in FAQ pages. Within this task, a document is considered as a collection of several fields: question and answer of a question-answer pair, title of the corresponding FAQ page, and the full text of the FAQ page. In order to keep this document-specific information, annotations are added to the document in the collection reading step by means of a parameterized `XMLReader` annotator. Further, the preprocessing stage allows to normalize the content by performing lemmatization and stemming, which are required for later building both stemmed and lemmatized indices. Additional information, such as the document's language and contained stopwords, is also added at this point. The `IndexTermGenerator` allows to index different fields, e.g. a non-stemmed question keeping stopwords, a stemmed answer without stopwords, etc. Easy combination of annotation components and flexibility during indexing make it possible to easily evaluate different system configurations as described by Jijkoun & de Rijke (2005). Our current baseline system reimplements several of their models with comparable results. E.g., the performance of the baseline model for the retrieval of the so called 'adequate' and 'material' answers is 45% in the top 10 results.

4.3. Web Service Retrieval

Web service retrieval is the task of retrieving from a repository of web services those services that provide a particular functionality. When cast as an IR task, topics are descriptions of required functionalities, while the services to be retrieved are represented as semi-structured documents. These documents have been created by crawling known web service repositories and processing the collected WSDL files. Within each WSDL file, the `WSDLReader` identifies and analyzes operation names and operation signatures (i.e. names and types of operation parameters) and creates a textual representation to be processed using the standard IR workflow.

5. Conclusion

In this paper, we presented a suite of flexible UIMA-based components for research-oriented information retrieval which have been successfully used (and re-used) in several projects in different application domains. The usage of UIMA as framework not only shows benefits for the preprocessing components, but also for the actual retrieval components. Apart from well-known features of UIMA like configuration management, component reuse, and replicating processing pipelines, the tight coupling inside UIMA of the preprocessing and the actual retrieval process offers possibilities for fast prototyping of new IR algorithms by directly using UIMA annotations instead of developing custom indexing formats. It also extends analysis and visualization capabilities by offering combined views of preprocessing and retrieval on different levels of granularity.

The described IR and preprocessing components are part of the DKPro repository and (with some exceptions) will be made available to interested researchers.

Acknowledgements Parts of this work were carried out in two projects funded by the German Research Foundation (DFG): "Semantic Information Retrieval from Texts in the Example Domain Electronic Career Guidance" (grant GU 798/1-2), and "Mining Lexical-Semantic Knowledge from Dynamic and Linguistic Sources and Integration into Question Answering for Discourse-Based Knowledge Acquisition in eLearning" (grant GU 798/3-1).

References

Gurevych, Iryna (2005). Using the structure of a conceptual network in computing semantic relatedness. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP'2005)*. Jeju Island, Republic of Korea.

Gurevych, Iryna, Christof Müller & Torsten Zesch (2007). What to be? - Electronic career guidance based on semantic relatedness. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pp. 1032–1039. Prague, Czech Republic.

Jijkoun, Valentin & Maarten de Rijke (2005). Retrieving answers from frequently asked questions pages on the web. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pp. 76–83. New York, NY, USA: ACM.

Schmid, Helmut (1994). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing (NeMLaP). Manchester, U.K., 14–16 September 1994*.

Tomuro, Noriko & Steven Lytinen (2004). Retrieval Models and Q&A Learning with FAQ Files. In Mark T. Maybury (Ed.), *New Directions in Question Answering*, pp. 183–194. AAAI Press.

Zhao, Shiqi, Ming Zhou & Ting Liu (2007). Learning Question Paraphrases for QA from Encarta Logs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 1795–1801. Hyderabad, India.