



**GSCL 2013**

International Conference of the German Society for  
Computational Linguistics and Language Technology

September 25–27, 2013  
Darmstadt, Germany



# **Unstructured Information Management Architecture (UIMA)**

**3rd UIMA@GSCL Workshop**

**September 23, 2013**

Copyright © 2013 for the individual papers by the papers' authors. Copying permitted only for private and academic purposes. This volume is published and copyrighted by its editors.

# Preface

For many decades, NLP has suffered from low software engineering standards causing a limited degree of re-usability of code and interoperability of different modules within larger NLP systems. While this did not really hamper success in limited task areas (such as implementing a parser), it caused serious problems for the emerging field of language technology where the focus is on building complex integrated software systems, e.g., for information extraction or machine translation. This lack of integration has led to duplicated software development, work-arounds for programs written in different (versions of) programming languages, and ad-hoc tweaking of interfaces between modules developed at different sites.

In recent years, the Unstructured Information Management Architecture (UIMA) framework has been proposed as a middleware platform which offers integration by design through common type systems and standardized communication methods for components analysing streams of unstructured information, such as natural language. The UIMA framework offers a solid processing infrastructure that allows developers to concentrate on the implementation of the actual analytics components. An increasing number of members of the NLP community thus have adopted UIMA as a platform facilitating the creation of reusable NLP components that can be assembled to address different NLP tasks depending on their order, combination and configuration.

This workshop aims at bringing together members of the NLP community – users, developers or providers of either UIMA components or UIMA-related tools in order to explore and discuss the opportunities and challenges in using UIMA as a platform for modern, well-engineered NLP.

This volume now contains the proceedings of the 3rd UIMA workshop to be held under the auspices of the German Language Technology and Computational Linguistics Society (Gesellschaft für Sprachverarbeitung und Computerlinguistik - GSCL) in Darmstadt, September 23, 2013. From 11 submissions, the programme committee selected 7 full papers and 2 short papers. The organizers of the workshop wish to thank all people involved in this meeting - submitters of papers, reviewers, GSCL staff and representatives - for their great support, rapid and reliable responses, and willingness to act on very sharp time lines. We appreciate their enthusiasm and cooperation.

September 2013

*Peter Kluegl, Richard Eckart de Castilho, Katrin Tomanek (Eds.)*

## Program Committee

Sophia Ananiadou	University of Manchester
Steven Bethard	KU Leuven
Ekaterina Buyko	Nuance Deutschland
Philipp Cimiano	University of Bielefeld
Anni R. Coden	IBM Thomas J. Watson Research Center
Kevin Cohen	University of Colorado
Richard Eckart de Castilho	Technische Universität Darmstadt
Frank Enders	Averbis
Nicolai Erbs	Technische Universität Darmstadt
Stefan Geissler	Temis Deutschland
Thilo Götz	IBM Deutschland
Udo Hahn	FSU Jena
Nicolas Hernandez	University of Nantes
Michael Herweg	IBM Deutschland
Nancy Ide	Vassar College
Peter Kluegl	University of Würzburg
Eric Nyberg	Carnegie Mellon University
Kai Simon	Averbis
Michael Tanenblatt	IBM Thomas J. Watson Research Center
Martin Toepfer	University of Würzburg
Katrin Tomanek	Averbis
Karin Verspoor	National ICT Australia
Graham Wilcock	University of Helsinki
Torsten Zesch	University of Duisburg-Essen

## Additional Reviewers

Roman Klinger	University of Bielefeld
---------------	-------------------------

## Table of Contents

Keynote: Apache clinical Text Analysis and Knowledge Extraction System (cTAKES) . . . .	1
<i>Pei Chen and Guergana Savova</i>	
A Model-driven approach to NLP programming with UIMA . . . . .	2
<i>Alessandro Di Bari, Alessandro Faraotti, Carmela Gambardella and Guido Vetere</i>	
Storing UIMA CASes in a relational database . . . . .	10
<i>Georg Fette, Martin Toepfer and Frank Puppe</i>	
CSE Framework: A UIMA-based Distributed System for Configuration Space Exploration	14
<i>Elmer Garduno, Zi Yang, Avner Maiberg, Collin McCormack, Yan Fang and Eric Nyberg</i>	
Aid to spatial navigation within a UIMA annotation index . . . . .	18
<i>Nicolas Hernandez</i>	
Using UIMA to Structure An Open Platform for Textual Entailment . . . . .	26
<i>Tae-Gil Noh and Sebastian Padó</i>	
Bluima: a UIMA-based NLP Toolkit for Neuroscience . . . . .	34
<i>Renaud Richardet, Jean-Cedric Chappelier and Martin Telefont</i>	
Sentiment Analysis and Visualization using UIMA and Solr . . . . .	42
<i>Carlos Rodríguez-Penagos, David García Narbona, Guillem Massó Sanabre, Jens Grivolla and Joan Codina</i>	
Extracting hierarchical data points and tables from scanned contracts . . . . .	50
<i>Jan Stadermann, Stephan Symons and Ingo Thon</i>	
Constraint-driven Evaluation in UIMA Ruta . . . . .	58
<i>Andreas Wittek, Martin Toepfer, Georg Fette, Peter Kluegl and Frank Puppe</i>	

# Keynote: Apache clinical Text Analysis and Knowledge Extraction System (cTAKES)

## Abstract

The presentation will focus on methods and software development behind the cTAKES platform. An overview of the modules will set the stage, followed by more in-depth discussion of some of the methods and evaluations of select modules. The second part of the presentation will shift to software development topics such as optimization and distributed computing including UIMA integration, UIMA-AS, as well as our plans for UIMA-DUCC integration. A live demo of cTAKES will conclude the talk.

## About the speakers

**Pei Chen** is a Vice President of Apache Software Foundation, leading the top-level cTAKES project<sup>1</sup>. He is also a lead application development specialist at the Informatics Program at Boston Children's Hospital/Harvard Medical School. Mr. Chen's interests lie in building practical applications using machine learning techniques. He has a passion for the end-user experience and has a background Computer Science/Economics. Mr. Chen is a firm believer in the open source community contributing to cTAKES as well as other Apache Software Foundation projects.

**Guergana Savova**, Ph.D. is member of the faculty at Harvard Medical School and Childrens Hospital Boston. Her research interest is in natural language processing (NLP), especially as applied to the text generated by physicians (the clinical narrative) focusing on higher level semantic and discourse processing which includes topics such as named entity recognition, event recognition, relation detection, and classification including co-reference and temporal relations. The methods are mostly machine learning spanning supervised, lightly supervised, and completely unsupervised. Her interest is also in the application of the NLP methodologies to biomedical use cases. Dr. Savova has been leading the development and is the principal architect of cTAKES. She holds a Master's of Science in Computer Science and a PhD in Linguistics with a minor in Cognitive Science from University of Minnesota.

---

<sup>1</sup><http://ctakes.apache.org/>

# A Model-driven approach to NLP programming with UIMA

Alessandro Di Bari, Alessandro Faraotti,  
Carmela Gambardella, and Guido Vetere

IBM Center for Advanced Studies of Trento  
Piazza Mancini, 1 Povo di Trento

**Abstract.** In Natural Language Processing, more complex business use cases and shorter delivery times drive a growing need of smoother, more flexible and faster implementations. This trend also requires integrating and orchestrating different functionalities delivered by services belonging to different technological platforms. All these needs imply raising the level of abstraction for NLP components development. In this paper we present a Model Driven Architecture approach suitable to develop an open and interoperable UIMA-based NLP stack. By decoupling UIMA NLP models from other solution specific platforms and services, we obtain major architectural improvements.

## 1 Introduction

As Natural Language Processing (NLP) approaches complex tasks such as Question Answering or Dialog Management, the capability for NLP tools to seamlessly interoperate with other software services, such as knowledge bases or rules engines, becomes crucial. Such level of integration may require linguistic models to be shared among a variety of different platforms, each of which comes with its own information representation language. Platforms like UIMA<sup>1</sup> or GATE<sup>2</sup> consist of middleware and tools for designing and pipelining NLP specific tasks, including support for modeling data structures for text annotation, such as lexical, morphological and syntactic features, which may be embedded in inter-process communication protocols. However, while perfectly suited for annotation purposes, NLP specific schema languages, such as the UIMA Type System, fall short on fulfilling solution-level modeling needs. Model-Driven software Architectures (MDA), on the other hand, are specifically aimed at tackling the complexity of modern software infrastructures, with emphasis on the integration and the orchestration of different technological platforms. The MDA approach is based on providing formal descriptions (models) of requirements, interactions, data structures, protocols, and many other aspects of the desired system, which are automatically turned into technical resources, such as schemes and software modules, by activating transformation rules.

---

<sup>1</sup> <http://uima.apache.org/>

<sup>2</sup> <http://gate.ac.uk/>

Based on this consideration, we adopted an MDA approach to develop a “Watson ready”<sup>3</sup>, UIMA-based NLP stack for Italian, as part of the activity of the newborn IBM Language & Knowledge Center for Advanced Studies of Trento<sup>4</sup>. We wanted our stack to be as open and interoperable as possible, to help users leveraging the availability of NLP resources and tools in the Open Source / Open Data space. In addition, our stack aims at being independent from language specific issues and domains, to facilitate its reuse across projects and within our (multinational) Company. The basic idea was to design a highly modularized general model including all the required structures, and to obtain technical platform-specific resources from a suitable set of model-to-model transformations. Also, we embraced the idea of abstracting semantic information away from the UIMA Type System, as in [5] and in [7], and evaluated the benefit of representing such kind of information by specific means. In sum, we looked at UIMA as a well-suited platform for linguistic analysis, which allows the integration of analytic components into managed workflow pipelines, but regarded at the UIMA Type System as a schema specification for that platform, rather than as a general modeling language for any NLP-based solution.

Here we present an overview of the basic ideas behind our approach, introduce our project, and discuss future directions. At the present stage of development, we can share our vision on MDA positioning and motivation with respect to NLP development (section 3), and we can report our first implementation experiences (section 4). Finally, we outline some related topic and introduce future works.

## 2 Motivating Scenario

Natural Language based solutions may require the NLP stack to cooperate with other components in a complex system. Such cooperation typically involves data exchanges with reference to a shared information model. The picture 1 shows the integration of an NLP stack with a Knowledge Base (e.g. an Ontology-based Data Access System) and a Rule Engine.

An UIMA-based NLP pipeline produces an annotated text (step 1 in the picture 1) contained in an UIMA CAS (Common Annotation Structure). A wrapper of the UIMA Type System defines all the operations needed for a consumer (the Rule Engine in this case) in order to access the CAS and invoke the appropriate operations within the cooperating subsystem when needed (see 4.2). When developing and maintaining the solution, an Engineer builds a rule set (see step 3) in order to process linguistic structures and interact with a Knowledge Base (step 4), which, in turn, uses the annotated text to store assertions as the result of an Information Extraction process (step 5). In a separate flow, the Knowledge Base can be queried by a User through a Question Answering System based on a suitable query language (step 6). The integration of all components involved is guaranteed by a common abstract model (Platform Independent Model) that contains the overall conceptualization of the system. The transition from one

---

<sup>3</sup> [www.ibm.com/watson/](http://www.ibm.com/watson/)

<sup>4</sup> [www.ibm.com/ibm/cas/](http://www.ibm.com/ibm/cas/)



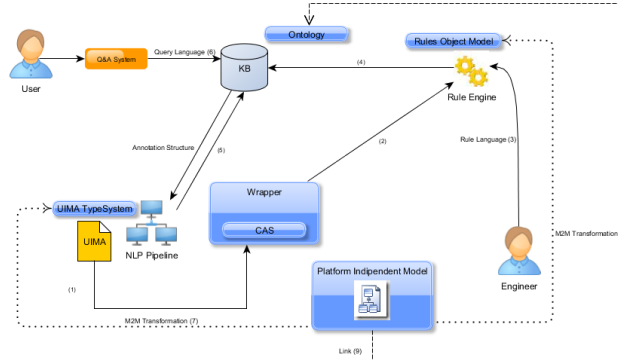


Fig. 1. Architectural sketch

platform specific data structure to another is handled by a set of Model-to-Model transformations (steps 7 and 8). The figure also shows the link to legacy (possibly huge) conceptual models, such as the KB ontology (step 9).

### 3 Model Driven Architecture for NLP

Model Driven Architecture (MDA) [6] is a development approach, strictly based on formal specifications of information structures and behaviors, and their semantics. MDA is managed by Object Management Group (OMG)<sup>5</sup> based on several modeling standard such as: Unified Modeling Language (UML)<sup>6</sup>, Meta-Object Facility (MOF), XML Metadata Interchange (XMI) and others. MDA supports Model Driven Development/Engineering (MDD, MDE).

The key idea behind MDA is to provide a higher level of abstraction so that software can be fully designed independently from the underlying technological platform. More formally, MDA defines three macro “modeling” layers:

- **Computation Independent Model (CIM)**
- **Platform Independent Model (PIM)**
- **Platform Specific Model (PSM)**

The first one can be related to a Business Process Model and does not necessary imply the existence of a system that automates it. The PIM is a model that is independent from any technical platform; the third (PSM) layer is the actual implementation of the model with respect to a given technology and it is automatically derived from the PIM. Notice that the PIM allows a comprehensive representation of the structure and behavior of the system being developed.

<sup>5</sup> <http://omg.org/>

<sup>6</sup> <http://www.uml.org/>

The modeling language is typically UML or EMF<sup>7</sup>, but it could actually be any other Domain Specific Language (DSL).

Developing powerful NLP tasks, such as Question Answering systems, requires combining a great variety of analytic components, which is what UIMA has been designed for. We consider UIMA the standard solution for document workflow analysis. Within this framework, MDD tools can be effectively used to better manage the UIMA Type System. In particular, we decided to look at it as a PSM dedicated to text annotation. The motivation for leveraging MDD (in the NLP field) can be summarized as follows:

- **Formalization:** MDA languages are well studied in logics and reasoning mechanisms can be developed upon. [1]
- **Expressiveness:** MOF meta-modeling allow great and well-founded expressiveness [4], including modeling behaviors.
- **Support:** The availability of tools, including diagramming and code generation, improves software life-cycle and team collaboration.

In particular, with respect to our architecture, we modeled UIMA annotations by defining *classes* rather than just (data) types, so that a consumer is able to invoke operations designed for those objects. Access to UIMA annotation is then achieved by means of automatically generated wrappers. Another motivation for a model driven approach was the need to represent complex linguistic data, and exploit existing tooling and resources for generating training data for a statistical parser.

In sum, we tried to exploit the maturity and flexibility of MDD tools while keeping up the power of UIMA as a framework for component integration, pipeline execution, and workflow management in general. As the PIM language, we chose EMF because it is already integrated with UIMA and provides powerful and mature model driven features. Once also the code is generated (by UIMA JCASgen), the type system correspond to an implementation of a (business) domain model, limited to the structural aspects (as opposed to behavioral aspects).

At PIM level, we also have to represent those properties that, once transformed against a target model, give specific characteristics on that model. For instance, in order to generate the UIMA Type System (PSM) starting from the PIM, we have to represent on the source model whether a class (that is a root in a hierarchy on the PIM model) will be generated as an UIMA annotation or not (UIMA TOP). Here we have taken two possible scenarios into account:

- Having an UML PIM, this specification is easily accomplished by using an UML *profile*<sup>8</sup>. Profiles define stereotypes that can be further structured with custom properties. This way, we have a generic "Unstructured Information" profile that at least, encompasses an **Annotation** stereotype; thus a class that is thought to become an annotation will be simply "marked" with this stereotype.

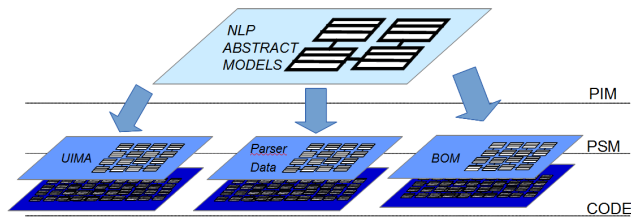
<sup>7</sup> <http://www.eclipse.org/modeling/emf/>

<sup>8</sup> <http://www.omg.org/spec/#M&M>

- Having an EMF PIM (such as our current implementation), we can represent the same thing as an EMF annotation. Therefore, (we apologize for the words conflict) we will have a class annotated as `Annotation`.

In any case, a class stereotyped as `Annotation` on the PIM will take the role of a generic annotation for document analysis, independently from the underlying framework.

The main benefit of our approach is the ability to represent NLP objects independently from any particular implementation: we are using different (generated) PSMs (that are better explained in section 4) all deriving from the starting (PIM) model, as shown in the picture 2.



**Fig. 2.** Mdd for NLP: different abstraction layers

These benefits have certainly a price, that is essentially represented by the cost of developing the necessary transformations. However, following basic assumptions of the MDD approach, we estimate that those cost are well paying back, especially when heterogeneous components have to be integrated, development is managed iteratively, and models are subject to high volatility.

## 4 Model Driven Implementation Aspects

In order to better clear up how we are leveraging the Model Driven approach, we list here the artifacts (PSMs and code) we are generating through appropriate transformations that we have developed. Starting from our “application” model:

- UIMA type system (we modified the existing transformation from EMF in order to avoid any further modification on the UIMA type system)
- EMF wrapper of UIMA type system
- this wrapper also acts as the input for creating the model for the Rule engine as explained below

Starting from (our) models of common standard data for parser training such as CONLL, PENN and others we generated all necessary (OpenNLP-specific) data for training the parser on:

- Tokenization
- Named Entities
- Part of Speech tagging
- Chunking
- Parsing

To represent the model (PIM), we use the Eclipse Modeling Framework<sup>9</sup> (EMF), which represents a *de facto* Java-based standard for meta-modeling. Informally, we may say EMF represents a subset of UML (the structural part) with very precise semantics for code generation. In the future, we could move this representation to a profiled UML, as mentioned above (see section 3). Furthermore, EMF offers very powerful generation features. Summarizing, in the current implementation we use EMF in two ways:

1. A language to represent the model
2. A PIM model to generate different target PSM

#### 4.1 NLP Parser

The NLP Parser component is implemented using Apache OpenNLP<sup>10</sup> and UIMA<sup>11</sup>; it is based on a UIMA Type System built from the Syntax and the Abstract models using the UIMA transformation utility. The training corpora for the parser has to be provided in a specific format required by OpenNLP. Since the data that we had available for training were in standard formats such as PENN<sup>12</sup>, CONLL<sup>13</sup> and others, some transformations were required. Ecore models have been created for the purpose of representing source formats. Furthermore, some simple JET<sup>14</sup> transformations has been developed in order to generate our corpora (in specific OpenNLP formats)

Compared to other solutions, this makes our infrastructure extremely flexible: should the parser be replaced or the data formats changed, the only operation we will have to make is to modify the JET template accordingly.

#### 4.2 Type System EMF Wrapper

As anticipated in 2, in the higher layers of our architecture, we have a Rule Engine that acts as a reasoner on annotation objects coming from the UIMA pipeline. We wanted this layer to be able to call operations implemented on those objects (as explained in section 3) and those objects always implementing the exact interfaces of the (Ecore) PIM model. Given these requirements, we developed a transformation that generates a wrapper of the UIMA type system and that

<sup>9</sup> <http://www.eclipse.org/modeling/emf/>

<sup>10</sup> <http://opennlp.apache.org/>

<sup>11</sup> <http://uima.apache.org/>

<sup>12</sup> <http://www.cis.upenn.edu/~treebank/>

<sup>13</sup> <http://ilk.uvt.nl/conll/#dataformat>

<sup>14</sup> <http://www.eclipse.org/modeling/m2t/?project=jet>

fully reflects the starting PIM model, including operations. Once implemented, the code will be kept up also against future re-generations, thanks to merging capabilities of this transformation. Thus, as shown in figure 1, the Rule Engine “consumes” instances of this wrapper, and still can access the underlying UIMA annotation. We considered the possibility of directly adding these operations on classes generated by UIMA (via JCAS generation utility) but this would not be consistent with our model-driven approach since those operations would not be part of a general, system-wide model.

### 4.3 Rule Engine

As far as the Rule Engine is concerned, we chose IBM Operational Decision Manager (ODM)<sup>15</sup>. ODM rules have to be written against a specific model, called Business Object Model (BOM), that allows a user-friendly business rule editing; ODM provides tools to set up a natural language vocabulary: users can use it to write business rules in a pseudo-natural language. Once defined, the rules are executed on a BOM-related Java implementation named Execution Object Model (XOM). We obtained the BOM by reverse engineering the XOM, and the XOM directly from Java classes (implementing the type system wrapper) generated from our PIM (EMF) model. Therefore, the BOM model can be seen as just another manifestation of our PIM model.

### 4.4 Knowledge Base

Our architecture is backed by a Knowledge Base Management System which stores and reasons on information extracted from many sources. Leveraging on the Knowledge Model included in the PIM, we were able to integrate an external pre-existing system, named ONDA (Ontology Based Data Access) [3]. ONDA supports Ontology Based Data Access (OBDA) on OWL2-QL (<sup>16</sup>), by ensuring sound and complete conjunctive query answering with the same efficiency a scalability of a traditional database [2]. Because the ONDA underlying Knowledge Model was already designed with EMF, we simply adopted it in order to be included in the PIM. This way, reasoning and query answering services have been included in the PIM model as operations available to all other components (i.e. the Rule Engine).

## 5 Conclusion and future works

We have outlined here an innovative approach to NLP development, based on the idea of setting UIMA as the target platform in a Model-Driven development process. A major benefit of this approach consists in giving NLP models a greater value, especially in terms of generality, usability, and interoperability.

<sup>15</sup> <http://www-03.ibm.com/software/products/us/en/odm/>

<sup>16</sup> <http://www.w3.org/TR/owl2-profiles/>

While developing this idea, we understood that a suitable Model-Driven machinery for NLP should be supported by specific design patterns for concrete models. In particular, the model we have developed has been abstracted both from morphosyntactic specificity and from semantic aspects. The former (including part-of-speech classes, genders, numbers, verbal tenses, etc) may significantly vary among different languages; the latter (including concepts like persons, events, places, etc) are related to specific application domains. By decoupling these layers, we achieved a lightweight “generic” UIMA type system[7], we designed a powerful generic model for morphosyntactic features, and we managed ontological information with proper expressive means. Refining and extending this model is part of our future plans.

We implemented a first prototype of a Knowledge Base query system based on the Eclipse Modeling Framework (EMF). For the future, we are considering the possibility of representing the model in UML, in order to have a greater representational power (such as modeling sequence diagrams).

The work presented here is still at an early stage. More work is needed to complete the linguistic model, for instance in the area of argument structures, such as verbal frames. From an implementation standpoint, our priority is to consolidate, improve and extend the set of Model-to-Model transformations, and to further exploit MDD tools.

## References

1. A. Cali, D. Calvanese, G. De Giacomo, and M. Lenzerini. A formal framework for reasoning on uml class diagrams. In *Proceedings of the 13th International Symposium on Foundations of Intelligent Systems*, ISMIS '02, pages 503–513, London, UK, UK, 2002. Springer-Verlag.
2. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Dl-lite: Tractable description logics for ontologies. In *AAAI*, volume 5, pages 602–607, 2005.
3. P. Cangialosi, C. Consoli, A. Faraotti, and G. Vetere. Accessing data through ontologies with onda. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '10, pages 13–26, Riverton, NJ, USA, 2010. IBM Corp.
4. Liliana Favre. A formal foundation for metamodeling. In F. Kordon and Y. Ker-marrec, editors, *Reliable Software Technologies Ada-Europe 2009*, volume 5570 of *Lecture Notes in Computer Science*, pages 177–191. Springer Berlin Heidelberg, 2009.
5. D. Ferrucci, J W. Murdock, and C. Welty. Overview of component services for knowledge integration in uima (aka suki). Technical report, IBM Research Report RC24074, 2006.
6. J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, Object Management Group (OMG), 2003.
7. K. Verspoor, W. Baumgartner Jr, C. Roeder, and L. Hunter. Abstracting the types away from a UIMA type system. *From Form to Meaning: Processing Texts Automatically*. Tübingen:Narr, pages 249–256, 2009.

# Storing UIMA CASes in a relational database

Georg Fette<sup>1,2</sup>, Martin Toepfer<sup>1</sup>, and Frank Puppe<sup>1</sup>

<sup>1</sup> Department of Computer Science VI, University of Wuerzburg,  
Am Hubland, Wuerzburg, Germany

<sup>2</sup> Comprehensive Heart Failure Center, University Hospital Wuerzburg,  
Straubmuehlweg 2a, Wuerzburg, Germany

**Abstract.** In the UIMA text annotation framework the most common way to store annotated documents (CAS) is by serializing the document to XML and storing this XML in a file in the file system. We present a framework to store CASes as well as their type systems in a relational database. This does not only provide a way to improve document management but also the possibility to access and manipulate selective parts of the annotated documents using the database's index structures. The approach has been implemented for MSSQL and MySQL databases.

**Keywords:** UIMA, data management, relational databases, SQL

## 1 Introduction

UIMA [2] has become a well known and often used framework for processing text data. The main component of the UIMA infrastructure is the CAS (Common Analysis Structure), a data structure which combines the actual data (the text of a document), annotations on this data and the type system the annotations are based on. In many UIMA projects CASes are stored as serialized XML-files in file folders with the corresponding type system file in a separate location. In this storage mode the resource management to load which CAS with which type system lies in the responsibility of the programmer who wants to perform an operation on specific documents. However, manual management of files in folders on local machines or network folders can quickly become confusing and messy especially when projects get bigger. We present a framework to store CASes as well as their corresponding type systems in a relational database. This storage mode provides the possibility to access the data in a centralized, organized way. Furthermore the approach provides all benefits that come along with relational databases including search indices on the data, selective storage, retrieval and deletion as well as the possibility to perform complex queries on the stored data in the well known SQL language.

The structure of the paper is as follows: Section 2 describes the related work, Section 3 describes the technical details of the database storage mechanism, Section 4 illustrates query possibilities using the database, Section 5 demonstrates performance experiences with the framework and Section 6 concludes with a summary of the presented work.

## 2 Related Work

The only approach known to the best knowledge of the authors where CASes are stored in a database is the Julielab DB Mapper [4] which serialized CASes to a PostgreSQL database. However, the mechanism does not store the CASes' type systems nor does it support features like referencing of annotations by features or derivation of annotation types. Other approaches use indices to improve query performance but do not allow to reconstruct the annotated documents from the index (Lucene based: LUCAS [4], Fangorn [3]; relational database based: XPath [1], ANNIS [7]; proprietary index based: TGrep/TGrep2 [6], SystemT [5]). The indices still need the documents to be stored in the file system. Furthermore some of the mentioned indices only allow specialized search capabilities (e.g. emphasis on parse trees) which are provided by the respective search index and cannot search directly on the UIMA data structures. In contrast to these approaches our system allows searches on arbitrary type systems by formulating queries closely related to the involved annotation and feature types.

## 3 Database storage

The storage mechanism is based on a relational database for which the table model is illustrated in Figure 1. The schema can be subdivided in a document related part (left), an annotation instance part (middle) and a type system related part (right). Documents are stored as belonging to a named collection and can be manipulated (retrieved, deleted, etc.) as a group, e.g. deleting all annotations of a specific type. Annotated documents can be handled individually by loading/saving a single CAS or by processing a whole collection by creating a collection reader/writer. In either way any communication (loading/saving) can (but need not) be parametrized so that only desired annotation types are loaded/saved, thus speeding up processing time, reducing memory consumption and facilitating debugging processes. A type system, instead of being stored in an XML file and containing a fixed type system, can be retrieved from the database in different task specific ways. One way is by requesting the type system which is needed to load all the annotated documents belonging to a certain collection. Other possibilities are by providing a set of desired type names or by providing a regular expression determining all desired type names. The storage mechanism is able to store the inheritance structures of UIMA type systems as well as referencing of annotations by features of other annotations. For further information on the technical aspects we refer to the documentation of the framework<sup>3</sup>.

## 4 Querying

A benefit from storing data in an SQL database is the database index and the well established SQL query standard. The database can be queried for counts of occurrences of specific annotation types, counts of covered texts of annotations or even complex annotation structures in the documents. We want to exemplify this with a query on documents which have been annotated with a dependency parser using the type system shown in Figure 2.

<sup>3</sup> <http://code.google.com/p/uima-sql/>



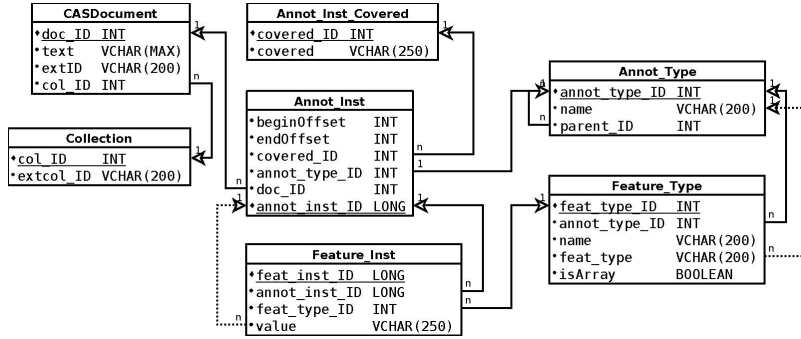


Fig. 1. schema of the relational database storing CASes and type systems.

```

<typeDescription>
<name>Token</name>
<supertypeName>
uima.tcas.Annotation
</supertypeName>
<features>
<featureDescription>
<name>Governor</name>
<rangeTypeName>Token</rangeTypeName>
</featureDescription></features>
</typeDescription>

```

Fig. 2. type system for parses

```

SELECT govText.covered FROM
annot_inst govToken, annot_inst_covered govText,
annot_inst baseToken, annot_inst_covered baseText,
feat_inst, feat_type WHERE
baseText.covered = 'walk' AND
baseToken.covered_ID = baseText.covered_ID AND
baseToken.annot_inst_ID = feat_inst.annot_inst_ID AND
feat_inst.feat_type_ID = feat_type.feat_type_ID AND
feat_type.name = 'Governor' AND
feat_inst.value = govToken.annot_inst_ID AND
govText.covered_ID = govToken.covered_ID

```

Fig. 3. SQL query for governor tokens

To query for all words governing the word *walk*, we have to look for tokens with the desired covered text, find the tokens governing those tokens and return their covered text. The SQL command for this task is shown in Figure 3. An abstraction layer to cover the complexity could be put on top (like a graph querying language), but even in the presented way with standard SQL the capabilities of the database engine can serve as a useful tool to improve corpus analysis.

## 5 Performance

To run a performance test on the storage engine we created a corpus of 1000 documents, each consisting of 1000 words. The words were taken from a dictionary of 1000 randomly created words, each of 8 characters length. From each document we created a CAS and added annotations so that each word was covered, with the annotations covering 1 to 5 successive words. Each annotation was given two features, one String feature with a value randomly taken from the word dictionary and a Long feature containing a random number. All documents were stored and then loaded again. This was done with the database engine as well as with a local file folder on the same hard drive the database files were located on. In a second experiment the same documents were loaded again and we added an annotation of another type with a Long feature containing a random number to each document. After adding the additional annotation the documents were stored again. In a third experiment we wanted to query for the frequencies of annotations covering each of the words from the word dictionary.

For file system storage this was done by accumulating the annotation counts during an iteration over all serialized CASEs, for database storage this was done by performing a single SQL query for each of the words from the dictionary.

In Table 1 we can observe that the time needed for database storage is quite long but reading is as fast as from the file system. Storing to the database during the second experiment was faster than in the first one, because this time only the additional annotations had to be incrementally stored. Storage to the file system again performed about five times faster than to the database but the benefit of being able to incrementally store only the additional annotations can be clearly observed. Physical storage space consumption is larger for database storage but that shouldn't pose a major problem as hard disc space is not an overly expensive resource nowadays. Query performance in the database is about 20 times faster than using file system storage illustrating the benefit of the database approach.

**Table 1.** Performance measures comparing database and file system storage

	exp1		exp2		exp3
	saving (sec.)	loading (sec.)	saving (sec.)	storage size (MB)	query (sec.)
DB	36.0	1.1	7.2	42.3	0.16
FileSystem	2.6	1.1	2.7	6.5	7.0

## 6 Conclusion

We have presented a framework to store/retrieve CASEs and perform analysis queries on them using a relational database. We examined the save, load and query speed compared to regular file based storage and presented examples how to use the database index structures to analyze annotations in the corpus. We hope to be able to improve the storage speed of the database engine so that the choice between file system storage and database storage will not be influenced by the still quite large difference in speed performance.

This work was supported by grants from the Bundesministerium fuer Bildung und Forschung (BMBF01 EO1004).

## References

1. Bird, S., Lee, H.: Designing and evaluating an xpath dialect for linguistic queries. In: 22nd International Conference on Data Engineering (2006)
2. Ferrucci, D., Lally, A.D.A.M.: Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering* 10(3-4), 327–348 (2004)
3. Ghodke, S., Bird, S.: Fangorn: A system for querying very large treebanks. In: COLING (Demos). pp. 175–182 (2012)
4. Hahn, U., Buyko, E., Landefeld, R., Mühlhausen, M., Poprat, M., Tomanek, K., Wermter, J.: An overview of JCoRe, the JULIE lab UIMA component repository. In: LREC'08 Workshop 'Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP' (2008)
5. Krishnamurthy, R., Li, Y., Raghavan, S., Reiss, F., Vaithyanathan, S., Zhu, H.: Systemt: a system for declarative information extraction. *SIGMOD Rec.* (2009)
6. Rohde, D.L.T.: Tgrep2 user manual (2001)
7. Zeldes, A., Lüdeling, A., Ritz, J., Chiarcos, C.: Annis: a search tool for multi-layer annotated corpora. In: *Proceedings of Corpus Linguistics 2009* (2009)

# CSE Framework: A UIMA-based Distributed System for Configuration Space Exploration

Elmer Garduno<sup>1</sup>, Zi Yang<sup>2</sup>, Avner Maiberg<sup>2</sup>, Collin McCormack<sup>3</sup>, Yan Fang<sup>4</sup>, and Eric Nyberg<sup>2</sup>

<sup>1</sup> Sinnia, [elmerg@sinnia.com](mailto:elmerg@sinnia.com)

<sup>2</sup> Carnegie Mellon University, [{ziy, amaiberg, eh}@cs.cmu.edu](mailto:{ziy, amaiberg, eh}@cs.cmu.edu)

<sup>3</sup> Boeing Company, [collin.w.mccormack@boeing.com](mailto:collin.w.mccormack@boeing.com)

<sup>4</sup> Oracle Corporation, [yan.fang@oracle.com](mailto:yan.fang@oracle.com)

**Abstract.** To efficiently build data analysis and knowledge discovery pipelines, researchers and developers tend to leverage available services and existing components by plugging them into different phases of the pipelines, and then spend hours to days seeking the right components and configurations that optimize the system performance. In this paper, we introduce the CSE framework, a distributed system for a parallel experimentation test bed based on UIMA and uimaFIT, which is general and flexible to configure and powerful enough to sift through thousands option combinations to determine which represent the best system configuration.

## 1 Introduction

To efficiently build data analysis and knowledge discovery “pipelines”, researchers and developers tend to leverage available services and existing components by plugging them into different phases of the pipelines [1], and then spend hours, seeking for the components and configurations that optimize the system performance. The Unstructured Information Management Architecture (UIMA) [3] provides a general framework for defining common types in the information system (*type system*), designing pipeline phases (*CPE descriptor*), and further configuring the components (*AE descriptor*) without changing the component logic. However there is no easy way to configure and execute a large set of combinations without repeated executions, while evaluating the performance of each component and configuration.

To fully leverage existing components, it must be possible to automatically explore the space of system configurations and determine the optimal combination of tools and parameter settings for a new task. We refer to this problem as *configuration space exploration*, which can be formally defined as a constraint optimization problem. A particular information processing task is defined by a configuration space, which consists of  $m_t$  components that define each of the  $n$  phases with corresponding configurations. Given a limited total resource capacity  $\mathcal{C}$  and input set  $\mathcal{S}$ , **configuration space exploration** (CSE) aims to find the trace (a combination of configured components) within the space that achieves the highest expected performance without exceeding  $\mathcal{C}$  total cost. Details on the mathematical definition and proposed greedy solutions can be found in [6].

In this paper, we introduce the CSE framework implementation, a distributed system for parallel experimentation test bed based on UIMA and uimaFIT [4]. In addition, we highlight the results from two case studies where we applied the CSE framework to the task of building biomedical question answering systems.

```

# main.yaml
collection-reader:
  inherit: input-reader
  dataset: TRECGEN06
  file: /trecgen06.txt

pipeline:
- inherit: ecd.phase
  name: tokenization
  options:
    - inherit: stanford-nlp
    - inherit: linepipe
- inherit: ecd.phase
  name: s/acronym-exp
  options:
    - inherit: mesh
    - inherit: entrez-gene
    - inherit: umls

- inherit: ecd.phase
  name: retrieval
  options:
    - inherit: indri
    - inherit: lucene
- inherit: ecd.phase
  name: reranking
  options:
    - inherit: proximity
    - inherit: base.noop
- inherit: prec-evaluator
  post-process:
    - inherit: map-evaluator
    - inherit: report-gen
    builders:
      - inherit: map-report

# indri.yaml
class: IndriWrapper
cross-opts:
  smoothing: [0.1, 0.3, 0.5, 0.7]
  weighting: [0.1, 0.3, 0.5, 0.7]

```

Fig. 1. Example YAML-based pipeline descriptors

## 2 Framework Architecture

We highlight some features of the implementation in this section. Source code, examples, documentation, and other resources are publicly available on GitHub<sup>5</sup>. To benefit developers who are already familiar with UIMA framework, we have developed a CSE tutorial in alignment with the examples in the official UIMA tutorial.

**Declarative descriptors.** To leverage the CSE framework, users need to specify how the components should be organized in the pipeline, which values need to be specified for each component configuration, which is the input set, and what measurement metrics should be applied. Analogous to a typical UIMA CPE descriptor, components, configurations, and collection readers in the CSE framework are declared in *extended configuration descriptors* which are based on the YAML format. An example of the main pipeline descriptor and a component descriptor are shown in Figure 1.

**Architecture.** Each pipeline can contain an arbitrary number of AnalysisEngines declared by using the *class* keyword or by *inheriting* configuration options from other components by name. Combinations of components are configured using an *options* block and parameter combinations within a component are configured on a *cross-opts* block. To take full advantage of the CSE framework capabilities, users inherit from a *cse.phase*, a CAS multiplier that provides, option multiplexing, intermediate resource persistence and resource management for long running components. The architecture also supports grouping options into sub-pipelines as a convenient way of reducing the configuration space for combinations whose performance is already known.

**Evaluation.** Unlike a traditional scientific workflow management system, CSE emphasizes the evaluation of component performance, based on user-specified evaluation metrics and gold-standard outputs at each phase. In addition the framework keeps track of the performance of all the executed traces, this allows inter-component evaluation and automatic tracking of performance improvements through time.

**Automatic data persistence.** To support further error analysis and reproduction of experimental results, intermediate data (CASes) and evaluation results are kept in a repository accessible from any trace at any point during the experiment. To prevent duplicate execution of traces the system keeps track of all the execution traces and recovers those CASes whose predecessors have

<sup>5</sup> <http://oaqa.github.io/>

**Table 1.** Case study result

	# Comp	# Conf	# Trace	# Exec	DocMAP			PsgMAP		
					Max	Median	Min	Max	Median	Min
Participants	~1,000	~1,000	92	~1,000	.5439	.3083	.0198	.1486	.0345	.0007
CSE	13	32	2700	190,680	.5648	.4770	.1087	.1773	.1603	.0311

already been executed. Also the overall results from experiments are kept in a historical database to allow researchers to keep track of the performance improvements along time.

**Configurable selection and pruning.** If gold-standard data is provided for a certain phase, then components up to that phase can be evaluated. Given the measured cost of executing the components provided, components can be ranked, selected or pruned for evaluation and optimization of subsequent phases. The component ranking strategy can be configured by the user; several heuristic strategies are implemented in the open source software.

**Distributed architecture.** We have extended the CSE framework implementation to execute the task set in parallel on a distributed system using JMS. The components and configurations are deployed into the cluster beforehand. The execution, fault tolerance and bookkeeping are managed by a master server. In addition we leverage the UIMA-AS capabilities to execute specific configurations in parallel as separate services directly from the pipeline.

### 3 Building biomedical QA Systems via CSE

As a case study, we apply the CSE framework to the problem of building effective biomedical question answering (BioQA) on two different tasks.

In one case, we employ the topic set and benchmarks, including gold-standard answers and evaluation metrics, from the question answering task of the TREC Genomics Track 2006, as well as commonly-used tools, resources, and algorithms cited by participants. The implemented components, benchmarks, task-specific evaluation methods are included in domain-specific layer named BioQA, which was plugged into the BaseQA framework.

The configuration space was explored with the CSE framework, automatically yielding an optimal configuration of the given components which outperformed published results for the same task. We compare the settings and results for the experiment with the official TREC 2006 Genomics test results for the participating systems in Table 1. We can see that the best system derived automatically by the proposed CSE framework can outperform the best participating system in terms of both DocMAP and PsgMAP, with fewer, more basic components. This experiments ran on a 40 node cluster during 24 hours allowing the execution on 200K components over 2,700 execution traces. More detailed analysis can be found in [6].

We also used the CSE framework to automatically configure a different type of biomedical question answering system for the QA4MRE (Question Answering for Machine Reading Evaluation) task at CLEF. The CSE framework identified a better combination, which achieved 59.6% performance gain over the original pipeline. Details can be found in the working note paper [5].

### 4 Related Work

Previous work has been done on this area, in particular DKPro Lab [2] a flexible lightweight framework for parameter sweep experiments and the U-Compare [7] framework, an evaluation

platform for running tools on text targets and compare components, that generates statistics and instance-based visualizations of outputs.

One of the main advantages of the CSE framework is that it allows the exploration of very large configuration spaces by distributing the experiments over a cluster of workers and collecting the statistics on a centralized way. Another advantage on the CSE framework is that configurations can have arbitrary nesting levels as long as they form a DAG by using sub-pipelines. Also results can be compared end-to-end at a global level to understand overall performance trends on time. One area where CSE could take advantage of the aforementioned frameworks is on having a graphical UI for pipeline configuration, better visualization tools for combinatorial and instance-based comparison and a more expressive language for workflow definition.

## 5 Conclusion & Future Work

In this paper, we present a UIMA-based distributed system to solve a common problem in rapid domain adaptation, referred to as Configuration Space Exploration. It features declarative descriptors, evaluations, automatic data persistence, global resource caching, configurable configuration selection and pruning, and distributed architecture. As a case study, we applied the CSE framework to build a biomedical question answering system, which incorporated the benchmark from TREC Genomics QA task, and the results showed the effectiveness of the CSE framework system.

We are planning to adapt the system to a wide variety of interesting information processing problems to facilitate rapid domain adaption and system building and evaluation of the community. For educational purpose, we are also interested in adopt the CSE framework as an experiment platform to teach students the principled ways to design, implement and evaluate an information system.

**Acknowledgement.** We thanks Leonid Boystov, Di Wang, Jack Montgomery, Alkesh Patel, Rui Liu, Ana Cristina Mendes, Kartik Mandaville, Tom Vu, Naoki Orii, and Eric Riebling for the contribution to the design and development of the system and valued suggestions to the paper.

## References

1. D. Ferrucci et al. Towards the Open Advancement of Question Answering Systems. Technical report, IBM Research, Armonk, New York, 2009.
2. R. E. de Castilho and I. Gurevych. A lightweight framework for reproducible parameter sweeping in information retrieval. In *Proceedings of the DESIRE'11 workshop*, New York, NY, USA, Oct. 2011.
3. D. Ferrucci and A. Lally. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10(3-4), Sept. 2004.
4. P. Ogren and S. Bethard. Building test suites for UIMA components. In *Proceedings of the (SETQA-NLP 2009) workshop*, Boulder, Colorado, June 2009. Association for Computational Linguistics.
5. A. Patel, Z. Yang, E. Nyberg, and T. Mitamura. Building an optimal QA system automatically using configuration space exploration for QA4MRE'13 tasks. In *Proceedings of CLEF 2013*, 2013.
6. Z. Yang, E. Garduno, Y. Fang, A. Maiberg, C. McCormack, and E. Nyberg. Building optimal information systems automatically: Configuration space exploration for biomedical information systems. In *Proceedings of the CIKM'13*, 2013.
7. K. Yoshinobu, W. Baumgartner, L. McCrohon, S. Ananiadou, K. Cohen, L. Hunter, and J. Tsujii. U-Compare: share and compare text mining tools with UIMA. *Bioinformatics*, 25(15):1997–1998, 2009. in press.

# Aid to spatial navigation within a UIMA annotation index

Nicolas Hernandez

Université de Nantes

**Abstract.** In order to support the interoperability within UIMA workflows, we address the problem of accessing one annotation from another when the type system does not specify an explicit link between the two kinds of objects but when a semantic relation between them can be inferred from a spatial relation which connects them. We discuss the limitations of the framework and briefly present the interface we have developed to support such navigation.

**Keywords:** Apache UIMA, Type System interoperability, Annotation Index, Spatial navigation

## 1 Introduction

One of the main ideas in using document analysis frameworks such as *Apache Unstructured Information Management Architecture*<sup>1</sup> (UIMA) [3] is to move away from handling directly the raw *subject of analysis*. The idea is to enrich the raw data with descriptions which can be used as basis for the processing of subsequent components. In the UIMA framework, the descriptions are *typed feature structures*. The component developer defines a *type system* which informs about the features of a type (set of (attribute, typed value) pairs) as well as how the types are arranged together (through inheritance and aggregation relations). *Annotations* are feature structures attached to specific regions of documents.

In this paper, we address the problem of accessing one annotation from another when the type system does not specify an explicit link between the two kinds of objects but when a semantic relation between them can be inferred from a *spatial relation* which connects them. The situation is a case of interoperability issue which can be encountered when developing a component (e.g. a term extractor) that uses analysis results produced by two components developed by different developers (e.g. part-of-speech and lemma information being both hold by distinct annotations at the same spans).

In practice, most of the existing type systems define annotation types which inherit from the built-in `uima.tcas.Annotation` type [4, 5, 7]. This type contains begin and end features which are used to attach the description to a specific region of the text being analysed. Thanks to these features, it is possible to cross-reference the annotations which extend this type.

---

<sup>1</sup> <http://uima.apache.org>

In this paper, we argue that the Apache UIMA Application Programming Interface (API) is not enough intuitive for a Natural Language Processing (NLP) developer. We argue that it has some restrictions which prevent from a complete and free navigation among the annotations. We also argue that the API is forcing the way of developing algorithms. In section 2, we define the kind of spatial navigation we would like to perform within an annotation index. In section 3, we describe the Apache UIMA solutions to index and explore the annotations within the indexes. In section 4, we discuss the API and show its limitation to access annotations by spatial relations. Finally, in section 5, we briefly present the structures and the interface we have developed to support a spatial navigation.

## 2 The spatial navigation problem

By spatial relations we mean that we assume that the annotations in a text can be located in a two-dimensional space: One axis to represent the position in the text linearity and an orthogonal axis to represent the covering degrees between the annotations. Indeed annotations can cover, be covered by, precede, or follow (contiguously or not) other annotations. The spatiality may inform about the semantic relations. Two annotations at the same span may mean that they are different aspects of the same object. They can *have complementary features* or one of them can *be the property of* the other. One annotation covering some others may mean that the former *is made of* the others, and in the opposite, that the others *are part of* the former. The semantic interpretation of the spatial relations may depend on the considered linguistic paradigm.

To give examples of situations we are dealing with, let's assume the following type system: **Document**, **Source** (information about the document such as its URI), **Sentence**, **Chunk**, **Word** (having a feature whose value informs about the lemma), **POS** (having a feature whose value informs about the part-of-speech) and **NamedEntity**. Let's also assume that all these types do not hold explicit references to each other and that there is no inheritance relation between them. In that context, examples of access we would like to carry out are to get :The words of a given sentence (can be interpreted as a *made of* relation); The sentence of a given word (*is part of* relation); The words which are a verb (*is a property of* relation); The named entities followed by a word which is a verb and has the lemma *visit* (*followed by* relation, ...). Indeed, we would like to be able to navigate within an annotation index from an annotation to its covering/covered annotations or to the spatially following/preceding annotation of a given type having such or such properties. We defined this problem as a navigation problem within an annotation index.

## 3 Accessing the annotations in the UIMA framework

The problem of accessing the annotations depends on the means offered by the framework<sup>2</sup> to build annotation indexes and to navigate within them.

---

<sup>2</sup> See the Reference Guide <http://uima.apache.org/d/uimaj-2.4.0/references.html> and the Javadoc <http://uima.apache.org/d/uimaj-2.4.0/apidocs>.



### 3.1 Defining feature structures indexes

Adding a feature structure to a subject of analysis corresponds to the act of indexing the feature structure. By default, an unnamed built-in bag index exists which holds all feature structures which are indexed. The framework defines also a built-in annotation index, called `AnnotationIndex`, which automatically indexes all feature structures of type (and subtypes of) `uima.tcas.Annotation`. As reported in the documentation, "the index sorts annotations in the order in which they appear in the document. Annotations are sorted first by increasing begin position. Ties are then broken by decreasing end position (so that longer annotations come first). Annotations that match in both their begin and end features are sorted using a *type priority*". If no type priority is defined in the component descriptor<sup>3</sup>, the order of the annotations sharing the same span in the text is undefined in the index. The UIMA API provides `getAnnotationIndex` methods to get all the annotations of that index (subtypes of `uima.tcas.Annotation`) or the annotations of a given subtype. The UIMA framework allows also to define indexes and possibly to sort the feature structures within them.

### 3.2 Parsing the annotation index

The UIMA API offers several methods to parse the `AnnotationIndex`. Given an annotation index, the *iterator* method returns an object of the same name which allows to move to the first (respectively the last) annotation of the index, the next (respectively the previous) annotation (depending on its position in the index) or to a given annotation in the index. It is also possible to get an *unambiguous* iterator to navigate among contiguous annotations in the text. In practice, this iterator consists of getting successively the first annotation in the index whose begin value is higher than the end of the current one. We will call this mechanism the *first-contiguous-in-the-index* principle.

The *subiterator* method returns an iterator whose annotations fall within the span of another annotation. It is possible to specify whether the returned annotations should be *strictly* covered (i.e. both begin and end offsets covered) or if it concerns only its begin offset. Subiterator can also be unambiguous. Annotations at the same span may be not returned depending on the order in the index as well as the type priority definition.

The *constrained iterator* allows to iterate over feature structures which satisfy given constraints. The constraints are objects that can test the type of a feature structure, or the type and the value of its features.

The *tree* method returns an *AnnotationTree* structure which contains nodes representing the results of doing recursively a strict, unambiguous subiterator over the span of a given annotation. The API offers methods to navigate within the tree from the root node. From any other nodes, it is possible to get the children nodes, the next or the previous sibling node, and the parent node.

---

<sup>3</sup> In a UIMA workflow, a component is interfaced by a text descriptor that indicates how to use the component.

## 4 Limitations of the UIMA framework

Table 1a shows the `AnnotationIndex` containing the analysis results of the data string ” *Verne visited the seaport of Nantes.\n*”. Annotations were initially added to the index in that order: First the `Document`, then the `Source`, the `Sentence`, the `Words`, the `POS`, the `Chunks` and the `NamedEntities`.

Offsets	Annotations	Covered text	LocatedAnnotations
(0,37)	<code>Document</code>	<i>Verne visited the seaport of Nantes.\n</i>	<code>Document</code>
(0,36)	<code>Sentence1</code>	<i>Verne visited the seaport of Nantes.</i>	<code>Sentence</code>
(0,5)	<code>Word1</code>	<i>Verne</i>	<code>Word1 NamedEntity1 Chunk1 POS1</code>
(0,5)	<code>NamedEntity1</code>	<i>Verne</i>	
(0,5)	<code>POS1</code>	<i>Verne</i>	
(0,5)	<code>Chunk1</code>	<i>Verne</i>	
(0,0)	<code>Source</code>		<code>Source</code>
(6,13)	<code>Word2</code>	<i>visited</i>	<code>Word2 Chunk2 POS2</code>
(6,13)	<code>POS2</code>	<i>visited</i>	
(6,13)	<code>Chunk2</code>	<i>visited</i>	
(14,35)	<code>Chunk3</code>	<i>the seaport of Nantes</i>	<code>Chunk3</code>
(14,25)	<code>Chunk4</code>	<i>the seaport</i>	<code>Chunk4</code>
(14,17)	<code>Word3</code>	<i>the</i>	<code>Word3 POS3</code>
(14,17)	<code>POS3</code>	<i>the</i>	
(18,25)	<code>Word4</code>	<i>seaport</i>	<code>Word4 POS4</code>
(18,25)	<code>POS4</code>	<i>seaport</i>	
(26,35)	<code>Chunk5</code>	<i>of Nantes</i>	<code>Chunk5</code>
(26,28)	<code>Word5</code>	<i>of</i>	<code>Word5 POS5</code>
(26,28)	<code>POS5</code>	<i>of</i>	
(29,35)	<code>Word6</code>	<i>Nantes</i>	<code>Word6 NamedEntity2 POS6</code>
(29,35)	<code>NamedEntity2</code>	<i>Nantes</i>	
(29,35)	<code>POS6</code>	<i>Nantes</i>	
(35,36)	<code>Word7</code>	<i>.</i>	<code>Word7 POS7</code>
(35,36)	<code>POS7</code>	<i>.</i>	

Table 1: An `AnnotationIndex` (a) and its corresponding `LocatedAnnotationIndex` (b). Both tables are aligned for comparison. `Annotations` and `LocatedAnnotations` are sorted in increasing order from the top of the tables. Annotations are identified by their type and an index number.

### 4.1 Index limitations

The definition of an index is usually done in the component descriptor. The defined index can only contain one specific type (and subtypes) of feature structures. So, to get an index made of two distinct types, the trick would be to declare them as subtypes of the same common type in the type system, and get the index of this super type. This can lead to make a less consistent type system from a linguistic point of view, but this is still coherent with the UIMA approach of doing whatever you need in your component.

The framework allows also so to sort the feature structures of a defined index. There are some restrictions. The sorting key, which should be a feature of the indexed type, can only be a string or a numerical value. Only the natural way of sorting such elements is available. There is no way to declare its own comparator to set the order between two elements. To sort on a different kind of key, the developer has to come down to the available systems. In addition, the

type system may need to be modified to add a feature to play the role of the sorting key, which can also make the type system less consistent.

## 4.2 Navigation limitations within an annotation index

**Iterator** With an ambiguous iterator, the result of a move to the previous/next annotation in the index may not correspond to the annotation which precedes/follows spatially in the text. It can also be a covering or a covered one. In Table 1a, the preceding of **Word3** is the covering **Chunk4**. Unambiguous iterators force the methods to return only spatially contiguous annotations. In practice, the method does not always return the expected result. When called on the full annotation index, it starts from the first annotation in the index. In Table 1a, it only returns the **Document** annotation and no more next annotation. When calling a unambiguous iterator on a typed annotation index, the effect of the *first-contiguous-in-the-index* principle will be remarkable if some annotations occur at the same span. In that situation, the developer has no access to all the annotations which effectively follow/precede spatially the current annotation. In Table 1a, an unambiguous iteration over the **Chunk** type returns **Chunk1**, **Chunk2** and **Chunk3**. **Chunk4** and **Chunk5** are not reachable. To iterate unambiguously over annotations of distinct types (e.g. Named Entities and POS to get the Named Entities followed by a verb), the developer has to create a super-type over them and call the iterator method on this super-type. The super-type may not have linguistic consistency and the iterator will still suffer from the limitation we have previously mentioned. Another drawback of the unambiguous iterator can be noticed when iterating an index in reverse order. If two overlapping annotations precede the current one, the one returned will be the one whose begin offset is the smallest and not the one with the highest end value, lower than the begin value of the current one. The iterator follows the *first-contiguous-in-the-index* principle in the normal order. Finally, the API does not allow to iterate over the index and in the text spatiality in the same time. It is not possible to switch from an ambiguous iterator to an unambiguous one (and vice-versa).

**Subiterator** is the kind of method to get the covered annotations of another one, like the words of a given sentence. Its major drawback is that, without a type priority definition, there is no assurance that annotations occurring at the same text span will fit an expected conceptual order. In Table 1a, an ambiguous **subiterator** over each chunk annotation for getting the words returns the **Source** annotation for **Chunk1**, nothing for **Chunk2**, and the expected words (and more to filter) for the all remaining Chunks. Concerning the unambiguous subiterator, the *first-contiguous-in-the-index* principle causes to hide some annotations. In Table 1a, when applying an unambiguous **subiterator** over each chunk, then **Chunk1** and **Chunk2** return the same bad result as previously. **Chunk3** only returns **Chunk4** and **Chunk5** annotations while **Chunk4** and **Chunk5** return the right word annotations. To subiterate unambiguously over a set of specific types, a super-type, which encompasses both the covered and the covering types, has to be defined in the type system. But the problem of the unambiguous iteration remains.

**Constraints objects** aim at testing one given feature structure at a time. The framework does not allow to define dynamic constraints. This means that the values to test cannot be instantiated relatively to the feature structure in the index. A constraint cannot be set to select annotations whose begin feature value is higher than the end feature value of another one. Rather, we have to specify at the creation the exact value to be higher than. Constraints objects are complex to understand and to set. It requires, for example, seven lines of code for creating an iterator which will get the annotations with a lemma feature. Constraint iterators remain iterators with the same limitations.

The **Tree** method returns an object close to the kind of structure we would like to manipulate to navigate within. Unfortunately, it can only give the children of a covering annotation. So to get the parent of an annotation, a trick could be to build the tree of the whole document by taking the most covering annotation as the root, then to browse the tree until finding the desired annotations for finally getting its parent. But in any case, there is no way to get directly a node and the structure will still suffer from the remarks we made about unambiguous subiterators (consequently some annotations may not be present in the tree).

**Missing Methods** The existing methods partially answer the problem and some navigation methods are missing. There is no dedicated method: to *super-iterate* and to get the annotations covering a given annotation; to move to the first/next annotation of a given type (respectively the last/previous annotation of a given type); or to get partially-covering preceding or following annotations.

All these remarks lead the developers to use preferentially ambiguous iterators and subiterators, even if, this causes to write more code to search the index backward/forward and tests to filter the desired annotations.

## 5 Supporting the spatial navigation

To support a spatial navigation among the annotations we propose to index the annotations by their offsets in a structure called **LocatedAnnotationIndex**, and to merge the annotations occurring at the same spans in a structure called **LocatedAnnotation**. Table 1b illustrates the transformation of the **AnnotationIndex** depicted in Table 1a into a **LocatedAnnotationIndex**. Figure 1 shows the spatial links which interconnect the **LocatedAnnotation**.

The **LocatedAnnotationIndex** is a sorted structure which follows the same sorting order than the **AnnotationIndex**: From a given **LocatedAnnotation**, covering and preceding **LocatedAnnotations** are located backward in the index, and the covered and following **LocatedAnnotations** forward in the index. The structure allows to access directly to a **LocatedAnnotation** thanks to a pair of begin/end offsets. The first characteristic of a **LocatedAnnotation** is to list all the annotations occurring at the same offsets. This prevents from having to define a type priority for handling the limitation of the subiterator. The structure comes with several kinds of links to navigate both within the **LocatedAnnotationIndex** and spatially in the text. Indeed, the structure has links to visit its spatial vicinity (parent/children/following/preceding) **LocatedAnnotation**. The structure has also links to access the previous/next element in the index. The contiguous spa-

tial vicinity of each `LocatedAnnotation` is computed when the `LocatedAnnotationIndex` is built. The API also offers some methods to dynamically search `LocatedAnnotation` containing annotations of a given type among the ancestor/descendant or self. Similarly, it is also possible to search the first/last (respectively following/preceding) `LocatedAnnotation` containing annotations of a given type.

In terms of memory consumption, the built `LocatedAnnotationIndex` takes approximatively as much memory as its `AnnotationIndex`; only the local vicinity of each `LocatedAnnotation` is kept in memory. The CPU time for building the index depends on the `AnnotationIndex` size. Some preliminary tests indicate that the time increases by a factor of three when doubling the size of the annotated text. It takes about 2 seconds for building the index of a 50-sentences text analysed with sentences, chunks and words.

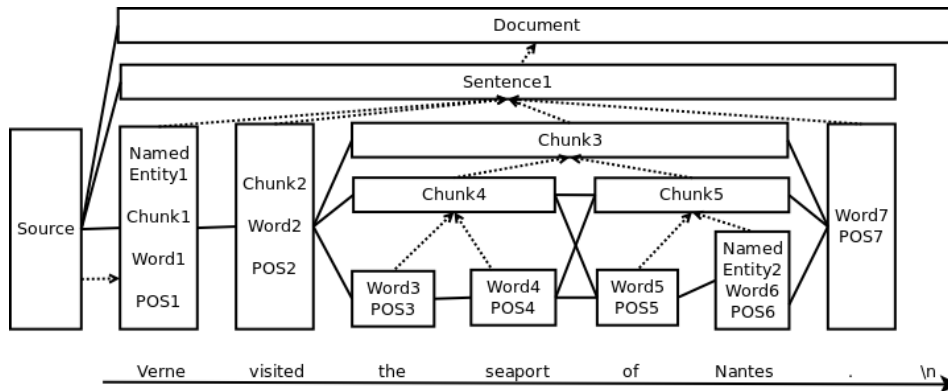


Fig.1: Example of `LocatedAnnotationIndex`. The boxes represent the `LocatedAnnotation`. They are aligned on the text span they cover. The solid lines represent the spatial preceding/following relation while the dotted lines represent the parent/child relations. The parent is indicated by an arrow.

## 6 Related works

With the prospect of developing a pattern matching engine over annotations, [2] have addressed some design considerations for navigating annotation lattices. They have so exposed a language for specifying spatial constraints among annotations. An engine has been implemented within the UIMA framework. Due to this technical choice, the design of the language and its implementation may suffer from the drawbacks we have enumerated. Indeed there is no example of patterns which involve annotation types without inheritance relation. In addition, as pointed out in the perspectives of the authors, it is not clear how the engine will behave when handling multiple annotations over the same spans without the guarantee of a consistent type priority. The `LocatedAnnotation` structure is a solution to the need of defining type priorities. More generally, the methods of our API can play the role of the navigation devices required to the development of a pattern matching engine.

`uimaFIT`<sup>4</sup> is a well-known library which aims at simplifying the UIMA developments. One appealing navigation option it offers is similar to our API. Some methods are designated to move from one annotation to the closest (covering/covered/preceding/following) ones by specifying the type of the annotations to get. In practice, nevertheless, the implementation relies on the UIMA API and may have some of the restrictions. The `selectFollowing` method, for example, follows the *first-contiguous-in-the-index* mechanism. In Table 1a, it returns only the `Chunk` 1 to 3, and misses the 4th and 5th, when calling it successively to get the following chunk from the first chunk.

## 7 Conclusion and perspectives

Solving the interoperability issues in the UIMA framework is a serious problem [1, 6]. Our opinion is to give the means to developers to do what they want. We show that the UIMA API presents some limitations regarding the spatial navigation within annotations in a text. We also show that by adapting his problem definition to the framework requirements the developer may succeed to accomplish his task. But the adaptation has a cost in development time and requires skills in the framework. To overcome this problem, we have developed a library which transforms an `AnnotationIndex` into a navigable structure which can be used in a UIMA component. It is available in the *uima-common* project<sup>5</sup>. Our perspectives are twofold: Reducing the processing time and adding a mechanism for updating the `LocatedAnnotationIndex`.

## References

1. Ananiadou, S., Thompson, P., Kano, Y., McNaught, J., Attwood, T.K., Day, P.J.R., Keane, J., Jackson, D., Pettifer, S.: Towards interoperability of european language resources. *Ariadne* 67 (2011)
2. Boguraev, B., Neff, M.S.: A framework for traversing dense annotation lattices. *Language Resources and Evaluation* 44(3), 183–203 (2010)
3. Ferrucci, D., Lally, A.: Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering* 10(3-4), 327–348 (2004)
4. Gurevych, I., Mühlhäuser, M., Müller, C., Steimle, J., Weimer, M., Zesch, T.: Darmstadt knowledge processing repository based on uima. In: *First Workshop on UIMA at GSCL*. Tübingen, Germany (2007)
5. Hahn, U., Buyko, E., Tomanek, K., Piao, S., McNaught, J., Tsuruoka, Y., Ananiadou, S.: An annotation type system for a data-driven nlp pipeline. In: *The LAW at ACL 2007*. pp. 33–40 (2007)
6. Hernandez, N.: Tackling interoperability issues within uima workflows. In: *LREC*. pp. 3618–3625 (2012)
7. Kano, Y., McCrohon, L., Ananiadou, S., Tsujii, J.: Integrated NLP evaluation system for pluggable evaluation metrics with extensive interoperable toolkit. In: *SETQA-NLP*. pp. 22–30 (2009)

<sup>4</sup> <http://uimafit.googlecode.com>

<sup>5</sup> <https://uima-common.google.com>

# Using UIMA to Structure an Open Platform for Textual Entailment

Tae-Gil Noh and Sebastian Padó

Department of Computational Linguistics  
Heidelberg University  
69120 Heidelberg, Germany  
{noh, pado}@cl.uni-heidelberg.de

**Abstract.** EXCITEMENT is a novel, open software platform for Textual Entailment (TE) which uses the UIMA framework. This paper discusses the design considerations regarding the roles of UIMA within EXCITEMENT Open Platform (EOP). We focus on two points: a) how to best design the representation of entailment problems within UIMA CAS and its type system. b) the integration and usage of UIMA components among non-UIMA components.

**Keywords:** Textual Entailment, UIMA type system, UIMA application

## 1 Introduction

Textual Entailment (TE) captures a common sense notion of inference and expresses it as a relation between two natural language texts. It is defined as follows: *A Text ( $T$ ) entails a Hypothesis ( $H$ ), if a typical human reading of  $T$  would infer that  $H$  is most likely true* [4]. Consider the following example:

**T:** *That was the 1908 Tunguska event in Siberia, known as the Tunguska meteorite fall.*

**H<sub>1</sub>:** *A shooting star fell in Russia in 1908.*

**H<sub>2</sub>:** *Tunguska fell to Siberia in 1908.*

The text (T) entails the first hypothesis (H<sub>1</sub>), since a typical human reader of T would (arguably) believe that H<sub>1</sub> is true. In contrast, T does not entail H<sub>2</sub>. Nor does H<sub>1</sub> entail T, that is, entailment is a directed relation.

The promise of TE lies in its potential to subsume the semantic processing needs of many NLP applications, offering a uniform, theory-independent semantic processing paradigm. Software for the Recognition of Textual Entailment (RTE) have been used to build proof-of-concept versions of various tasks, including Question Answering, Machine Translation Evaluation, Information Visualization, etc. [1, 7].

As a consequence of the theory-independence of TE, there are many different strategies to build RTE systems [1]. This has led to a practical problem of *fragmentation*: Various systems exist, and some have been made available as

open-source systems, but there is little to no interoperability between them, since the systems are, as a rule, designed to implement one specific algorithm to solve RTE. The problem is complicated by the fact that RTE systems generally rely on tightly integrated components such as linguistic analysis tools and knowledge resources. Thus, when a researcher wants to develop a new RTE algorithm, they often need to invest major effort to build a novel system from scratch: Many of the components already exist – but just not in a usable form.

EXCITEMENT open platform (EOP) has been developed to address those problems. It is a *suite* of textual inference components which can be combined into complete textual inference systems. The platform aims to become a common development platform for RTE researchers, and we hope that it can establish itself in the RTE community in a similar way to MOSES [6] in Machine Translation.

Compared to Machine Translation, however, a major challenge is that semantic processing typically depends on linguistic analysis as well as large knowledge sources, which is a direct source of the reusability problems mentioned above. In this paper, we focus on the architectural side of the platform which was designed with the explicit goal of improving component re-usability. We have adopted UIMA (Unstructured Information Management applications) and UIMA CAS (Common Analysis Structure) as the central building blocks for data representation and preprocessing within EOP.

One interesting aspect is that our adoption of UIMA has been *partial* and *parallel*. By *partial*, we mean that there are two groups of sharable components within EOP: the “core” components and the “LAP” components (see Section 2). We have adopted UIMA only for LAPs; however, we use UIMA CAS as one of the standard data containers, even in non-UIMA components. *Parallel* refers to the fact that we allow non-UIMA components to be integrated into our LAPs transparently.

## 2 EXCITEMENT: An Open Platform for Textual Entailment Systems

RTE systems traditionally rely on self-defined input types, pre-processing (linguistic annotation) representations, and resources, tailored to a specific approach to RTE. EXCITEMENT open platform (EOP) tries to alleviate this situation by providing a generic platform for sharable RTE components. The platform has the following requirements.

**Reusing of existing software** : The platform must permit easy integration and re-using of existing softwares, including language processing tools, RTE components, and knowledge resources.

**Multilinguality** : The platform is not tied to a specific language. Adding suites for a new language in the future should not be restricted by the platform design.



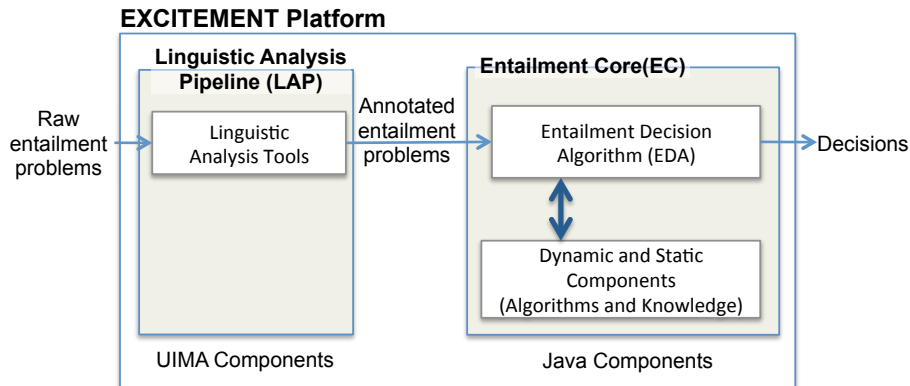


Fig. 1: EXCITEMENT Architecture Overview

**Component Independence** : Components of EOP should be independent and complete as they are. So they can be used by different RTE approaches. This is also true for linguistic annotation pipelines and their components: An annotation pipeline as a whole, or an individual component of the pipeline, can be replaced with equivalent components.

Figure 1 visualizes the top level of the platform. At this level, the platform can be grouped into two boxes: one is the Linguistic Analysis pipeline (LAP), and the other is the Entailment Core (EC). Entailment problems are first analyzed in the LAP, since almost all RTE algorithms require some level of linguistic annotation (e.g., POS tagging, parsing, NER, or lemmatization). The annotated TE problems are then passed to the EC box. In this box, the problems are analyzed by Entailment Decision Algorithms (EDAs), which are the “core” algorithms that make the entailment call and may in turn call other core components to provide either algorithmic additions or knowledge. Finally, the EDA returns an entailment decision.

It is relatively natural to think of the LAP in terms of UIMA, since the typical computational linguistic analysis workflow corresponds well to UIMA’s annotation pipeline concept. Each annotator in LAP adds some annotations, and downstream annotators can use existing annotations and add richer annotations. UIMA CAS and its type system are strong enough to represent any data. UIMA AEs (Analysis Engines) are a good solution for encapsulating and using annotator components. In Section 3, we describe the UIMA adoption in the LAP in more detail.

For Entailment Core (EC) components, however, the situation is different. In contrast to LAP, the functionalities of EC components are often not naturally mapped as “annotation behavior”. To visualize this, let’s check the example in Figure 2. The figure shows a conceptual search process of a RTE system that is based on textual rewriting. In this example, the text is “Google bought Motorola.”, and the system tries to determine hypothesis “Motorola is acquired by

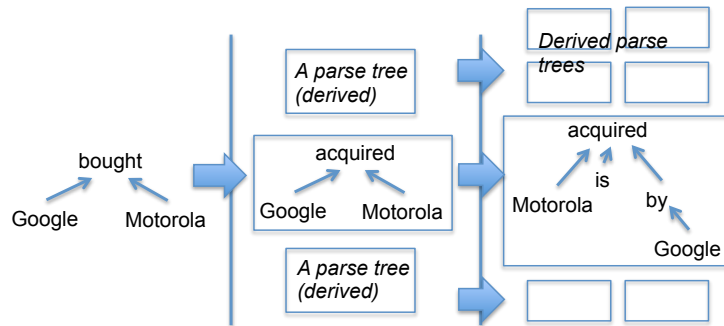


Fig. 2: Entailment as a search on possible rewritings

Google.” as an entailment. The example system gets a dependency parse tree of the text, and starts the rewriting process. On each iteration, it generates possible entailed sentences by querying knowledge bases. In the example, lexical knowledge is used on the first rewriting (*buy* entails *acquire*), and syntactic knowledge (change to passive voice) is used on the second derivation. The process will generate many derived candidates per iteration. The algorithm must employ a good search strategy to find the best rewriting path from text (T) to hypothesis (H).

On this example, there are three major component types. One is the knowledge component type that supports knowledge look-up, another is generation of derived parse trees, and finally the decision algorithm itself drives the search process and makes the entailment decision. Expressing behaviors of such components in terms of annotations on the artifact, might be possible, but is very hard and counter-intuitive.

Following this line of reasoning, we decided that the EC components are better thought of as Java modules whose common behavior is defined by a set of Java interfaces, data types, and contracts, and have defined them accordingly in the EXCITEMENT open platform specification.<sup>1</sup> More specifically, we have defined a typology of components. They include a type for the top-level EDA as well as (currently) five major component types: (1) a feature extractor (get a T-H pair CAS, return a set of features for the T-H pair); (2) a semantic distance calculator (get a T-H pair CAS, return semantic similarity); (3) a lexical resource type (lexical relation database); (4) a syntactic resource type (phrasal relation database); (5) an annotation component (dynamic enrichment of entailment problems).

Although UIMA components are not suitable for conceptualizing inference components, we decided to keep CAS as the data container even in the EC components as far as possible to take advantage of the CAS objects created in the LAP. Thus, various components (including EDAs) gets CAS (as JCas) as an argument on their methods. Also note that LAP and EC boxes are independent:

<sup>1</sup> *Specification and architecture for EXCITEMENT open platform*, <http://excitement-project.eu/index.php/results>

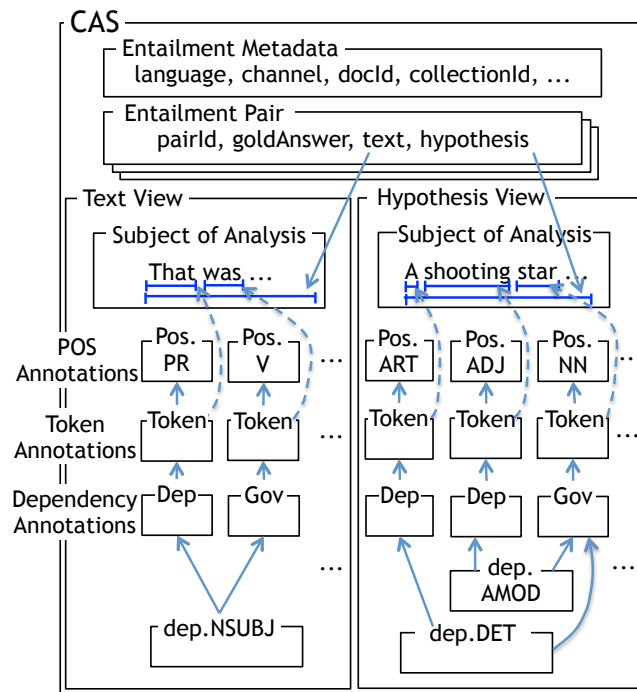


Fig. 3: CAS representation of a Text-Hypothesis pair

as long as the CAS holds correct data, the EC components does not care which pipeline has generated the data.

### 3 Details on the UIMA usage in EXCITEMENT

#### 3.1 CAS for Entailment Problems

The input to any RTE system is a set of *entailment problems*, typically Text-Hypothesis pairs, each of which is represented in one CAS. Figure 3 shows a pictorial example of the CAS data structure for the example pair (T, H<sub>1</sub>) from Section 1. It contains the two text fragments (in two views) and their annotations (here, POS tags and dependencies), as well as global data such as generic metadata (e.g., language) and entailment-specific metadata (e.g., the gold-standard answer).

On the level of the CAS representation, we had to address two points: one is the representation of entailment problems in terms of CASes, the other one is the type definitions.

Regarding the first point, general practice in text analysis use cases is to have one UIMA CAS corresponding to one document. This suggests representing both text and hypothesis (including, if available, their document context) as

separate CASes. However, we decided to store complete entailment problems as individual CASes, where each CAS has two named views (one view for text, the other for hypothesis). This approach has two major advantages: first of all, this enables us to represent cross-annotations between texts and hypotheses, notably alignments, which can be added by annotators. Second, this enables us to define a straightforward extension from “simple” entailment problems (one text and one hypothesis) to “complex” entailment problems (one text and multiple hypotheses or vice versa, as in the “RTE search” task [2]).

Regarding the second point, we adopted the DKPro type system [5], which was designed with language independence in mind. It provides types for morphological information, POS tags, dependency structure, named entities, and co-reference, etc. We extended the DKPro type system with the types necessary to define textual entailment-specific annotation. This involved types for marking stretches of text as texts and hypotheses, respectively, as well as storing correspondence information between texts and hypotheses, pair IDs, gold labels, and some meta data. We also added types for linguistic annotation that are not exclusively entailment-specific, but were not covered yet by DKPro. This included annotation for polarity, reference of temporal expressions, word and phrase alignments, and semantic role labels.

Details about the newly defined types can be found in the platform specification, and the type definition files are part of the platform code distribution.

### 3.2 Wrapping the Linguistic Annotation Pipeline

One decision that may be surprising at the first glance is that we defined our own top-level Java interface for users of the LAP that hides UIMA’s own runtime access methods. This interface dictates the common capabilities that all pipelines of LAP should provide.

The reason for this decision is twofold and pragmatic in nature, making transitioning to and using the EOP as easy as possible for developers.

The first aspect is the learning curve. We would like to avoid the need for Entailment Core developers to deeply understand UIMA AEs and Aggregated Analysis Engines (AAEs). We feel that a deep understanding of these points requires substantial effort but is not really to the point, since many EC developers will only want to use pre-existing LAPs. By making the UIMA aspect of the LAP transparent to the Entailment Core, EC developers do not need to know how the LAP works internally beyond knowledge of the (fairly minimal) LAP interface. Of course, the EC developers still need to understand UIMA CAS very well.

The second aspect is migration cost. If the LAP pipelines were nothing but UIMA AEs, all analysis pipelines of existing RTE systems would have to be deeply refactored, which comes at a considerable cost. Our approach allows such analysis pipelines to be kept largely intact and merely surrounded by a wrapper that provides the requires functionality and converts their output into valid UIMA CASes according to the EOP’s specification.

Nevertheless, there are good reasons to encourage the use of AE-based LAPs: AE-based components are generally much more flexible, and they are very easy to

assemble into AAE pipelines. Therefore, we encourage AE-based LAP development by providing ready-to-use code that implements our LAP interface, taking a list of AEs as input. Thus, if the individual components are already present as AEs, the implementation effort to assemble them into a LAP is near zero. In this sense, we see our LAP interface as a thin wrapper above UIMA with the purpose of enabling peaceful co-existence between UIMA and non-UIMA pipelines. In the long run, we also hope to provide some new AEs back to the UIMA community.

## 4 Some Open Issues

In this section, we discuss two open questions that we are facing in future work.

*CAS in non-UIMA environments.* There is considerable number of best-practice strategies for handling CAS objects (reset the data structure instead of creating a new one; use a CAS pool instead of generating multiple CASes, etc). When a CAS is used in an UIMA context (i.e., in the LAP), it is not hard to guide the developers to follow these rules. However, with CAS being used as a general data container throughout the EOP, developers also often encounter CAS (JCas) objects outside specific UIMA contexts, and we have found it harder to guide the developers towards “proper usage”.

For example, one part of the EXCITEMENT project is concerned with the construction of Entailment Graphs [3], structured knowledge repositories whose vertices are statements and whose edges indicate entailment relations. Since the standard data structure for annotations is JCas, the graph developers tend to add one JCas for each node. This is not problematic for small graphs, but once the graph gets bigger, this can be problematic; CAS is a very large data structure, and its creation and deletion take some time. We are still trying to establish best practices for using CASes in non-UIMA EOP environment.

*Annotation Styles: Hidden dependencies.* One of the EOP design requirement was the clear separation of LAP and EC. This has been fairly well achieved, at least on a technical level.

However, it is clear that there are still *implicit* dependencies between linguistic analysis tools and entailment core components. Consider the case of syntactic knowledge components such as DIRT-style paraphrase rules in the Entailment Core. Such components store entailment rules as pairs of partial dependency trees which have typically been extracted from large corpora. If the corpus used for rule induction was parsed with a different parser than the current entailment problem, then matching the sentence against the rule base will result in missing rules, due to differences in the analysis style. Note that this implicit dependency does not break the UIMA pipeline, since it does not involve the use of a novel type system, but rather differences in the interpretation of shared types. We are currently investigating what type of “style differences” can be observed from actual annotators.

## 5 Conclusion

In this paper, we have provided an overview of the EXCITEMENT open platform architecture and its adoption of UIMA. We have adopted and adapted UIMA CAS and the DKPro type system as a flexible, language-independent data container for Textual Entailment problems. UIMA also provides the backbone for platform’s LAP components. There are several open issues that is to be resolved in the future, but the EXCITEMENT project has already profited substantially from the use of the abstractions that UIMA offers as well as the integration of existing components from UIMA communities.

The first version of EXCITEMENT open platform has been finished<sup>2</sup> with three fully running RTE systems integrated with all core components and annotation pipelines. The platform currently supports three languages (German, Italian and English), and is also shipped with various tools and resources for TE researchers. We believe that the platform will become a valuable tool for researchers and users of Textual Entailment.

*Acknowledgment.* This work was supported by the EC-funded project EXCITEMENT (FP7 ICT-287923).

## References

1. Androutsopoulos, I., Malakasiotis, P.: A Survey of Paraphrasing and Textual Entailment Methods. *Journal of Artificial Intelligence Research* **38** (2010) 135–187
2. Bentivogli, L., Magnini, B., Dagan, I., Trang Dang, H., Giampiccolo, D.: The fifth PASCAL recognising textual entailment challenge. In: *Proceedings of the TAC 2009 Workshop on Textual Entailment*, Gaithersburg, MD (2009)
3. Berant, J., Dagan, I., Goldberger, J.: Learning entailment relations by global graph structure optimization. *Computational Linguistics* **38**(1) (2012) 73–111
4. Dagan, I., Glickman, O., Magnini, B.: The PASCAL Recognising Textual Entailment Challenge. In: *Proceedings of the First PASCAL Challenges Workshop on Recognising Textual Entailment*, Southampton, UK (2005)
5. Gurevych, I., Mühlhäuser, M., Müller, C., Steimle, J., Weimer, M., Zesch, T.: Darmstadt knowledge processing repository based on UIMA. In: *Proceedings of the First Workshop on Unstructured Information Management Architecture at the Conference of the Society for Computational Linguistics and Language Technology*, Tübingen, Germany (2007)
6. Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., Herbst, E.: Moses: Open source toolkit for statistical machine translation. In: *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, Prague, Czech Republic (2007) 177–180
7. Sammons, M., Vydiswaran, V., Roth, D.: Recognizing textual entailment. In Bikel, D.M., Zitouni, I., eds.: *Multilingual Natural Language Applications: From Theory to Practice*. Prentice Hall (2012)

---

<sup>2</sup> The platform has been released under an open source license, and all codes and resources can be freely accessed via the project repository. <http://hltfbk.github.io/Excitement-Open-Platform/project-info.html>

# Bluima: a UIMA-based NLP Toolkit for Neuroscience

Renaud Richardet, Jean-Cédric Chappelier, Martin Telefont

Blue Brain Project, EPFL, 1015, Lausanne, Switzerland  
renaud.richardet@epfl.ch

**Abstract.** This paper describes *Bluima*, a natural language processing (NLP) pipeline focusing on the extraction of neuroscientific content and based on the UIMA framework. Bluima builds upon models from biomedical NLP (BioNLP) like specialized tokenizers and lemmatizers. It adds further models and tools specific to neuroscience (e.g. named entity recognizer for neuron or brain region mentions) and provides collection readers for neuroscientific corpora.

Two novel UIMA components are proposed: the first allows configuring and instantiating UIMA pipelines using a simple scripting language, enabling non-UIMA experts to design and run UIMA pipelines. The second component is a common analysis structure (CAS) store based on MongoDB, to perform incremental annotation of large document corpora.

**Keywords:** UIMA, natural language processing, NLP, neuroinformatics, NoSQL

## 1 Introduction

Bluima started as an effort to develop a high performance natural language processing (NLP) toolkit for neuroscience. The goal was to extract structured knowledge from biomedical literature (PubMed<sup>1</sup>), in order to help neuroscientists gather data to specify parameters for their models. In particular, focus was set on extracting entities that are specific to neuroscience (like brain regions and neurons) and that are not yet covered by existing text processing systems.

After careful evaluation of different NLP frameworks, the UIMA software system was selected for its open standards, its performance and stability, and its usage in several other biomedical NLP (bioNLP) projects; e.g. JulieLab [11], ClearTK [22], DKPro [6], cTAKES [28], ccp-nlp, U-Compare [15], SciKnowMine [26], Argo [25]. Initial development went fast and several existing bioNLP models and UIMA components could rapidly be reused or integrated into UIMA without the need to modify its core system, as presented in Section 2.1.

Once the initial components were in place, an experimentation phase started where different pipelines were created, each with different components and parameters. Pipeline definition in verbose XML was greatly improved by the use

---

<sup>1</sup> <http://www.ncbi.nlm.nih.gov/pubmed>

of UIMAFit [21] (to define pipelines in compact Java code) but ended up being problematic, as it requires some Java knowledge and recompilation for each component or parameter change. To allow for a more agile prototyping, especially by non-specialist end users, a pipeline scripting language was created. It is described in Section 2.2.

Another concern was incremental annotation of large document corpus. For example, when running an initial pre-processing pipeline on several millions of documents, and then annotating them again at a later time. The initial strategy was to store the documents on disk, and overwrite them every time they would be incrementally annotated. Eventually, a CAS store module was developed to provide a stable and scalable strategy for incremental annotation, as described in Section 2.3. Finally, Section 3 presents two case studies illustrating the scripting language and evaluating the performance of the CAS store against existing serialization formats.

## 2 Bluima Components

Bluima contains several UIMA modules to read neuroscientific corpora, perform preprocessing, create simple configuration files to run pipelines, and persist documents on the disk.

### 2.1 UIMA Modules

Bluima’s **typesystem** builds upon the typesystem from JulieLab [10], which was chosen for its strong biomedical orientation and its clean architecture. Bluima’s typesystem adds neuroscientific annotations, like *CellType*, *BrainRegion*, etc.

Bluima includes several **collection readers** for selected neuroscience corpora, like PubMed XML dumps, PubMed Central NXML files, the BioNLP 2011 GENIA Event Extraction corpus [24], the Biocreative2 annotated corpus [16], the GENIA annotated corpus [14], and the WhiteText brain regions corpus [8]. A **PDF reader** was developed to provide robust and precise text extraction from scientific articles in PDF format. The PDF reader performs content correction and cleanup, like dehyphenation, removal of ligatures, glyph mapping correction, table detection, and removal of non-informative footers and headers.

For **pre-processing**, the OpenNLP-wrappers developed by JulieLab for sentence segmentation, word tokenization and part-of-speech tagging [31] were used and updated to UIMAFit. Lemmatization is performed by the domain-specific tool BioLemmatizer [19]. Abbreviation recognition (the task of identifying abbreviations in text) is performed by BIOADI, a supervised machine learning model trained on the BIOADI corpus [17].

Bluima uses UIMA’s ConceptMapper [29] to build **lexical-based NERs** based on several neuroscientific lexica and ontologies (Table 1). These lexica and ontologies were either developed in-house or were imported from existing sources. Bluima wraps several **machine learning-based NERs**, like OSCAR4 [13] (chemicals, reactions), Linnaeus [9] (species), BANNER [18] (genes and proteins), and Gimli [5] (proteins).



Name	Source	Scope	# forms
Age	BlueBrain	age of organism, developmental stage	138
Sex	BlueBrain	sex (male, female) and variants	10
Method	BlueBrain	experimental methods in neuroscience	43
Organism	BlueBrain	organisms used in neuroscience	121
Cell	BlueBrain	cell, sub-cell and region	862
Ion channel	Channelpedia [27]	ion channels	868
Uniprot	Uniprot [1]	genes and proteins	143,757
Biolexicon	Biolexicon [30]	unified lexicon of biomedical terms	2.2 Mio
Verbs	Biolexicon	verbs extracted from the Biolexicon	5,038
Cell ontology	OBO [2]	cell types (prokaryotic to mammalian)	3,564
Disease ont.	OBO [23]	human disease ontology	24,613
Protein ont.	OBO [20]	protein-related entities	29,198
Brain region	Neuronames [3]	hierarchy of brain regions	8,211
Wordnet	Wordnet [7]	general English	155,287
NIFSTD	NIF [12,4]	neuroscience ontology	16,896

**Table 1.** Lexica and ontologies used for lexical matching.

## 2.2 Pipeline Scripting Language

Tool	Advantages	Disadvantages
UIMA GUI	GUI	minimalistic UI, can not reuse pipelines
XML descriptor	typed (schema)	very verbose
raw UIMA java API	typed	verbose, requires writing and compiling Java
UIMAFit	compact, typed	requires writing and compiling Java code

**Table 2.** Different approaches to writing and running UIMA pipelines.

There are several approaches<sup>2</sup> to write and run UIMA pipelines (see Table 2). All Bluima components were initially written in Java with the UIMAFit library, that allows for compact code. To improve the design and experimentation with UIMA pipelines, and enable researchers without Java or UIMA knowledge to easily design and run such pipelines, a minimalistic scripting (domain-specific) language was developed, allowing UIMA pipelines to be configured with text files, in a human-readable format (Table 3). A *pipeline script* begins with the definition of a collection reader (starting with `cr:`), followed by several annotation engines (starting with `ae:`)<sup>3</sup>. Parameter specification starts with a space, followed by the

<sup>2</sup> Other interesting solutions exist (e.g. IBM LanguageWare, Argo), but are not open source.

<sup>3</sup> If not package namespace is specified, Bluima loads Readers and Annotator classes from the default namespace.

parameter name, a column and its value. The scripting language also supports embedding of inline Python and Java code, reuse of a portion of a pipeline with `include` statements, and variable substitution similar to shell scripts. Extensive documentation (in particular snippets of scripts) is automatically generated for all components, using the JavaDoc and the UIMAFit annotations.

### 2.3 CAS Store

A CAS store was developed to persist annotated documents, resume their processing and add new annotations to them. This CAS store was motivated by the common use case of repetitively and incrementally processing the same documents with different UIMA pipelines, where some pipeline steps are duplicated among the runs. For example, when performing resource-intensive operations (like extracting the text from full-text PDF articles, or performing syntactic parsing), one might want to perform these preliminary operation once, store these results, and subsequently perform different experiments with different UIMA modules and parameters. The CAS store thus allows to perform the preprocessing only once, to then persist the annotated documents, and to perform the various experiments in parallel.

MongoDB<sup>4</sup> was selected as the datastore backend. MongoDB is a scalable, high-performance, open-source, schema-free (NoSQL), document-oriented database. No schema is required on the database side, since the UIMA typesystem acts as a schema, and data is validated on-the-fly by the module. Every CAS is stored as a MongoDB document, along with its annotations. UIMA annotations and their features are explicitly mapped to MongoDB fields, using a simple and declarative language. For example, a `Protein` annotation is mapped to a `prot` field in MongoDB. The mappings are used when persisting and loading from the database. As of this writing, annotations are declared in Java source files. In future versions, we plan to store mappings directly in MongoDB to improve flexibility. Persistence of complex typesystem has not been implemented yet, but could be easily added in the future.

Currently, the following UIMA components are available for the CAS store:

- *MongoCollectionReader* reads CAS from a MongoDB collection. Optionally, a (filter) query can be specified;
- *RegexMongoCollectionReader* is similar to *MongoCollectionReader* but allows specifying a query with a regular expression on a specific field;
- *MongoWriter* persists new UIMA CASes into MongoDB documents;
- *MongoUpdateWriter* persists new annotations into an existing document;
- *MongoCollectionRemover* removes selected annotations in a MongoDB collection.

With the above components, it is possible within a single pipeline to read an existing collection of annotated documents, perform some further processing, add more annotations, and store these annotations back into the same MongoDB documents.

---

<sup>4</sup> <http://www.mongodb.org/>

### 3 Case Studies and Evaluation

A first experiment to illustrate the scripting language was conducted on a large dataset of full-text biomedical articles. A second simulated experiment evaluates the performance of the MongoDB CAS store against existing serialization formats.

#### 3.1 Scripting and Scale-Out

```
# collection reader configured with a list of files (provided as external params)
cr: FromFilelistReader
  inputFile: $1
# processes the content of the PDFs
ae: ch.epfl.bbp.uima.pdf.cr.PdfCollectionAnnotator

# tokenization and lematization
ae: SentenceAnnotator
  modelFile: $ROOT/modules/julielab_opennlp/models/sentence/PennBio.bin.gz
ae: TokenAnnotator
  modelFile: $ROOT/modules/julielab_opennlp/models/token/Genia.bin.gz
ae: BlueBioLemmatizer

# lexical NERs, instantiated with some helper java code
ae_java: ch.epfl.bbp.uima.LexicaHelper.getConceptMapper("/bbp_onto/brainregion")
ae_java: ch.epfl.bbp.uima.LexicaHelper.getConceptMapper("/bams/bams")

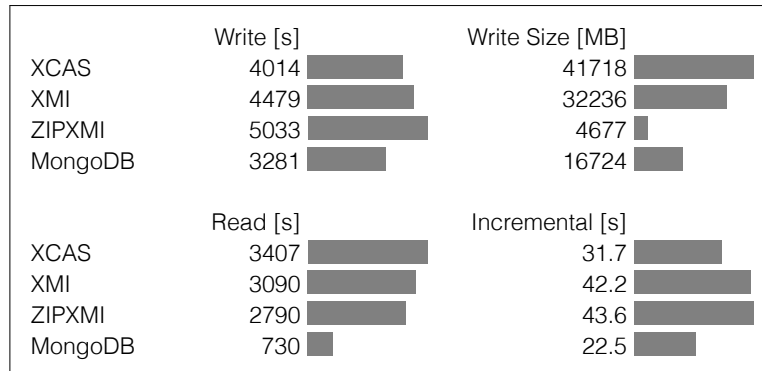
# removes duplicate annotations and extracts collocated brainregion annotations
ae: DeduplicatorAnnotator
  annotationClass: ch.epfl.bbp.uima.types.BrainRegionDictTerm
ae: ExtractBrainregionsCooccurrences
  outputDirectory: $2
```

**Table 3.** Pipeline script for the extraction of brain regions mention co-occurrences from PDF documents.

Bluima was used to extract brain region mention co-occurrences from scientific articles in PDF. The pipeline script (Table 3) was created and tested on a development laptop. Scale-out was performed on a 12-node (144-core) cluster managed by SLURM (Simple Linux Utility for Resource Management). The 383,795 PDFs were partitioned in 767 jobs. Each job was instantiated with the same pipeline script, using different input and output parameters. The processing completed in 809 minutes ( $\simeq 8$  PDF/s).

#### 3.2 MongoDB CAS Store

The MongoDB CAS store (MCS) has been evaluated against 3 other available serialization formats (XCAS, XMI and ZIPXMI). For each, 3 settings were evaluated: writes (CASes are persisted to disk), reads (CASes are loaded from their persisted states), and incremental (CASes are first read from their persisted



**Fig. 1.** Performance evaluation of MongoDB CAS Store against 3 other serialization formats.

states, then further processed, and finally persisted again to disk). Writes and reads were performed on a random sample of 500,000 PubMed abstracts and annotated with all available Bluima NERs. Incremental annotation was performed on a random sample of 5,000 PubMed abstracts and incrementally annotated with the **Stopwords** annotator. Processing time and disk space was measured on a commodity laptop (4 cores, 8GB RAM).

In terms of speed, the MCS significantly outperforms the other formats, especially for reads (Figure 1). The MCS disk size is significantly smaller than XCAS and XMI formats, but almost 4 times larger than the compressed ZIPXMI format. The incremental annotation is significantly faster with MongoDB, and does not require duplicating or overwriting files, like with the other serialization formats. The MCS could be scaled up in a cluster setup, or using solid states drives (SSDs). Writes could probably be improved by turning MongoDB’s “safe mode” option off. Furthermore, by adding indexes, the MCS can act as a searchable annotation database.

## 4 Conclusions and Future Work

In the process of developing Bluima, a toolkit for neuroscientific NLP, we integrated and wrapped several specialized resources to process neuroscientific articles. We also created two UIMA modules (scripting language and CAS store). These additions proved to be very effective in practice and allowed us to leverage UIMA, an enterprise-grade framework, while at the same time allowing an agile development and deployment of NLP pipelines.

In the future, we will open-source Bluima and add more models for NER and relationship extraction. We also plan to ease the deployment of Bluima (and its scripting language) on a Hadoop cluster.

## References

1. Bairoch, A., Apweiler, R., Wu, C.H., Barker, W.C., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., Magrane, M.: The universal protein resource (UniProt). *Nucleic acids research* 33(suppl 1), D154–D159 (2005)
2. Bard, J., Rhee, S.Y., Ashburner, M.: An ontology for cell types. *Genome Biology* 6(2) (2005)
3. Bowden, D., Dubach, M.: NeuroNames 2002. *Neuroinformatics* 1(1), 43–59 (2003)
4. Bug, W.J., Ascoli, G.A., Grethe, J.S., Gupta, A., Fennema-Notestine, C., Laird, A.R., Larson, S.D., Rubin, D., Shepherd, G.M., Turner, J.A.: The NIFSTD and BIRNLex vocabularies: building comprehensive ontologies for neuroscience. *Neuroinformatics* 6(3), 175–194 (2008)
5. Campos, D., Matos, S., Oliveira, J.L.: Gimli: open source and high-performance biomedical name recognition. *BMC Bioinformatics* 14(1), 54 (Feb 2013)
6. De Castilho, R.E., Gurevych, I.: DKPro-UGD: a flexible data-cleansing approach to processing user-generated discourse. In: *Onlineproceedings of the First French-speaking meeting around the framework Apache UIMA, LINA CNRS UMR* (2009)
7. Fellbaum, C.: *WordNet. Theory and Applications of Ontology: Computer Applications* p. 231–243 (2010)
8. French, L., Lane, S., Xu, L., Pavlidis, P.: Automated recognition of brain region mentions in neuroscience literature. *Front Neuroinformatics* 3 (Sep 2009)
9. Gerner, M., Nenadic, G., Bergman, C.: Linnaeus: A species name identification system for biomedical literature. *BMC Bioinformatics* 11(1), 85 (2010)
10. Hahn, U., Buyko, E., Tomanek, K., Piao, S., Mcnaught, J., Tsuruoka, Y., Ananiadou, S.: An Annotation Type System for a Data-Driven NLP Pipeline (2007)
11. Hahn, U., Buyko, E., Landefeld, R., Mülhhausen, M., Poprat, M., Tomanek, K., Wermter, J.: An overview of JCoRe, the JULIE lab UIMA component repository. In: *Proceedings of the LREC*. vol. 8, p. 1–7 (2008)
12. Imam, F.T., Larson, S.D., Grethe, J.S., Gupta, A., Bandrowski, A., Martone, M.E.: NIFSTD and NeuroLex: a comprehensive neuroscience ontology development based on multiple biomedical ontologies and community involvement (2011)
13. Jessop, D., et al.: OSCAR4: a flexible architecture for chemical text-mining. *Journal of Cheminformatics* 3(1), 41 (Oct 2011)
14. Kim, J.D., Ohta, T., Tateisi, Y., Tsujii, J.: GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics* 19, i180–i182 (Jul 2003)
15. Kontonatsios, G., Korkontzelos, I., Kolluru, B., Thompson, P., Ananiadou, S.: Deploying and sharing u-compare workflows as web services. *J. Biomedical Semantics* 4, 7 (2013)
16. Krallinger, M., Morgan, A., Smith, L., Leitner, F., Tanabe, L., Wilbur, J., Hirschman, L., Valencia, A.: Evaluation of text-mining systems for biology: overview of the second BioCreative community challenge. *Genome Biology* 9(Suppl 2), S1 (2008)
17. Kuo, C.J., et al.: BioAdi: a machine learning approach to identifying abbreviations and definitions in biological literature. *BMC Bioinformatics* 10(Suppl 15), S7 (Dec 2009)
18. Leaman, R., Gonzalez, G., et al.: BANNER: an executable survey of advances in biomedical named entity recognition. In: *Pacific Symposium on Biocomputing*. vol. 13, p. 652–663 (2008)
19. Liu, H., et al.: BioLemmatizer: a lemmatization tool for morphological processing of biomedical text. *Journal of Biomedical Semantics* 3(1), 3 (Apr 2012)

20. Natale, D.A., Arighi, C.N., Barker, W.C., Blake, J.A., Bult, C.J., Caudy, M., Drabkin, H.J., D'Eustachio, P., Evsikov, A.V., Huang, H., Nchoutmboube, J., Roberts, N.V., Smith, B., Zhang, J., Wu, C.H.: The protein ontology: a structured representation of protein forms and complexes. *Nucleic Acids Res.* 39(Database issue), D539–545 (Jan 2011)
21. Ogren, P.V., Bethard, S.J.: Building test suites for UIMA components. *NAACL HLT 2009* p. 1 (2009)
22. Ogren, P.V., Wetzler, P.G., Bethard, S.J.: ClearTK: a UIMA toolkit for statistical natural language processing. *Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP* p. 32 (2008)
23. Osborne, J., Flatow, J., Holko, M., Lin, S.M., Kibbe, W.A., Zhu, L.J., Danila, M.I., Feng, G., Chisholm, R.L.: Annotating the human genome with disease ontology. *BMC Genomics* 10(Suppl 1), S6 (Jul 2009)
24. Pyysalo, S., Ohta, T., Rak, R., Sullivan, D., Mao, C., Wang, C., Sobral, B., Tsujii, J., Ananiadou, S.: Overview of the ID, EPI and REL tasks of BioNLP shared task 2011. *BMC Bioinformatics* 13(Suppl 11), S2 (Jun 2012)
25. Rak, R., Rowley, A., Black, W., Ananiadou, S.: Argo: an integrative, interactive, text mining-based workbench supporting curation. *Database: the journal of biological databases and curation* 2012 (2012)
26. Ramakrishnan, C., Baumgartner Jr, W.A., Blake, J.A., Burns, G.A., Cohen, K.B., Drabkin, H., Eppig, J., Hovy, E., Hsu, C.N., Hunter, L.E.: Building the scientific knowledge mine (SciKnowMine1): a community-driven framework for text mining tools in direct service to biocuration. *malta. Language Resources and Evaluation* (2010)
27. Ranjan, R., Khazen, G., Gambazzi, L., Ramaswamy, S., Hill, S.L., Schürmann, F., Markram, H.: Channelpedia: an integrative and interactive database for ion channels. *Frontiers in neuroinformatics* 5 (2011)
28. Savova, G.K., Masanz, J.J., Ogren, P.V., Zheng, J., Sohn, S., Kipper-Schuler, K.C., Chute, C.G.: Mayo clinical text analysis and knowledge extraction system (cTAKES): architecture, component evaluation and applications. *Journal of the American Medical Informatics Association* 17(5), 507–513 (2010)
29. Tanenblatt, M.A., Coden, A., Sominsky, I.L.: The ConceptMapper approach to named entity recognition. In: *LREC* (2010)
30. Thompson, P., et al.: The BioLexicon: a large-scale terminological resource for biomedical text mining. *BMC Bioinformatics* 12(1), 397 (2011)
31. Tomanek, K., Wermter, J., Hahn, U.: A reappraisal of sentence and token splitting for life sciences documents. *Studies in health technology and informatics* 129(Pt 1), 524–528 (2006)

# Sentiment Analysis and Visualization using UIMA and Solr

Carlos Rodríguez-Penagos, David García Narbona, Guillem Massó Sanabre,  
Jens Grivolla, Joan Codina Filbá

Barcelona Media Innovation Centre

**Abstract.** In this paper we present an overview of a UIMA-based system for Sentiment Analysis in hotel customer reviews. It extracts object-opinion/attribute-polarity triples using a variety of UIMA modules, some of which are adapted from freely available open source components and others developed fully in-house. A Solr based graphical interface is used to explore and visualize the collection of reviews and the opinions expressed in them.

## 1 Introduction

With the continuing growth of Social Media such as Twitter, Facebook, and many others, both in terms of volume of content produced daily by users, and in terms of the impact it can have for reputation and decision making (buying, travelling, ...) there is a strong commercial need (and social interest) to efficiently analyze those vast amounts of mostly unstructured information and extract summarized knowledge, while also being able to explore and navigate the content.

We present here a prototype system for analyzing customer reviews of hotels, detecting what people talk about and what opinions they express. The literature agrees on two main approaches for classifying opinion expressions: using supervised learning methods and applying dictionary/rule-based knowledge (see [3] for an overview). The choice of content to be processed also determines what kind of technique yields better results, since longer, more textured text accommodates deeper linguistic analyses including, for example, dependency parsing (see, for example, the use of Machine Learning informed with linguistic analyses in [5]) while shorter, noisy messages, such as those from Twitter microblogs can be tackled with more superficial processing that is strengthened by massive training data and extensive lexical resources (as shown in previous work from some of the authors: [2,4]). Each of them on its own has been used in workable systems (e.g. [6]) and a principled combination of both of them can yield good results on noisy data, since generally one (dictionaries/rules) offers good precision while the other (ML) is able to discover unseen examples and thus enhances recall. In the case at hand, the processing at the level of individual reviews is done using UIMA with a variety of analysis engines using both stochastic and symbolic approaches; the summary of the results, visualization and exploration interface is based on Solr.

## 2 Extraction of Opinionated Units

The prototype presented here focuses on the extraction of customer opinions from full-text unstructured reviews, provided by the users of a big customer review site. We identified as the object of interest for our analysis what we call "opinionated units". These OUs consist in:

- the object of the opinion, i.e. the thing that is being commented on, which we call *Target*.
- the opinion expression, i.e. the words or sentence fragments that represent what is being said about the target, which we call *Cues*.
- the *polarity* of the opinion, as it relates to the target.

Our system proceeds by first detecting possible opinion Targets and possible Cues in the review text. These Target and Cue candidates are then correlated to form Opinionated Units using relevant paths of the syntactic dependencies graph that link the two together. Finally, the polarity of the opinionated unit is established using an *a priori* polarity taken from the Cue (possibly dependent on the type of target) and taking into account quantifiers and negations that appear in the context of the opinionated unit.

For the detection of the possible targets and cues that anchor the OUs, we relied on a JNET annotator that uses Conditional Random Fields over richly annotated vectors (POS, NER, polar words, NP chunks, etc.), and that was trained with a manually annotated corpus of similar customer reviews.<sup>1</sup> We used this supervised approach since the hotel review domain is pretty regular inasmuch the kinds of things and features people comment on, but we wanted to leave open the possibility to discover items and concepts outside a closed list.

### 2.1 Recognizing Opinionated Units and their polarity

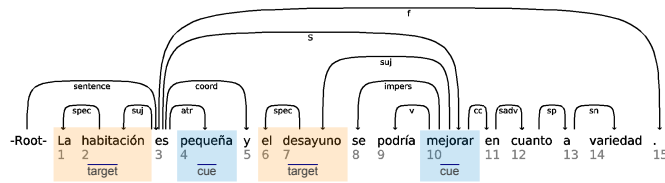
In order to detect candidate opinion-bearing linguistic structures we parsed the sentences with the DeSR dependency parser[1]. We looked for possible paths in the graph linking Cues to Targets, as show in Figure 1, where the Target "la habitación" (the room) is qualified by "pequeña" (small) and the Target "el desayuno" (breakfast) is being described as "with room for improvement".

We identified both correct and wrong paths between Targets and Cues in annotated documents and analysed them. This allowed us to extract relevant patterns, and individualize opinionated units even if they were expressed in the same phrase. The most important patterns are structures of linking verbs and name-adjective relations, but there are prepositional phrases, adverbial phrases and subject-verb relations as well. All the relevant paths can be represented by

---

<sup>1</sup> On evaluating this process, we allowed for partial overlap (e.g. "The room" and "room" counted as equally correct answers), and we obtained models that had F1 of 0.69 for Targets only, 0.54 for Cues and a combined Target/Cue identification model that provided an F1 of 0.62, with a top precision of 0.84 for the Cue only model and a top coverage (or recall) of 0.63 for Target-only models.

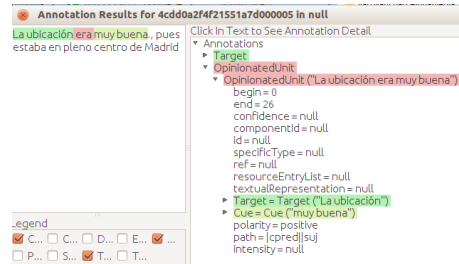




**Fig. 1.** Opinionated Units in the dependency graph

a limited number of regular expressions, which are used to correlate Targets and Cues of the same OU. This approach maximized precision at the expense of recall, focusing further analyses only in semantically relevant fragments, as identified through Targets and Cues.

We used different strategies and tools to detect the possible polarity of an Opinionated Unit, since each one has both advantages and weaknesses. A first strategy is to assign polarities to Cues, and then expand this polarity to the Opinionated Unit, using an approach based on the words sequence (Conditional Random Fields). A second strategy uses Support Vector Machines on a "bag of features" that includes words, polar words and their polarity, negations, and quantifiers to build a feature vector used for training and classification. After



**Fig. 2.** Type representation of the Opinionated Unit visualized with the Annotation Viewer

statistical models have been applied, we also used heuristics that combined those polarities with the polarities of key words (detected using dictionaries) in the context of the OUs, in order to assign a final polarity to the Opinionated Unit. Our UIMA type for Opinionated Units representing the Target-Polarity-Cue triplet has pointers to corresponding Targets and Cues from the relevant dependency graph, as well as the span and ultimate polarity of the complete object covered by it, as shown in Figure 2 for the text "The location [Target] was very good [Cue]".

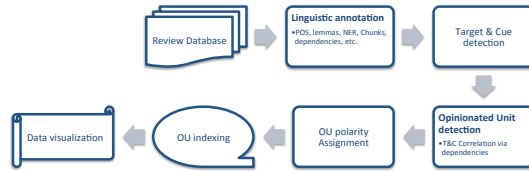


Fig. 3. System overview

### 3 Architecture and Implementation

This section describes all UIMA modules used in the prototype, as implemented in Figure 3. Some of them are existing open source components, some are adaptations, and some are our own custom developments. We have been publishing our work on Github and will continue doing so as far as possible.<sup>2</sup>

**UIMA Collection Tools** This prototype is designed to work on a static document collection, previously loaded into a MySQL database (including the review text as well as associated metadata). UIMA Collection Tools<sup>3</sup> is an *ecosystem* of tools for allowing UIMA pipelines to store and retrieve data from database systems, such as MySQL. Plain text documents can be retrieved from a database, XMI documents can be retrieved from and stored in a database either compressed or uncompressed, features can be extracted into a database table, and annotations within database-stored XMI blobs can be visualized the same way as the standard AnnotationViewer does for XMI files.

- *DBCollectionReader* is a UIMA collection reader which retrieves plain text documents stored in a MySQL database. Database connection parameters as well as SQL query have to be specified in the component descriptor. It is derived from the *FileSystemCollectionReader*.
- *SolrCollectionReader* is equivalent to *DBCollectionReader*, but using a Solr index as the document source.
- *DBXMICollectionReader* is a UIMA collection reader that retrieves XMI documents stored in a MySQL database. *DBXMICollectionReader* is also prepared to read compressed XMI documents by means of ZLIB compression. This option can be set in the descriptor file.
- *DBAnnotationsCASConsumer* is a CAS consumer which stores values of the features specified in the component descriptor file in a MySQL database table. Each table row corresponds to the annotation defined as the *splitting annotation*, e.g. if Sentence annotation has been defined as the *splitting annotation*, each table row will correspond to a Sentence, and this row will

<sup>2</sup> See <https://github.com/BarcelonaMedia-ViL/>

<sup>3</sup> The UIMA Collection tools have been developed at Barcelona Media, some of them based on the example Collection Readers and CAS Consumers provided with the UIMA distribution. They are published under the Apache License at <https://github.com/BarcelonaMedia-ViL/uima-collection-tools>.

contain features of the Sentence annotation and/or features of annotations covered by the Sentence annotation.

- *DBXMICASConsumer* is a CAS consumer that persists XMI documents in a database. *DBXMICASConsumer* is also prepared to store compressed XMI documents by means of ZLIB compression.
- *DBAnnotationViewer* is a modification of the Annotation Viewer, and allows reading XMI files directly from a MySQL database without needing to extract them first.

**OpenNLP** We use OpenNLP<sup>4</sup> with the standard UIMA wrappers for our base pipeline, including Sentence Detector, Tokenizer, and POS Tagger, using our own trained models for Spanish.

**Lemmatizer** We apply Lemmatization using a large dictionary developed in-house. All candidate lemmas are first added to the CAS using *ConceptMapper*<sup>5</sup> but a second custom component selects the right one using the POS tag.

**JNET** For ML-based detection of Targets and Cues we use JNET<sup>6</sup> (the Julielab Named Entity Tagger), which is based on Conditional Random Fields (CRF). It detects token sequences that belong to certain classes, taking into account a variety of features associated with each token (such as the surface form, lemma, POS tag, surface features such as capitalization, etc.) as well as its context of preceding and successive tokens. While originally intended for Named Entity Recognition, we trained JNET with our own manually annotated corpus.

Compared to the original JNET as released by JulieLab we introduced a series of changes, most importantly making it type system independent by taking all input and output types and features as parameters, and fixing some bugs that were triggered when using a larger amount of token features. We expect to release our changes soon, but are still looking into the question of licensing, to comply with JNET's original license.

**DeSR** We developed a UIMA wrapper for the DeSR dependency parser<sup>7</sup>. The parser creates dependency annotations based on previously generated sentence, token and POSTag annotations. It is available at <https://github.com/BarcelonaMedia-ViL/desr-uima>. The UIMA DeSR analysis engine is a UIMA C++ annotator, developed using the C++ SDK provided by UIMA. It translates between the format required by the DeSR parser shared library and the UIMA CAS format. The mapping between UIMA types and features and the features used internally by DeSR is configurable in the annotator descriptor.

<sup>4</sup> <http://opennlp.apache.org/>

<sup>5</sup> <http://uima.apache.org/sandbox.html#concept.mapper.annotator>

<sup>6</sup> [http://www.julielab.de/Resources/Software/NLP\\_Tools.html](http://www.julielab.de/Resources/Software/NLP_Tools.html)

<sup>7</sup> <https://sites.google.com/site/desrparser/>

**DependencyTreeWalker** This is a Pythonnator-based analysis engine for wrapping the DependencyGraph Python module (both developed in-house). This allows us to work easily with the dependency graph generated by DeSR in order to e.g. determine and validate the path between two given UIMA annotations.

**Weka Wrapper** We used the Mayo Weka/UIMA Integration (MAWUI<sup>8</sup>), as a basis for the machine learning tools. The version we use is adapted to newer versions of UIMA and made much more configurable. MAWUI generates a single vector for each document, that is used to classify it as a whole. In our case, a document can contain several Opinionated Units that need to be classified. For this reason the Weka Wrapper was adapted to be able to deal with all the annotations of a given type inside a document (or collection when generating the training data).

## 4 Visualization

Beyond being able to extract and classify the opinions, users need an interface that allows them to access and explore the data. They need to know which are the Targets or its features that are being addressed by the opinions and what is being said about them, and this has to be shown in an aggregated way, with drill-down capabilities, so that the end user has a clear view of the contents of hundreds or thousands of opinions.

UIMA does not provide tools to deal with collections of documents, and we use Solr, a Lucene based indexing tool, to index the Opinionated Units. Through the use of Solr's faceting and pivot utilities we are able to graphically summarize thousands of opinions. Special charts have been dconstructed in order to allow not only to represent the data but also to select subsets of opinions and summarize and compare them. For example, we can compare the global user's opinions with the opinions about a single hotels or the hotels in a specific area.

To index the data we needed the linguistic information, but also the metadata associated with the opinion, which is located in databases and is not processed with UIMA. For this reason we import the data to Solr in two steps. In a first step we generate from UIMA a table with the data that we then import to Solr together with the metadata.

### 4.1 Indexing Opinionated Units

To index the Opinionated Units we use the DBAnnotationsCASConsumer component. We generate a register for each OU, containing: the Target, the Cue, the text span, the polar words, their polarity, the polarity of the cue, and the polarity of the Opinionated Unit. Cues and targets are grouped in single tokens by means of underscoring.

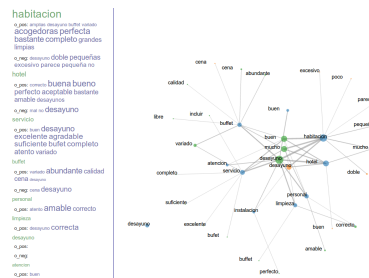
---

<sup>8</sup> <http://informatics.mayo.edu/text/index.php?page=weka>

We use the the `DataImportHandler` from Solr in order to import the data from the database. To do it, a query combines the opinionated unit information with the one related with the hotel or the user who writes the opinion. Cues are indexed twice, once all merged and later in different fields depending on the opinion’s polarity, making it easy to retrieve just the positive or negative opinion markers. We selected this option because it is a bit faster, more flexible and reliable than the other ones: when indexing directly from UIMA we have problems in adding all the desired metadata, and if we call UIMA from Solr (or Lucene) then it is difficult to have a general framework that splits a single document into several Opinionated Units.

AJAX-Solr<sup>9</sup> is a JavaScript library for creating user interfaces to Apache Solr. This library works with facets. Faceting is a capability of Solr that allows to have a fast statistic of the most frequent terms in each field, after performing a query. Since version 4.0 Solr also has pivots that combine the facets from two or more different fields. We adapted AJAX-Solr to work with pivots and wrote a series of widgets to visualize them. Our own extensions to AJAX-Solr are also published on github<sup>10</sup>.

By means of clicking the different facets that appear on the widgets, the user can build a query that restricts the set of opinions to summarize. These opinions are then summarized by showing the most frequent terms they contain, or the most differentiating ones (i.e. those terms that are frequent in the current subset but that are less frequent in the general one). Figure 4 shows the pivot result in text and force diagram formats. It shows the relationship between Targets, and positive and negative Cues. In the textual representation, the relationships are not shown directly but scaled to magnify the most discriminative ones.



**Fig. 4.** Visualization of Cue and Target correlations across the whole corpus

<sup>9</sup> <https://github.com/evolvingweb/ajax-solr>

<sup>10</sup> <https://github.com/BarcelonaMedia-ViL/ajax-solr>

## 5 Conclusion

The combination of UIMA and Solr has allowed us to develop a very flexible platform that makes it easy to integrate and combine processing modules from a variety of sources and in a variety of programming languages, as well as navigate and visualize the results easily and efficiently.

In our evaluations with 700 OUs manually annotated by 3 independent reviewers, there was an agreement on the correctness of the OU identified by the system of 88.5%, while the polarity assigned was found to be correct an average of 70%.

We found many useful UIMA components to be available as open source, and encountered few compatibility issues (other than adapting some components to be type system independent). Solr provides us with a very flexible platform to access large document collections, and in combination with UIMA allows us to explore even complex hidden relationships within those collections.

One of our main objectives was to make all modules configurable and reusable, inasmuch as Sentiment Analysis in general requires tweaking to adapt to domain and genre, but this generalization often requires considerable effort. We found the different open source communities to be very receptive, and we try to participate by publishing our own contributions under permissive licenses that make them easy for others to adopt and use.

## 6 Thanks

This work has been partially funded by the Spanish Government project Hologopedia, TIN2010-21128-C02-02, and the CENIT program project Social Media, CEN-20101037.

## References

1. G. Attardi, F. Dell’Orletta, M. Simi, A. Chanev, and M. Ciaramita. Multilingual dependency parsing and domain adaptation using *desr*. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, page 1112–1118, 2007.
2. Jose M. Chenlo, Jordi Atserias, Carlos Rodriguez, and Roi Blanco. FBM-Yahoo! at RepLab 2012. In *CLEF (Online Working Notes/Labs/Workshop)*, 2012.
3. Bing Liu. Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167, 2012.
4. Carlos Rodríguez-Penagos, Jordi Atserias, Joan Codina-Filba, David García-Narbona, Jens Grivolla, Patrik Lambert, and Roser Saurí. FBM: combining lexicon-based ML and heuristics for social media polarities. In *SemEval 2013*, 2013.
5. Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity: An exploration of features for phrase-level sentiment analysis. *Computational Linguistics*, 35(3):399–433, December 2010.
6. L. Zhang, R. Ghosh, M. Dekhil, M. Hsu, and B. Liu. Combining lexicon-based and learning-based methods for twitter sentiment analysis. *HP Technical Report HPL-2011-89*, 2011.

# Extracting hierarchical data points and tables from scanned contracts

Jan Stadermann, Stephan Symons, and Ingo Thon

Recommind Inc., 650 California Street, San Francisco, CA 94108, United States  
{jan.stadermann,stephan.symons,ingo.thon}@recommind.com  
<http://www.recommind.com>

**Abstract.** We present a technique for developing systems to automatically extract information from scanned semi-structured contracts. Such contracts are based on a template, but have different layouts and client-specific changes. While the presented technique is applicable to all kinds of such contracts we specifically focus on so called *ISDA credit support annexes*. The data model for such documents consists of 150 individual entities some of which are tables that could span multiple pages. The information extraction is based on the Apache UIMA framework. It consists of a collection of small and simple *Analysis Components* that extract increasingly complex information based on earlier extractions. This technique is applied to extract individual data points and tables. Experiments show an overall precision of 97% with a recall of 93% regarding individual/simple data points and 89%/81% for table cells measured against manually entered ground truth. Due to its modular nature our system can be easily extended and adapted to other collections of contracts as long as some data model can be formulated.

**Keywords:** OCR robust information extraction, hierarchical taggers, table extraction

## 1 Introduction

Despite the existence of electronic document handling and content management systems there is still a large amount of paper based contracts. Even when scanned and OCRed the interesting data contained in the document is not machine-readable as there is no semantic attached to the text. Especially in the banking domain it is necessary to have the underlying information available, e.g., for risk assessment. Until now, the information has to be extracted by human reviewers. The goal of the system presented here is to automatically obtain the relevant information from OTC (over-the-counter) contracts which are based on a template provided by the ISDA<sup>1</sup>. The data is given in the form of image-embedded pdf documents. Each contract contains around 150 data points organized in a complex hierarchical data model. A data point can be either a (possibly multi valued) simple field or a table. The main challenges of such a system are:

---

<sup>1</sup> International Swaps and Derivatives Association, [www.isda.org](http://www.isda.org)

<b>(a) Base Currency and Eligible Currency.</b>				
	(i)	"Base Currency" means United States Dollars unless otherwise specified here: Euro.		
<b>(a)</b>	(ii)	"Eligible Currency" means the Base Currency and each other currency specified here: United States Dollars.		
<b>(b)</b>				
		<b>Party A</b>	<b>Party B</b>	<b>Valuation Percentage</b>
(A)	Cash, in the Eligible Currency	No	No	--
(B)	negotiable debt obligations issued by the Governments of the United States of America and Germany (excluding inflation-linked bonds) having a residual maturity of not more than one year.	Yes	Yes	98%
(C)	Negotiable debt obligations issued by the Governments of the United States of America and Germany (excluding inflation-linked bonds) having a residual maturity of more than one year but not more than five years	Yes	Yes	97%

**Fig. 1.** (a) Example simple valued fields (`base_currency` and `eligible_currency`). Note that for the `eligible_currency` one or more currencies can be specified. (b) Example table (collateral eligibility).

1. The complex legal language used in the contracts.
2. Despite existing contract templates, the wording varies across customers.
3. The layout varies. Especially tables can be represented in various forms.
4. The scanning quality of the contracts is often poor, especially in old contracts or documents sent by fax. Still the remaining information needs to be extracted correctly.

Figure 1 shows examples of two simple data points (a), and a table (b).

In general, on the one hand, there are a lot of sophisticated entity extraction systems that try to find flat entities only ("Named entity extraction") [9]. These systems sometimes use hierarchical information, like Tokens, Part-Of-Speech-Tags, Sentences, but only on a linguistic level without collecting and combining this information. These approaches work well on well-defined and general entities such as persons or locations. However, they are difficult to adapt to a new domain since a new classifier needs to be created which requires huge amounts of labeled training data which is expensive to produce.

On the other hand, there are systems that use a deep hierarchical structure, e.g. represented using Ontologies, but still do the classification in one single, flat step [1]. This approach is not as flexible and extensible compared to the presented one since in general it requires a re-training or re-building of the classifier if layers within the hierarchy are changed. An early solution for dealing with scanned forms was presented by Taylor et al., who used a model-based approach for data extraction from tax forms [12]. Semi structured texts have been analyzed using



rule based approaches [10] or discriminative context free grammars [13]. Closest to our solution is a system described by Surdeanu et al. [11]. They employ two layers of extraction using Conditional Random Fields [5], and deal with OCR data. For table extraction, heuristic methods [8] have been proposed as well as Conditional Random Fields [7].

In contrast, our system uses a theoretically unlimited number of layers with separate classifiers for each piece of information, including tables, on each level. Instead of processing the whole text at once, our classifiers just collect the information they require, and decide only on that data. Therefore, they allow for better performance and extensibility, as additional data does not affect the existing classifiers. Our work follows strategies commonly used in spoken dialogue systems [4] and uses a set of small classifiers which is inspired by the boosting idea [6]. In addition, we use automatically extracted segmentation information and cross-checks between our classifiers to increase the precision of the extracted data. From a UI standpoint there is a similar application called GATE [2] which extracts entities based on given rule-sets. This application provides a hierarchical organization of entities and the architecture seems to be very similar to the UIMA framework. However, GATE has no special provisions to deal with noise from due to the OCR step and it only allows to specify simple extraction rules. Furthermore there is no direct way that the entity extraction works hierarchically but only the result can be organized in a hierarchical way.

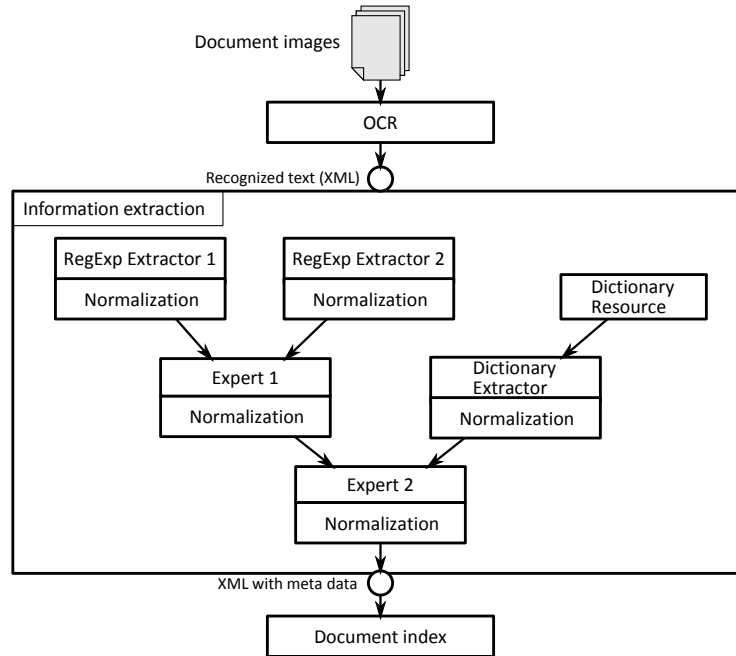
## 2 Information extraction

An overview of our system's architecture is shown in figure 2. Prior to information extraction, the OmniPage<sup>2</sup> OCR engine is used to convert the image to readable text. However, many character level errors, and layout distortions remain which need to be dealt with in the following processing steps. The overall strategy is based on the idea that small pieces of relevant text can be extracted quite accurately even in the presence of OCR errors. On top of these pieces we build several layers of higher level extractors – here called "experts" – that combine these small pieces to decide on a final data point. The extraction of tables works in a similar fashion by first trying to extract small pieces that form table cells. Then stretches of cells are collected, trying to deduce a layout from order and type of the pieces. Finally, an optimal result table is selected (see section 2.2).

Our solution is based on the UIMA framework [3]. Each type of expert is implemented as a configurable annotation engine. The overall extraction system consists of a large hierarchy of analysis engines, encompassing several hundred elements. The type system, in contrast, only consists of three principal types, i.e. for simple fields, tables and table rows. Annotation types, extracted values, etc. are stored as features. Both final and intermediate annotations are represented by these types.

---

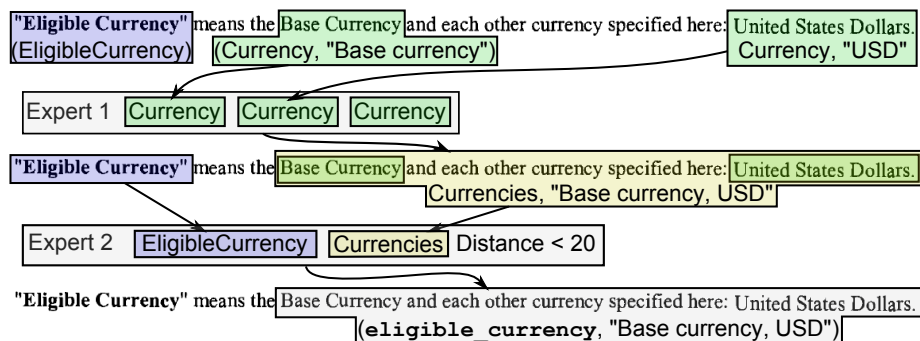
<sup>2</sup> <http://www.nuance.com/omnipage>



**Fig. 2.** Extraction architecture

## 2.1 Extraction of simple-valued fields

We use the term “simple-valued fields” for data points, where one key has one or more values. They differ from named entities as they may include multi-valued data. Figure 1(a) shows an example of the key `eligible_currency` with the (normalized) values “USD” and “Base currency”. Fields are extracted layer-wise. On the lowest layer, all instances of the identifying term “Eligible currency”, are captured, as well as the different currency expressions, including the special term “Base currency”, which refers to another simple field. On this level we typically use annotators based on dictionaries and regular expressions, where variations due to OCR errors are reflected in dictionary variants, respectively the regular expressions. All such annotators are implemented as analysis engines. On the next level, so-called “expert-extractors” combine the existing annotations to a new one. An expert is a rule, defined as a set of slots for annotations of specific types, and a definition of which slots form a new annotation if the rule is satisfied, i.e. if all slots are filled. To allow for fine tuning the experts, slots can be configured, e.g. by indicating certain slots as optional. Furthermore, it is possible to specify the order of annotations in slots appearing in the document. It is also possible to specify a maximum distance. If the distance between two found annotations exceeds the defined threshold for this expert, the expert assumes to be in the wrong area of the document and clears its internal state to start all over



**Fig. 3.** Extraction of a simple field. First level components have tagged the “Eligible Currency” phrase and the different variants of currencies. Expert 1 collects two or more currencies (the third slot is optional). The resulting annotation is used by Expert 2 to build the final annotation. All elements are represented in UIMA as simple fields types.

again. Finally, slots can be write-protected, accepting only the first occurrence of the configured annotation.

To extract `eligible_currency`, two experts are employed (see figure 3). The first expert collects adjacent currency annotations. The second one combines the “Eligible Currency” term, and the collected currencies found by expert one, if both annotations are found within a short distance. The resulting annotation will span the relevant currency terms. This modular design allow us to reduce the number of extractors and re-use the already made annotations for completely different data points. In general, the information found in the examined contracts is not independent of each other. We use business rules and other constraints to validate and normalize the found results, e.g., the set of currencies is well-defined. If the validation fails or the normalization repairs some value due to business rules a corresponding message can be attached to the annotation to inform the reviewer.

## 2.2 Extraction of tables

We define a table as multi-dimensional, structured data present in a document either in a classical tabular layout, or defined in a series of sentences or paragraphs in free text form (like in figure 4). We aim at extracting tables of both structure types and intermediate formats (e.g. as in figure 1(b)) only from the document’s OCR output at character level. In our application, table extraction extends the simple valued field extraction: The basic input for a table expert is a document annotated with simple value fields and intermediate annotations. The experts attempt to match sequences of simple annotations to a set of table models. A table model is user-defined and describes which columns the resulting extracted table should have. Each column can contain multiple types of simple

fields. Furthermore, columns can be configured to be optional and to accept only unique or non-overlapping annotations. This allows for both more general models with variable columns and fine-tuning the accepted annotations.

The process of detecting tables by the table expert (see figure 4 for an example) begins with collecting all accepted annotations for a model, within a predefined range or until a table stop annotation is found, into a list sorted by order of appearance. For each such list, several *filling strategies* are employed. A filling strategy addresses the problem that multiple columns may accept the same types of annotations. If elements appear row-wise, or column-wise, the corresponding strategies will recover the correct table, also compensating for some errors from omitted table elements. In mixed cases, adding a new table cell to the shortest relevant column is used as a fall back strategy. Each strategy is evaluated, using the fraction of cells filled in the resulting table  $c$  and the filling strategy specific score  $s$ . The latter score measures how well the annotations match the expectations of the filling strategy. The table which maximizes  $s_f = c \cdot s$  is annotated as a candidate, if  $s_f$  is above a predefined threshold. The table expert is implemented an analysis engine. Configuration encompasses the columns describing the table model, distance and scoring threshold, and the set of filling strategies to be evaluated. The output is a table type annotation, which in turn contains several table rows, each containing simple fields as cells.

Multiple table experts may be used to generate candidate tables for a single target, and candidates may occur in several locations in a document. Usually, the correct location gives raise to tables with certain properties, e.g. short, dense tables. This is used by a feature-based selection of the optimal table candidate. We model this using both general purpose features (e.g. size, and number of empty cells) as well as domain specific features. The table with the highest weighted sum of score features is selected as the final output. The weights can either be user defined or fitted using a formal optimization model.

### 3 Experiments

We composed a document set containing 449 documents<sup>3</sup> to measure the extraction quality of our system. These documents are from various customers and represent as many variants of different wordings and layouts as possible.

With our customers we agreed upon certain quality gates that the automatic extraction system has to meet. Due to the nature of the contracts it is much more important to achieve a high precision of the extracted data instead of recall. For simple fields the gate's threshold is 95% precision and 80% recall. Table cells are more difficult to extract since the OCR component not only mis-recognizes individual characters but makes errors on the structure of a table. For table cells, our goal is to have a high recall since errors within a structured table are easier to detect and correct than simple field errors by a human reviewer. Table 1 shows our results against a manually created ground truth. The numbers represent the

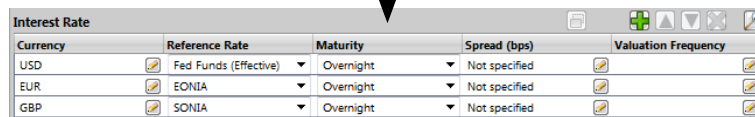
---

<sup>3</sup> see [tinyurl.com/csa-example](http://tinyurl.com/csa-example) for a public sample document.

	Insertions	Deletions	Substitutions	Correct	Precision	Recall
Simple fields	375	1267	330	20519	0.966	0.928
Table cells	1492	3563	906	18838	0.887	0.808

**Table 1.** Results for simple fields and table cells on our document corpus, shown as absolute numbers of (in)correct data points and values for precision and recall.

(1) *Interest Rate.* The "Interest Rate" in relation to Eligible Currency denominated in US Dollars, for any day will be , the rate opposite the caption "Federal Funds ( Effective Rate )" for such day as published by the Federal Reserve Publication H . 15 ( 519 ) or any successor publication as published by the Board of Governors of the Federal Reserve System . The " Interest Rate " in relation to Eligible Currency denominated in Euro, for any day will be , the rate EONIA ( Euro Over Night Index Average ) for such day as displayed on Reuters screen EONIA = . The " Interest Rate " in relation to Cash denominated in Sterling will be , the rate SONIA ( Sterling Overnight Index ) for such day as displayed on Reuters screen SONIAOSR = .



Currency	Reference Rate	Maturity	Spread (bps)	Valuation Frequency
USD	Fed Funds (Effective)	Overnight	Not specified	
EUR	EONIA	Overnight	Not specified	
GBP	SONIA	Overnight	Not specified	

**Fig. 4.** Textual source (OCR output rendered as rich text) and extraction result of an interest rate table. The tabular result is extracted from the paragraph with three similar sentences which contain currencies (solid frame) and interest rate names (dashed frame). Domain knowledge is used to fill the maturity and spread columns.

total number of data points and errors respectively over all of our documents. In total, we meet our gate criterion for simple fields. Precision can be as low as 33% for rare fields, where fitting appropriate data experts is hard. In contrast, for frequent fields, precision may exceed 99%. In principle, the same is true for recall, with both maximum and minimum lower, due to our target criteria. For table cells, the precision needs improvement mainly due to the OCR's structural errors like swapping rows within a table or switching between row-wise and column-wise recognition in one table. This is especially true for tables which are complex with respect to both lay-out and contents, like the collateral eligibility table in figure 1(b). Here, precision and recall are 84.4% and 80.2%, respectively. In contrast, structurally simple tables, like the interest rate table (see figure 4 for an example) can be extracted with much higher confidence (97.4% precision and 90.8% recall).

## 4 Conclusion and outlook

This article presents a system to automatically extract simple data points and tables from OTC contract images. The system consists of an OCR component and a hierarchical set-up of small modular extractors either capturing (noisy) text or combining already annotated clues using a slot-filling strategy. Our experiments are conducted on a in-house contract collection resulting in a precision of 97% (recall 93%) on simple fields and a precision of 89% (recall 81%) on table cells. While the evaluation we conducted is limited, we expect overfitting to be

moderate. The legal nature of the contracts limits the layout and wording options. Our next steps include the introduction of a confidence score on data-point level and the use of statistical classification methods for selecting the best-suited table model.

*Acknowledgement.* We would like to thank our partner Rule Financial for providing the data model and for their assistance in understanding the documents.

## References

1. Paul Buitelaar and Srikanth Ramaka. Unsupervised ontology-based semantic tagging for knowledge markup. In *Proceedings of the Workshop on Learning in Web Search at the International Conference on Machine Learning*, 2005.
2. Hamish Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.
3. David Ferrucci and Adam Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.
4. Kyungduk Kim et al. A frame-based probabilistic framework for spoken dialog management using dialog examples. In *Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue*, 2008.
5. John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, 2001.
6. Ron Meir and Gunnar Rätsch. An introduction to boosting and leveraging. In *Advanced lectures on machine learning*, pages 118–183. Springer, 2003.
7. David Pinto, Andrew McCallum, Xing Wei, and W Bruce Croft. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 235–242, 2003.
8. Pallavi Pyreddy and W Bruce Croft. Tintin: A system for retrieval in text tables. In *Proceedings of the second ACM international conference on Digital libraries*, pages 193–200, 1997.
9. Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the thirteenth conference on Computational Natural Lanugage Learning*, pages 147–155, 2009.
10. Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine learning*, 34(1-3):233–272, 1999.
11. Mihai Surdeanu, Ramesh Nallapati, and Christopher D. Manning. Legal claim identification: Information extraction with hierarchically labeled data. In *Proceedings of the LREC 2010 Workshop on the Semantic Processing of Legal Texts*, 2010.
12. Suzanne Liebowitz Taylor, Richard Fritzson, and Jon A Pastor. Extraction of data from preprinted forms. *Machine Vision and Applications*, 5(3):211–222, 1992.
13. Paul Viola and Mukund Narasimhan. Learning to extract information from semi-structured text using a discriminative context free grammar. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 330–337, 2005.

# Constraint-driven Evaluation in UIMA Ruta

Andreas Wittek<sup>1</sup>, Martin Toepfer<sup>1</sup>, Georg Fette<sup>1,2</sup>,  
Peter Kluegl<sup>1,2</sup>, and Frank Puppe<sup>1</sup>

<sup>1</sup> Department of Computer Science VI, University of Wuerzburg,  
Am Hubland, Wuerzburg, Germany

<sup>2</sup> Comprehensive Heart Failure Center, University of Wuerzburg,  
Straubmuehlweg 2a, Wuerzburg, Germany

{a.wittek,toepfer,fette,pkluegl,puppe}@informatik.uni-wuerzburg.de

**Abstract.** This paper presents an extension of the UIMA Ruta Workbench for estimating the quality of arbitrary information extraction models on unseen documents. The user can specify expectations on the domain in the form of constraints, which are applied in order to predict the  $F_1$  score or the ranking. The applicability of the tool is illustrated in a case study for the segmentation of references, which also examines the robustness for different models and documents.

## 1 Introduction

Apache UIMA [5] and the surrounding ecosystem provide a powerful framework for engineering state-of-the-art Information Extraction (IE) systems, e.g., in the medical domain [13]. Two main approaches for building IE models can be distinguished. One approach is based on manually defining a set of rules, e.g., with UIMA Ruta<sup>3</sup> (Rule-based Text Annotation) [7]<sup>4</sup>, that is able to identify the interesting information or annotations of specific types. A knowledge engineer writes, extends, refines and tests the rules on a set of representative documents. The other approach relies on machine learning algorithms, such as probabilistic graphical models like Conditional Random Fields (CRF) [10]. Here, a set of annotated gold documents is used as a training set in order to estimate the parameters of the model. The resulting IE system of both approaches, the statistical model and the set of rules, is evaluated on an additional set of annotated documents in order to estimate its accuracy or  $F_1$  score, which is then assumed to hold for the application in general. However, while the system performed well in the evaluation setting, its accuracy decreases when applied on unseen documents, maybe because the set of documents applied for developing the IE system was not large or not representative enough. In order to estimate the actual performance, either more data is labeled or the results are manually checked by a human, who is able to validate the correctness of the annotations.

Annotated documents are essential for developing IE systems, but there is a natural lack of labeled data in most application domains and its creation is

<sup>3</sup> <http://uima.apache.org/ruta.html>

<sup>4</sup> previously published as TextMarker

error-prone, cumbersome and time-consuming as is the manual validation. An automatic estimation of the IE system’s quality on unseen documents would therefore provide many advantages. A human is able to validate the created annotations using background knowledge and expectations on the domain. This kind of knowledge is already used by current research in order to improve the IE models (c.f. [1, 6, 11]), but barely to estimate IE system’s quality.

This paper introduces an extension of the UIMA Ruta Workbench for exactly this use case: Estimating the quality and performance of arbitrary IE models on unseen documents. The user can specify expectations on the domain in the form of constraints thus the name Constraint-driven Evaluation (CDE). The constraints rate specific aspects of the labeled documents and are aggregated to a single CDE score, which provides a simple approximation of the evaluation measure, e.g., the token-based  $F_1$  score. The framework currently supports two different kinds of constraints: Simple UIMA Ruta rules, which express specific expectations concerning the relationship of annotations, and annotation-distribution constraints, which rate the coverage of features. We distinguish two tasks: predicting the actual  $F_1$  score of a document and estimating the ranking of the documents specified by the actual  $F_1$  score. The former task can give answers on how good the model performs. The latter task points to documents where the IE model can be improved. We evaluate the proposed tool in a case study for the segmentation of scientific references, which tries to estimate the  $F_1$  score of a rule-based system. The expectations are additionally applied on documents of a different distribution and on documents labeled by a different IE model. The results emphasize the advantages and usability of the approach, which works already with minimal efforts due to a simple fact: It is much easier to estimate how good a document is annotated than to actually identify the positions of defective or missing annotations.

The rest of the paper is structured as follows. In the upcoming section, we describe how our work relates to other fields of Information Extraction research. We explain the proposed CDE approach in Section 3. Section 4 covers the case study and the corresponding results. We conclude with pointers to future work in Section 5.

## 2 Related Work

Besides standard classification methods, which fit all model parameters against the labeled data of the supervised setting, there have been several efforts to incorporate background knowledge from either user expectations or external data analysis. Bellare et al. [1], Graça et al. [6] and Mann and McCallum [11], for example, showed how moments of auxiliary expectation functions on unlabeled data can be used for such a purpose with special objective functions and an alternating optimization procedure. Our work on constraint-driven evaluation is partly inspired by this idea, however, we address a different problem. We suggest to use auxiliary expectations to estimate the quality of classifiers on unseen data.



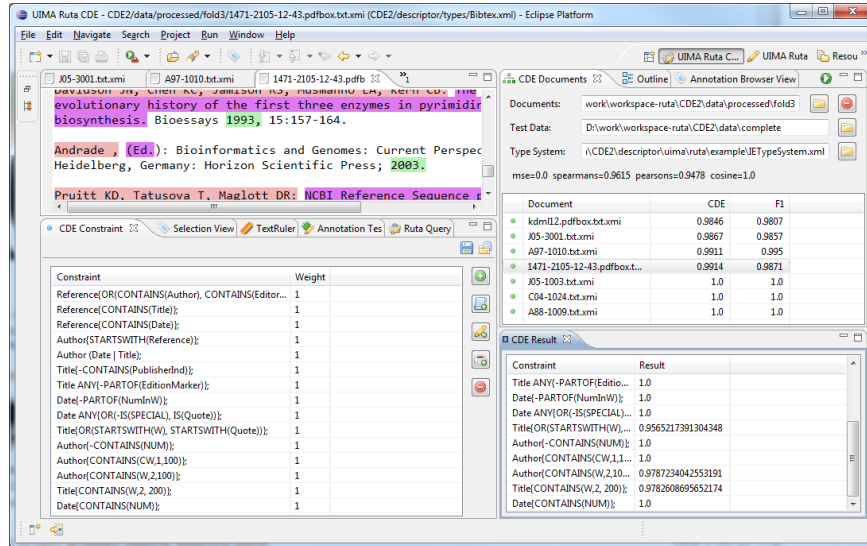
A classifier’s confidence describes the degree to which it believes that its own decisions are correct. Several classifiers provide intrinsic measures of confidence, for example, naive Bayes classifiers. Culotta and McCallum [4], for instance, studied confidence estimation for information extraction. They focus on predictions about field and record correctness of single instances. Their main motivation is to filter high precision results for database population. Similar to CDE, they use background knowledge features like record length, single field label assignments and field confidence values to estimate record confidence. CDE generalizes common confidence estimation because the goal of CDE is the estimation of the quality of arbitrary models.

Active learning algorithms are able to choose the order in which training examples are presented in order to improve learning, typically by selective sampling [2]. While the general CDE setting does not necessarily contain aspects of selective sampling, consider for example the batch  $F_1$  score prediction task, the ranking task can be used as a selective sampling strategy in applications to find instances that support system refactoring. The focus of the  $F_1$  ranking task, however, still differs from active learning goals which is essential for the design of such systems. Both approaches are supposed to favor different techniques to fit their different objectives. Popular active learning approaches such as density-weighting (e.g., [12]) focus on dense regions of the input distribution. CDE, however, tries to estimate the quality of the model on the whole data set and hence demands for differently designed methods. Despite their differences, the combination of active learning and CDE would be an interesting subject for future work. CDE may be used to find weak learners of ensembles and informative instances for these learners.

### 3 Constraint-driven Evaluation

The Constraint-driven Evaluation (CDE) framework presented in this work allows the user to specify expectations about the domain in form of constraints. These constraints are applied on documents with annotations, which have been created by an information extraction model. The results of the constraints are aggregated to a single CDE score, which reflects how well the annotations fulfill the user’s expectations and thus provide a predicted measurement of the model’s quality for these documents. The framework is implemented as an extension of the UIMA Ruta Workbench. Figure 1 provides a screenshot of the CDE perspective, which includes different views to formalize the set of constraints and to present the predicted quality of the model for the specified documents.

We define a constraint in this work as a function  $C : CAS \rightarrow [0, 1]$ , which returns a confidence value for an annotated document (CAS) where high values indicates that the expectations are fulfilled. Two different types of constraints are currently supported: RULE constraints are simple UIMA Ruta rules without actions and allow to specify sequential patterns or other relationships between annotations that need to be fulfilled. The result is basically the ratio of how often the rule has tried to match compared to how often the rule has actually



**Fig. 1.** CDE perspective in the UIMA Ruta Workbench. Bottom left: Expectations on the domain formalized as constraints. Top right: Set of documents and their CDE scores. Bottom right: Results of the constraints for the selected document.

matched. An example for such a constraint is  $\text{Document}\{\text{CONTAINS}(\text{Author})\}$ , which specifies that each document must contain an annotation of the type *Author*. The second type of supported constraints are Annotation Distribution (AD) constraints (c.f. Generalized Expectations [11]). Here, the expected distribution of an annotation or word is given for the evaluated types. The result of the constraint is the cosine similarity of the expected and the observed presence of the annotation or word within annotations of the given types. A constraint like "Peter": *Author* 0.9, *Title* 0.1, for example, indicates that the word "Peter" should rather be covered by an *Author* annotation than by a *Title* annotation. The set of constraints and their weights can be defined using the *CDE Constraint* view (c.f. Figure 1, bottom left).

For a given set of constraints  $C = \{C_1, C_2, \dots, C_n\}$  and corresponding weights  $w = \{w_1, w_2, \dots, w_n\}$ , the CDE score for each document is defined by the weighted average:

$$\text{CDE} = \frac{1}{n} \sum_i^n w_i \cdot C_i \quad (1)$$

The CDE scores for a set of documents may already be very useful as a report how well the annotations comply with the expectations on the domain. However, one can further distinguish two tasks for CDE: the prediction of the actual evaluation score of the model, e.g., the token-based  $F_1$  score, and the

prediction of the quality ranking of the documents. While the former task can give answers how good the model performs or whether the model is already good enough for the application, the latter task provides a useful tool for introspection: Which documents are poorly labeled by the model? Where should the model be improved? Are the expectations on the domain realistic? Due to the limited expressiveness of the aggregation function, we concentrate on the latter task. The CDE scores for the annotated documents are depicted in the *CDE Documents* view (c.f. Figure 1, top right). The result of each constraint for the currently selected document is given in the *CDE Results* view (c.f. Figure 1, bottom right).

The development of the constraints needs to be supported by tooling in order to achieve an improved prediction in the intended task. If the user extends or refines the expectations on the domain, then a feedback whether the prediction has improved or deteriorated is very valuable. For this purpose, the framework provides functionality to evaluate the prediction quality of the constraints itself. Given a set of documents with gold annotations, the CDE score of each document can be compared to the actual  $F_1$  score. Four measures are applied to evaluate the prediction quality of the constraints: the mean squared error, the Spearman’s rank correlation coefficient, the Pearson correlation coefficient and the cosine similarity. For optimizing the constraints to approximate the actual  $F_1$  score, the Pearson’s  $r$  is maximized, and for improving the predicted ranking, the Spearman’s  $\rho$  is maximized. If documents with gold annotations are available, then the  $F_1$  scores and the values of the four evaluation measures are given in the *CDE Documents* view (c.f. Figure 1, top right).

## 4 Case Study

The usability and advantages of the presented work are illustrated with a simple case study concerning the segmentation of scientific references, a popular domain for evaluating novel information extraction models. In this task, the information extraction model normally identifies about 12 different entities of the reference string, but in this case study we limited the relevant entities to *Author*, *Title* and *Date*, which are commonly applied in order to identify the cited publication.

In the main scenario of the case study, we try to estimate the extraction quality of a set of UIMA Ruta rules that shall identify the *Author*, *Title* and *Date* of a reference string. For this purpose, we define constraints representing the background knowledge about the domain for this specific set of rules. Additionally to this main setting of the case study, we also measure the prediction of the constraints in two different scenarios: In the first one, the documents have been labeled not by UIMA Ruta rules, but by a CRF model [10]. The CRF model was trained with a limited amount of iterations in a 5-fold manner. In a second scenario, we apply the UIMA Ruta rules on a set of documents of a different distribution including unknown style guides.

Table 1 provides an overview of the applied datasets. We make use of the references dataset of [9]. This data set is homogeneously divided in three sub-datasets with respect to their style guides and amount of references, which are

**Table 1.** Overview of utilized datasets.

$D_{ruta}$	219 references in 8 documents used to develop the set of UIMA Ruta rules.
$D_{dev}$	192 references in 8 documents labeled by the UIMA Ruta rules and applied for developing the constraints.
$D_{test}$	155 references in 7 documents labeled by the UIMA Ruta rules and applied to evaluate the constraints.
$D_{crf}$	$D_{ruta}$ , $D_{dev}$ and $D_{test}$ (566 references in 23 documents) labeled by a (5-fold) CRF model.
$D_{gen}$	452 references in 28 documents from a different source with unknown style guides labeled by the UIMA Ruta rules.

applied to develop the UIMA Ruta rules, define the set of constraints, and to evaluate the prediction of the constraints compared to the actual  $F_1$  score. The CRF model is trained on the partitions given in [9]. The last dataset  $D_{gen}$  consists of a mixture of the datasets CORA, CITESEERX and FLUX-CiM described in [3] generated by the rearrangement of [8].

**Table 2.** Overview of evaluated sets of constraints.

$C_{ruta}$	15 RULE constraints describing general expectations for the entities <i>Author</i> , <i>Title</i> and <i>Date</i> . The weight of each constraint is set to 1.
$C_{ruta+bib}$	$C_{ruta}$ extended with one additional AD constraint covering the entity-distribution of words extracted from Bibsonomy. The weight of each constraint is set to 1.
$C_{ruta+5xbib}$	Same set of constraints as in $C_{ruta+bib}$ , but the weight of the additional AD constraint is set to 5.

Table 2 provides an overview of the different sets of constraints, whose predictions are compared to the actual  $F_1$  score. First, we extended and refined a set of UIMA Ruta rules until they achieved an  $F_1$  score of 1.0 on the dataset  $D_{ruta}$ . Then, 15 RULE constraints  $C_{ruta}$ <sup>5</sup> have been specified using the dataset  $D_{dev}$ . The definition of the UIMA Ruta rules took about two hours and the definition of the constraints about one hour. Additionally to the RULE constraints, we created an AD constraint, which consists of the entity distribution of words that occurred at least 1000 times in the latest Bibtex database dump of Bibsonomy<sup>6</sup>. The set of constraints  $C_{ruta+bib}$  and  $C_{ruta+5xbib}$  combine both types of constraints with different weighting.

Table 3 contains the evaluation, which compares the predicted CDE score to the actual token-based  $F_1$  score for each document. We apply two different

<sup>5</sup> The actual implementation of the constraints as UIMA Ruta rules is depicted in Figure 1 (lower left part).

<sup>6</sup> <http://www.kde.cs.uni-kassel.de/bibsonomy/dumps>

**Table 3.** Spearman’s  $\rho$  and Pearson’s  $r$  given for the predicted CDE score (for each document) compared to the actual  $F_1$  score.

<i>Dataset</i>	<i>Cruta</i>		<i>Cruta+bib</i>		<i>Cruta+5xbib</i>	
	$\rho$	$r$	$\rho$	$r$	$\rho$	$r$
$D_{dev}$	0.8708	0.9306	0.9271	0.9405	0.8051	0.6646
$D_{test}$	0.9615	0.9478	0.9266	0.8754	0.8154	0.6758
$D_{crf}$	0.6793	0.7881	0.7429	0.8011	0.7117	0.7617
$D_{gen}$	0.7089	0.8002	0.7724	0.8811	0.8150	0.9504

correlation coefficients for measuring the quality of the prediction: Spearman’s  $\rho$  gives an indication about the ranking of the documents and Pearson’s  $r$  provides a general measure of linear dependency.

Although the expectations defined by the sets of constraints are limited and quite minimalistic covering mostly only common expectations, the results indicate that they can be useful in any scenario. The results for dataset  $D_{dev}$  are only given for completeness since this dataset was applied to define the set of constraints. The results for the dataset  $D_{test}$ , however, reflect the prediction on unseen documents of the same distribution. The ranking of the documents was almost perfectly estimated with a Spearman’s  $\rho$  of 0.9615<sup>7</sup>. The coefficients for the other scenarios  $D_{crf}$  and  $D_{gen}$  are considerably decreased, but the CDE scores are nevertheless very useful for an assessment of the extraction model’s quality. The five worst documents in  $D_{gen}$  (including new style guides), for example, have been reliably detected. The results show that the AD constraints can improve the prediction, but do not exploit their full potential in the current implementation. The impact measured for the dataset  $D_{crf}$  is not as distinctive since the CRF model already includes such features and thus is able to avoid errors that are detected by these constraints. However, the prediction in the dataset  $D_{gen}$  is considerably improved. The UIMA Ruta rules produce severe errors in documents with new style guides, which are easily detected by the word distribution.

## 5 Conclusions

This paper presented a tool for the UIMA community implemented in UIMA Ruta, which enables to estimate the extraction quality of arbitrary models on unseen documents. Its introspective report is able to improve the development of information extraction models already with minimal efforts. This is achieved by formalizing the background knowledge about the domain with different types of constraints. We have shown the usability and advantages of the approach in a case study about segmentation of references. Concerning future work, many prospects for improvements remain, for example a logistic regression model for

<sup>7</sup> The actual CDE and  $F_1$  scores of  $D_{test}$  are depicted in Figure 1 (right part)

approximating the scores of arbitrary evaluation measures, new types of constraints, or approaches to automatically acquire the expectations on a domain.

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

## References

1. Bellare, K., Druck, G., McCallum, A.: Alternating Projections for Learning with Expectation Constraints. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in AI. pp. 43–50. AUA Press (2009)
2. Cohn, D., Atlas, L., Ladner, R.: Improving generalization with active learning. *Machine Learning* 15, 201–221 (1994)
3. Councill, I., Giles, C.L., Kan, M.Y.: ParsCit: an Open-source CRF Reference String Parsing Package. In: Proceedings of the Sixth International Language Resources and Evaluation (LREC’08). ELRA, Marrakech, Morocco (2008)
4. Culotta, A., McCallum, A.: Confidence Estimation for Information Extraction. In: Proceedings of HLT-NAACL 2004: Short Papers. pp. 109–112. HLT-NAACL-Short ’04, Association for Computational Linguistics, Stroudsburg, PA, USA (2004)
5. Ferrucci, D., Lally, A.: UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering* 10(3/4), 327–348 (2004)
6. Graca, J., Ganchev, K., Taskar, B.: Expectation Maximization and Posterior Constraints. In: Platt, J., Koller, D., Singer, Y., Roweis, S. (eds.) NIPS 20, pp. 569–576. MIT Press, Cambridge, MA (2008)
7. Kluegl, P., Atzmueller, M., Puppe, F.: TextMarker: A Tool for Rule-Based Information Extraction. In: Chiarcos, C., de Castilho, R.E., Stede, M. (eds.) Proceedings of the 2nd UIMA@GSCL Workshop. pp. 233–240. Gunter Narr Verlag (2009)
8. Kluegl, P., Hotho, A., Puppe, F.: Local Adaptive Extraction of References. In: 33rd Annual German Conference on Artificial Intelligence (KI 2010). Springer (2010)
9. Kluegl, P., Toepfer, M., Lemmerich, F., Hotho, A., Puppe, F.: Collective Information Extraction with Context-Specific Consistencies. In: Flach, P.A., Bie, T.D., Cristianini, N. (eds.) ECML/PKDD (1). Lecture Notes in Computer Science, vol. 7523, pp. 728–743. Springer (2012)
10. Lafferty, J., McCallum, A., Pereira, F.: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proc. 18th International Conf. on Machine Learning* pp. 282–289 (2001)
11. Mann, G.S., McCallum, A.: Generalized Expectation Criteria for Semi-Supervised Learning with Weakly Labeled Data. *J. Mach. Learn. Res.* 11, 955–984 (2010)
12. McCallum, A., Nigam, K.: Employing EM and Pool-Based Active Learning for Text Classification. In: Shavlik, J.W. (ed.) ICML. pp. 350–358. Morgan Kaufmann (1998)
13. Savova, G.K., Masanz, J.J., Ogren, P.V., Zheng, J., Sohn, S., Kipper-Schuler, K.C., Chute, C.G.: Mayo clinical Text Analysis and Knowledge Extraction System (cTAKES): architecture, component evaluation and applications. *Journal of the American Medical Informatics Association : JAMIA* 17(5), 507–513 (Sep 2010)

## Author Index

Chappelier, Jean-Cedric	34
Chen, Pei	1
Codina, Joan	42
Di Bari, Alessandro	2
Fang, Yan	14
Faraotti, Alessandro	2
Fette, Georg	10, 58
Gambardella, Carmela	2
García Narbona, David	42
Garduno, Elmer	14
Grivolla, Jens	42
Hernandez, Nicolas	18
Kluegl, Peter	58
Maiberg, Avner	14
Massó Sanabre, Guillem	42
McCormack, Collin	14
Noh, Tae-Gil	26
Nyberg, Eric	14
Padó, Sebastian	26
Puppe, Frank	10, 58
Richardet, Renaud	34
Rodríguez-Penagos, Carlos	42
Savova, Guergana	1
Stadermann, Jan	50
Symons, Stephan	50
Telefont, Martin	34
Thon, Ingo	50
Toepfer, Martin	10, 58
Vetere, Guido	2
Wittek, Andreas	58
Yang, Zi	14