An Overview of JCoRe, the JULIE Lab UIMA Component Repository

U. Hahn, E. Buyko, R. Landefeld, M. Mühlhausen, M. Poprat, K. Tomanek, J. Wermter

Jena University Language & Information Engineering (JULIE) Lab
Friedrich-Schiller-Universität Jena
Fürstengraben 30, D-07743 Jena, Germany
{hahn|buyko|landefeld|muehlhausen|poprat|tomanek|wermter}@coling-uni-jena.de

Abstract

We introduce JCoRe, a full-fledged UIMA-compliant component repository for complex text analytics developed at the Jena University Language & Information Engineering (JULIE) Lab. JCoRe is based on a comprehensive type system and a variety of document readers, analysis engines, and CAS consumers. We survey these components and then turn to a discussion of lessons we learnt, with particular emphasis on managing the underlying type system. We briefly sketch two complex NLP applications which can easily be built from the components contained in JCoRe.

1. Introduction

During the past years, we have witnessed an unmatched growth of language processing modules such as tokenizers, stemmers, chunkers, parsers, etc. This software was usually created in a stand-alone manner, locally at the implementator's lab, and sometimes made publicly available on the programmer's personal or institutional web pages. In the last couple of years, several repositories have been set up, including, e.g., those of the Linguistic Data Consortium,[1] the Open Language Archives Community,[2] the European Language Resources Association,[3] and the Natural Language Software Registry[4]. As a common feature, these repositories just posted software modules but offered no additional service besides making available the plain resources (i.e., code, with – often fairly limited or even no – documentation). Hence, reusability was hampered by various different data exchange formats, let aside dependencies of different programming languages and operating systems. Any attempt to reuse this software or even create composite NLP systems from modules selected from these repositories created a heavy burden for system developers to achieve at least a decent level of interoperability. Under these conditions, although substantial collections of code were available, the compilation of NLP pipelines based on such components was quite inefficient and time-consuming.

Those Human Language Technologists already involved in complex system building activities, at that time, rendered rather monolithic and hard-shell pipelines that often resisted flexible exchange of single, externally developed components and their easy adaptation. Modification of these systems' architecture and basic functionality often required a major re-design, and, hence, re-programming directly at the code level.

With the advent of NLP framework architectures this impediment started to be resolved at the design level. GATE (Cunningham, 2002) and ATLAS (Laprun et al., 2002) were among the first of those systems that abstracted away from nitty-gritty programming details and moved system architectures to the level of data abstraction. UIMA, the *Unstructured Information Management Architecture*, provided additional abstraction layers, most notably by explicitly requiring a type system which described the underlying data structures to be specified (Ferrucci and Lally, 2004; Götz and Suhre, 2004).

Recently, second generation NLP repositories have been set up such as the one located at Carnegie Mellon University[5] or the JULIE Component Repository[6] (JCoRe), which we will describe in more depth in the remainder of this paper. JCoRe offers a large variety of NLP components for diverse NLP tasks which may range from sentence splitting, tokenization, via chunking and parsing, to named entity recognition and relation extraction. At the heart of JCoRe lies a comprehensive common type system for text analytics. Thus, the components in this repository can easily and flexibly be assembled into a variety of NLP applications without the need of any format conversion and re-programming.

After a brief introduction to UIMA in Section 2., in Section 3., we will describe JCoRe, the JULIE Component Repository, including the type system and the different kinds of readers, analysis engines, and consumers we currently supply. After that, in Section 4., we will discuss our experience with management issues related to the type system, in particular, dealing with type incompatibility and type system modifications.

2. UIMA In Brief

UIMA is a software framework and a platform for unstructured information management solutions. While originally developed by IBM, UIMA is now an Apache-licensed open source project. In the following we will shortly describe the basic concepts of UIMA. For more detailed and technical information we refer the reader to the Apache UIMA documentation.[7]

[1] http://www.ldc.upenn.edu
[2] http://linguistlist.org/olac
[3] http://www.elra.info
[4] http://registry.dfki.de
[5] http://uima.lti.cs.cmu.edu
[6] http://www.julielab.de
[7] http://incubator.apache.org/uima/documentation.html

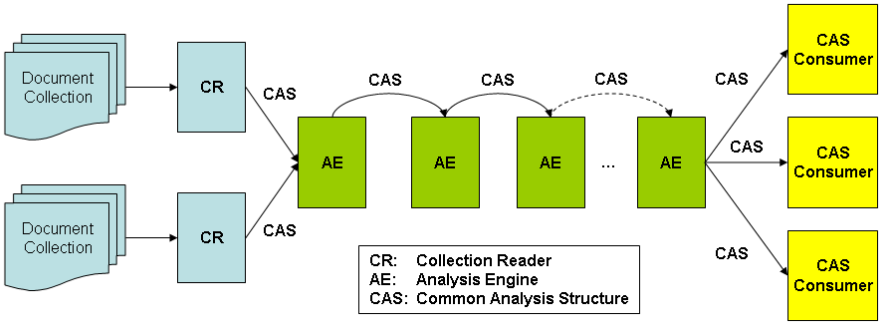

Figure 1: A schematic representation of an NLP pipeline to process unstructured data with UIMA components (Collection Readers (CR), Analysis Engines (AE), and Common Analysis Structure (CAS) Consumers). Information that is handed over and enriched or modified from AE to AE is managed within CAS objects that are based on a type system as their data model.

UIMA is a data-driven architecture which means that single components communicate with each other by exchanging (annotated) data. The integration of components in the UIMA framework thus requires clear interface definitions with respect to the input and the output data. The *Common Analysis Structure* (CAS) is UIMA's underlying object-oriented data structure (Götz and Suhre, 2004). The CAS represents one single item of unstructured data (e.g. a single document) and consists of one or more *Sofas* (subject of analysis, a view of the data item) with the meta data added by the single UIMA components. UIMA meta data objects are so called feature structures which are instances of UIMA *types*. These types can be arranged in an inheritance hierarchy and thus constitute a *type system* very similar to a class model in the object-oriented programming paradigm. The type system concept is UIMA's key feature to add structure to an unstructured chunk of data. For text processing purposes, UIMA comes with a predefined basic type, the annotation type. This annotation type and all its sub-types determine the particular annotation schema.

In UIMA, three different types of components are distinguished in the processing cycle (see also Figure 1): *Collection Readers* (CR) have different kinds of (typically unstructured, i.e., textual, audio or video) data as their input, textual documents in our case. CRs read this data from the selected source (files, database, etc.) and make it accessible for further processing steps. The linguistic processing proper of the documents is carried out by *Analysis Engines* (AE), each of which adds annotations according to different levels of analysis (e.g., POS tags, named entities, etc.). Finally, these annotation-enriched documents can be handed over to *CAS Consumers* (CC), which realize different functional requirements such as data conversion to specific output formats, search engine index construction, database feed, Web viewers, etc.

Several components, CRs, AEs, and CCs, can then be assembled into *pipelines* to build specific UIMA-based applications. Although UIMA is designed to be used for any kind of data, we here consider only textual data as our subject of analysis.

3. JCoRe — Julie Component Repository

JCoRe, the Julie Component Repository, provides all components to configure NLP analysis pipelines based on the UIMA framework presented in the previous section. A comprehensive annotation type definition is provided as the backbone of the whole system which allows flexible data exchange between all components involved. Several collection readers enable users to access markup and annotations from other projects within the UIMA framework. A continuously growing collection of text analytics components incorporates low-level NLP tasks such as sentence segmentation as well as high-end functionality in terms of relation extraction. Finally, several consumers are supplied to deploy or export the annotations.

All components of the JCoRe are written in Java. The components are available for download from `http://www.julielab.de/` as PEAR packages[8] and contain compiled classes, the source code, and an example model for components based on machine learning techniques.

Table 1 gives an overview of the components currently contained in JCoRe. In the remainder of this section we briefly describe the single components. For a more detailed explanation of any component, we refer the reader to the documentation contained in the PEAR packages and the cited publications.

3.1. Annotation Language: JCoRe's Type System

The data structure backbone of our component repository is a comprehensive annotation type system (Hahn et al., 2007; Buyko and Hahn, 2008), which covers major steps of NLP processing. It consists of several specification layers which provide (mostly) genre-, language-, and domain-independent definitions of the respective annotation types. When applied in specific scenarios, these generic annotation types might be extended by application-specific ones.

The *Document Meta* layer comprises annotation types for bibliographical and content information about a document – such as author, title, or year of publication. There is an extension to this layer for the biomedical domain allowing

[8] PEAR (Processing Engine ARchive) is a UIMA standard for packaging and automatically deploying components.

Component	Type	Comment	Source/Reference
JULIE Type System	TS	–	see Hahn et al. (2007), Buyko and Hahn (2008)
MEDLINE Reader	CR	–	–
ACE Reader	CR	–	–
MUC7 Reader	CR	–	–
JULIE Sentence Segmenter	AE	ML-based, self-developed	see Tomanek et al. (2007)
JULIE Token Segmenter	AE	ML-based, self-developed	see Tomanek et al. (2007)
Simple Sentence Segmenter	AE	rule-based, wrapper for JTokenizer	`http://www.andy-roberts.net/software/`
Simple Tokenizer	AE	rule-based, wrapper for JTokenizer	`http://www.andy-roberts.net/software/`
OPENNLP Sentence Segmenter	AE	ML-based, wrapper	`http://opennlp.sourceforge.net/`
OPENNLP Token Segmenter	AE	ML-based, wrapper	`http://opennlp.sourceforge.net/`
Stemmer	AE	rule-based, wrapper for Porter stemmer	`http://snowball.tartarus.org/`
OPENNLP POS Tagger	AE	ML-based, wrapper	`http://opennlp.sourceforge.net/`
OPENNLP Chunker	AE	ML-based, wrapper	`http://opennlp.sourceforge.net/`
OPENNLP Constituency Parser	AE	ML-based, wrapper	`http://opennlp.sourceforge.net/`
MST Dependency Parser	AE	ML-based, wrapped/modified	see McDonald et al. (2005)
Acronym Resolution	AE	rule-based, reimplementation	see Schwartz and Hearst (2003)
JULIE Named Entity Tagger	AE	ML-based, self-developed	–
Gazetteer	AE	dictionary, wrapper for Lingpipe's list look-up tool	`http://alias-i.com/lingpipe/`
JULIE Coordination Resolution	AE	rule-/ML-based, self-developed	see Buyko et al. (2007)
Relation Extractor	AE	ML-based, self-developed	–
Lucene Indexer	CC	–	–
CAS2DB Consumer	CC	–	–
CAS2IOB Consumer	CC	–	–

Table 1: Overview of Components in JCORE, the JULIE Component Repository

to store the meta information exclusively provided for documents when retrieved from PUBMED,[9] such as MESH[10] terms, chemicals, and genes referred to in a document. To incorporate information about the document structure, such as formal zones typically used in scientific texts (e.g., sections and paragraphs), the *Document Structure & Style* layer offers dedicated types. The types from the *Morpho-Syntax & Syntax* layer refer to linguistic annotations ranging from sentence up to parse annotations. Finally, the *Semantics* layer offers types for semantic annotations, including entities, relations, and events.

3.2. Preprocessing: Collection Readers

Currently, we provide three different exemplars of collection readers (see Table 1). Two of them import semantically annotated newswire corpora, *viz.* the ACE 2005 (Doddington et al., 2004) and the MUC-7 (Hirschman and Chinchor, 1998) corpora, and convert the given annotations to the CAS representation. From the annotated ACE corpus, the ACE *Reader* extracts named entities (persons, organizations, values, etc.), coreferences, relations, and events. From the MUC-7 data set, the MUC*7 Reader* extracts named entities and coreferences (event annotations are intentionally ignored). For both corpora, our type system has been extended with the respective types (Buyko and Hahn, 2008). Once, external annotations are read into the UIMA framework, this allows for further processing of the documents making immediate use of this annotated data. Moreover, such annotations might serve as input material for training and testing NLP components, such as named entities recognizers, coreference resolvers, and relation extractors.

The MEDLINE *Reader* parses MEDLINE records that come in an XML encoded format. They not only contain the plain text but also various meta data such as information about the authors and their affiliations, the publication date, information about the journal the article appeared in, manually assigned descriptors (mainly MESH terms), etc. In summary, all our readers extract the originally encoded (meta) data and map this information to the types and features of JCORE's type system.

3.3. Text Processing: Analysis Engines

JCORE contains text analytics components for different processing levels including linguistic preprocessing and semantic processing up to relation extraction at the time of this writing.

Depending on the task to be served, NLP component developers may either choose rule-based approaches or make use of machine learning (ML) methods (Hahn and Wermter, 2006). While, e.g., for the recognition of city names a simple gazetteer look-up might be sufficient, entity recognition in the biomedical domain is usually better performed by ML-based approaches due to complex and inconsistent naming conventions, ambiguities, etc. (Park and Kim, 2006).

JCORE contains both rule-based and ML-based components for language processing. For the ML-based compo-

[9] PUBMED (`http://www.ncbi.nlm.nih.gov/`) is a bibliographical database which includes over 17 million citations from MEDLINE and other life science journals for biomedical articles.

[10] Medical Subject Headings (MESH, `http://www.nlm.nih.gov/mesh`) is a high-coverage controlled biomedical terminology.

nents, we also provide some pre-trained models for download in case training material was freely available. Of course, these components can be retrained for usage in different domains or with different semantic types given the respective training material.

While some of our text processing components are entirely *self-developed*, others are based on already existing third party libraries or tools for which we wrote *wrappers* so that they could be used as a component inside the UIMA framework. Mostly, wrapping only meant to call the respective methods from within the analysis engine class and to convert and write the tool's output to the CAS. In some cases, however, wrapping required some *modifications* of the original tool to let it fit into the UIMA framework.

Linguistic Processing JCORE contains three components for both sentence and token segmentation. First, there are rule-based components which provide an UIMA wrapper for the JTokenizer,[11] a third party package mainly based on regular expression segmentation. The segmentation rules for these components can be flexibly defined by the user. Second, there are UIMA wrappers of the segmenter tools from the OPENNLP tool suite.[12] These are based on Maximum Entropy (ME) models (Berger et al., 1996). To address special intricacies of scientific subdomains such as biomedicine we have developed our own segmentation tools (Tomanek et al., 2007) based on Conditional Random Fields (CRF) (Lafferty et al., 2001) and a rich set of features.

To handle morphological variation of words (deletion of inflection suffixes, in particular) we have created a wrapper for the Java version of the SNOWBALL stemmers,[13] including the original Porter stemmer for English and additional versions for many other languages.

For syntactic analysis, we provide UIMA wrappers for the POS tagger, the phrase chunker, and the constituency-based parser from the OPENNLP tool suite. These are also based on ME models and have proven to work well on scientific documents when retrained on appropriate training data (Buyko et al., 2006). Further, we have integrated the MSTPARSER (McDonald et al., 2005), a parser for non-projective dependency structures, also based on ML methods. Writing an UIMA wrapper here also meant to slightly modify the MSTPARSER's source code so that the model needs to be loaded only once during the initialization phase.

Semantic Processing For acronym resolution, we reimplemented a simple, but well-performing algorithm originally presented by Schwartz and Hearst (2003): For each locally introduced acronym (some upper-case letters in brackets, such as "WHO"), the full form is searched to the left of this acronym until each letter from the acronym is found in the proper order of appearance. All occurrences of an acronym in a document are annotated with the identified full form.

Our repository comprises two tools for named entity recognition. One is based on Lingpipe's[14] list look-up tool. Provided with a list of names, these are searched for in the document. Both exact and approximate matching (based on weighted edit distance) are possible. Second, we have developed an entity tagger based on CRFs, which is similar in spirit to the one proposed by Settles (2004). Given appropriate training material, our ML-based entity tagger can be used for arbitrary domains and entity classes. It comprises a rich set of features which can be configured according to the respective scenario. Further, it allows for acronyms being expanded to their full forms during tagging (given they were marked before as such) to avoid erroneous tagging especially of ambiguous acronyms.

The repository also contains a component to resolve elliptical entity mentions in coordinations, such as normalizing *"Mr. and Mrs. Miller"* to *"Mr. Miller"* and *"Mrs. Miller"* (Buyko et al., 2007). Our coordination resolver can be configured either for the use of a set of rules considering POS information only, or for the use of an ML model with a variety of lexical, morpho-syntactic and even semantic features.

Finally, there is a component for relation extraction based on supervised ML. Here, an ME classifier is applied to determine for any ordered pair of two entities in a document whether these are in a specific relation. Relation extraction is currently the top level analysis component as it is based on the analysis results of many other components including, e.g., POS tagging, parsing, and entity recognition.

3.4. Postprocessing: CAS Consumers

The processing results of our text analysis components can be deployed by CAS Consumers. These components constitute an interface to arbitrary applications which use the UIMA annotations. A consumer that is a default part of UIMA allows to store the UIMA analysis results in the XMI (XML Metadata Interchange) format which is an OMG[15] standard for exchanging meta data based on XML (Extensible Markup Language). In many scenarios, however, consumers tailored to the specific needs of an application will be required. We here present three consumers which were created in the context of different NLP applications and information extraction research projects.

CAS2IOB Consumer The IOB format (inside, outside, begin) is a common exchange format for segmentation-based, non-nesting annotations (e.g., chunking (Ramshaw and Marcus, 1995)). Many publicly available corpora are annotated in this format (e.g., for the CONLL 2003 (Tjong Kim Sang and De Meulder, 2003) or the CONLL 2004 (Carreras and Màrquez, 2004) shared tasks). Furthermore, training, testing, and evaluation software that is based on this format has been developed for several competitions (e.g., CoNLL) and is widely accepted. The *CAS2IOB Consumer* extracts specified annotations from the CAS to the IOB format, following simple heuristics to resolve nested and overlapping annotations (e.g., preference for the longest annotation).

Lucene Indexer Apache Lucene[16] is an open source text retrieval software which can efficiently manage millions of

[11] http://www.andy-roberts.net/software/
[12] http://opennlp.sourceforge.net/
[13] http://snowball.tartarus.org/
[14] http://alias-i.com/lingpipe/
[15] http://www.omg.org/
[16] http://lucene.apache.org

documents. Our Lucene Indexer automatically creates a Lucene search engine index by mapping the CAS annotations to particular Lucene index fields. This allows us to retrieve documents not only by the information contained directly in the (unstructured) text itself but also by their meta data that is given by the documents' provider (e.g., author names, publication data, etc.) as well as the annotations added by the UIMA components. It can flexibly be used for any kinds of annotations. Mapping rules define the assignment of annotation types or their attribute to the particular fields in the Lucene index.

CAS2DB Consumer Whereas the Lucene Indexer makes feasible efficient search within the annotated documents, for further processing (such as displaying retrieved documents with all their annotations) the annotated documents must be stored in a way which allows fast access. This can be accomplished by our *CAS2DB Consumer*, which feeds the annotations derived from and assigned to the documents into a relational database. The CAS2DB Consumer is based on an abstract database schema which is independent of any particular annotation type system and thus allows flexible reuse. Once stored in the database, the data can be optimized and re-arranged according to the particular application's needs. Currently the database schema and the CAS2DB Consumer are implemented for the PostgreSQL[17] database management system, but it can easily be adapted to any other relational database system.

4. Lessons Learnt: Type System Management

Most difficulties we faced when extending our component repository and using the UIMA framework in our daily work are related to the management of the type system. In the following we discuss two of these problems together with a possible solution.

4.1. Type System Incompatibility

Although UIMA, in theory, allows for easy interoperability between UIMA components, this idea is only realized directly if all components are based on the same type system. Since, however, the NLP community has not yet agreed on a common NLP type system for UIMA, there are several home-grown, possibly very specific type systems in use for different components resulting in impaired interoperability. Assume the following example: Given a component A from the JCORE repository, e.g., a sentence splitter which writes its analysis output to the annotation type *de.julielab.types.sentence*, and a third party component B, e.g., a tokenizer which assumes as input sentence annotations stored in the annotation type *org.mylab.types.sentence*. Without further synchronization these two components cannot be linked in the same pipeline.

Now, we could of course modify component B to work on the sentence annotations of component A (or vice versa), given the source code were available. Yet, to avoid such source code modification, the following workaround seems helpful:[18] Write a small preprocessing AE which copies the relevant annotations from component A to the types component B expects. In a postprocessing AE, annotations created by component B will be copied and transferred to annotation types expected by the other components of a pipeline.

4.2. Type System Modification

Another problematic issue in the UIMA framework pops up when a type system used by several components of a repository is changed. Monotonic *extensions* of a type system, i.e., adding new types or extending the attributes of already existing types, is not really problematic. However, *modifications* of whole types or attributes (even if only names of types or attributes are changed), might lead to severe conflicts.

Why do type systems change? Although we have attempted to design a comprehensive and linguistically motivated type system before we started to implement the UIMA components, we continuously face the following two reasons for ubiquitous change:

- At the "borders" of a type system new subtypes are constantly required due to specific application scenarios. For example, for the semantics layer this could mean that we need special types inheriting from the general entity mention annotation type.
- But also the "core" of our type system is not immune to changes. This is mostly due to new findings regarding the design of specific annotation types.

The first issue can easily be solved by designing the type system as a domain- and application-independent core where application-specific types should not be integrated. Rather, specific requirements would be integrated into application-dependent extensions.

We have further split the "core" type system into several logical partitions, i.e., the layers addressed in Section 3.1. Each such layer is realized by a separate UIMA type system descriptor, possibly including other layers in case of dependencies between UIMA types. The availability of single logical units also adds to the clarity because our type system altogether contains several dozens of annotation types, partly arranged in a multi-level type hierarchy. Application-specific extensions can then be realized for the respective unit.

The second issue, i.e., modifications in the core type system, is more serious since components based on different versions of the type system might probably not be integrable into one pipeline due to conflicting annotation types. As we have organized all of our UIMA components as separate Java projects, managed by the build management tool Maven,[19] the modification of the type system currently implies lot of manual work because we need to go through all of these projects, exchange the type system (or at least the

[17] `http://www.postgresql.org`

[18] Thanks to Olivier Terrier from Temis for fruitful discussions and the idea for this solution.

[19] `http://maven.apache.org/`

respective layer if it has changed), see whether the component's source code needs to be updated with respect to the type modifications, and finally deploy the updated versions. This problem could be solved semi-automatically with the following labor-saving workflow. Once we have changed the type system in a way that it could impair the functionality of one of our UIMA components, and provided that for each component there is a (reasonable) functionality test (such as a JUnit[20] test), then, in a first step, all type system descriptors in every component will be replaced by the modified version and the Java classes of type system will be updated automatically. In a next step, the functionality test of each component will be executed. If the test runs successfully, the component will be marked as successfully type-system-updated. Otherwise, the developer in charge has to modify this particular component with respect to the new type system. Finally, only if the functionality tests of all components are passed without functional deviations, all components can be updated in a version control system (e.g., SVN[21]), be deployed as a Java project, and PEARs can be build in order to exchange the components easily. In particular by the last constraint which accepts only updated and fully functional components as part of the component repository, we can compile pipelines in a convenient way without running into errors caused by inconsistent type system versions.

The workflow proposed here has not been tested yet, but we are about to implement it as a repository management procedure in our lab.

5. Conclusion

We gave an overview of the JULIE Component Repository (JCORE). This work is motivated by our goal to develop complex NLP software in a disciplined and flexible way. We have implemented, up until now, two particular application systems both of which are fully set up by components from our UIMA component repository.

The STEMNET project[22] aims at building a semantic search engine for the biomedical subdomain of immunology. For this scenario, we set up an NLP pipeline which reads MEDLINE abstracts using the MEDLINE Reader and then processes these documents by means of linguistic preprocessing (sentence and token segmentation, POS tagging) and recognition of various entity types. The data sink here is the Lucene Indexer which stores the annotations in a search engine index which can then be queried by users searching for relevant scientific documents.

Our second application accounts for the automatic synthesis of a biomedical fact database as done in the BOOTSTREP project.[23] To identify interactions between proteins described in scientific documents, we set up a pipeline passing through all levels of linguistic and semantic processing, including especially relation extraction. Finally, the CAS2DB Consumer is used to store the identified relations as facts in a database.

[20] `http://junit.org/`

[21] `http://subversion.tigris.org/`

[22] `http://www.stemnet.de`

[23] `http://www.bootstrep.eu`

Besides these lab-internal uses, our download statistics indicate that JCORE resources are of high interest to and used by many visitors of our web site. The Open Source policy we support allows external users to integrate our work in a flexible way in their application frameworks.

This is certainly an advantage over Web services often considered as an alternative. Relying on Web services, users have to turn to the developers of the code and negotiate with them changes they are after. With Open Source material they can do it on their own. Also Web services have certainly performance deficits when large amounts of data have to be shuffled across the WWW. Web services might be useful for testing but they might not really be competitive for large-scale production systems.

Finally, we hope that JCORE resources, the type system in particular, might stimulate discussions about emerging standards for NLP. It offers an appropriate level of abstraction to talk about the essential parameters of our research and development work.

Acknowledgements

This research was funded by the EC within the BOOTSTREP project (FP6-028099), and by the German Ministry of Education and Research within the STEMNET project (01DS001A-C). The first author is recipient of the *UIMA Innovation Award 2007* and holds an award grant from IBM.

6. References

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Ekaterina Buyko and Udo Hahn. 2008. Fully embedded type systems for the semantic annotation layer. In *ICGL 2008 – Proceedings of the 1st International Conference on Global Interoperability for Language Resources*, pages 26–33. Hong Kong, SAR, January 9-11, 2008. City University of Hong Kong.

Ekaterina Buyko, Joachim Wermter, Michael Poprat, and Udo Hahn. 2006. Automatically adapting an NLP core engine to the biology domain. In Hagit Shatkay, Lynette Hirschman, Alfonso Valencia, and Christian Blaschke, editors, *Proceedings of the Joint BioLINK-Bio-Ontologies Meeting. A Joint Meeting of the ISMB Special Interest Group on Bio-Ontologies and the BioLINK Special Interest Group on Text Data Mining in Association with ISMB*, pages 65–68. Fortaleza, Brazil, August 5, 2006.

Ekaterina Buyko, Katrin Tomanek, and Udo Hahn. 2007. Resolution of coordination ellipses in biological named entities using conditional random fields. In *PACLING 2007 - Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics*, pages 163–171. Melbourne, Australia, September 19-21, 2007. Melbourne: Pacific Association for Computational Linguistics.

Xavier Carreras and Luís Màrquez. 2004. Introduction to the CONLL-2004 shared task: Semantic role labeling. In Hwee Tou Ng and Ellen Riloff, editors, *CoNLL-2004*

– *Proceedings of the 8th Conference on Computational Natural Language Learning at HLT-NAACL 2004*, pages 89–97. Boston, MA, USA, 2004. Association for Computational Linguistics.

Hamish Cunningham. 2002. GATE, a general architecture for text engineering. *Computers and the Humanities*, 36:223–254.

George Doddington, Alexis Mitchell, Mark Przybocki, Lance Ramshaw, Stephanie Strassel, and Ralph Weischedel. 2004. The Automatic Content Extraction (ACE) Program: Tasks, data, & evaluation. In *LREC 2004 – Proceedings of the 4th International Conference on Language Resources and Evaluation. In Memory of Antonio Zampolli. Vol. 3*, pages 837–840. Lisbon, Portugal, 26-28 May 2004. Paris: European Language Resources Association (ELRA).

David Ferrucci and Adam Lally. 2004. UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348.

Thilo Götz and Oliver Suhre. 2004. Design and implementation of the UIMA Common Analysis System. *IBM Systems Journal*, 43(3):476–489.

Udo Hahn and Joachim Wermter. 2006. Levels of natural language processing for text mining. In Sophia Ananiadou and John McNaught, editors, *Text Mining for Biology and Biomedicine*, pages 13–41. Norwood, MA: Artech House.

Udo Hahn, Ekaterina Buyko, Katrin Tomanek, Scott Piao, John McNaught, Yoshimasa Tsuruoka, and Sophia Ananiadou. 2007. An annotation type system for a data-driven NLP pipeline. In *The LAW at ACL 2007 – Proceedings of the Linguistic Annotation Workshop*, pages 33–40. Prague, Czech Republic, June 28-29, 2007. Stroudsburg, PA: Association for Computational Linguistics.

Lynette Hirschman and Nancy Chinchor. 1998. Muc-7 coreference task definition (version 3.0). In *Proceedings of the MUC-7, Message Understanding Conference*, `http://www.itl.nist.gov/iad/894.02/related_projects/muc/proceedings/co_task.html`.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML-2001 – Proceedings of the 18th International Conference on Machine Learning*, pages 282–289. Williams College, MA, USA, June 28 - July 1, 2001. San Francisco, CA: Morgan Kaufmann.

Christophe Laprun, Jonathan G. Fiscus, John Garofolo, and Sylvain Pajot. 2002. A practical introduction to ATLAS. In M.G. Rodriguez and C. Paz Suarez Araujo, editors, *LREC 2002 – Proceedings of the 3rd International Conference on Language Resources and Evaluation*, pages 1928–1932. Las Palmas de Gran Canaria, Spain, 29-31 May, 2002. Paris: European Language Resources Association (ELRA).

Ryan McDonald, Fernando Pereira, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *HLT-EMNLP'05 – Proceedings of the 5th Human Language Technology Conference and 2005 Conference on Empirical Methods in Natural Language Processing*, pages 523–530. Vancouver, B.C., Canada, October 6-8, 2005. East Stroudsburg, PA: Association for Computational Linguistics.

Jong C. Park and Jung-Jae Kim. 2006. Named entity recognition. In Sophia Ananiadou and John McNaught, editors, *Text Mining for Biology and Biomedicine*, pages 121–142. Norwood, MA: Artech House.

Lance Ramshaw and Mitchell P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the 3rd ACL Workshop on Very Large Corpora*, pages 82–94. Cambridge, MA, USA, June 30, 1995. Association for Computational Linguistics.

Ariel S. Schwartz and Marti A. Hearst. 2003. A simple algorithm for identifying abbreviation definitions in biomedical text. In Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Tiffany A. Jung, and Teri E. Klein, editors, *PSB 2003 – Proceedings of the Pacific Symposium on Biocomputing 2003*, pages 451–462. Kauai, Hawaii, USA, January 3-7, 2003. Singapore: World Scientific Publishing.

Burr Settles. 2004. Biomedical named entity recognition using conditional random fields and rich feature sets. In Nigel Collier, Patrick Ruch, and Adeline Nazarenko, editors, *JNLPBA – Proceedings of the COLING 2004 International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, pages 107–110. Geneva, Switzerland, August 28-29, 2004.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CONLL-2003 shared task: Language-independent named entity recognition. In Walter Daelemans and Miles Osborne, editors, *CoNLL-2003 – Proceedings of the 7th Conference on Computational Natural Language Learning*, pages 142–147. Edmonton, Canada, 2003. Association for Computational Linguistics.

Katrin Tomanek, Joachim Wermter, and Udo Hahn. 2007. Sentence and token splitting based on conditional random fields. In *PACLING 2007 - Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics*, pages 49–57. Melbourne, Australia, September 19-21, 2007. Melbourne: Pacific Association for Computational Linguistics.