A UIMA Database Interface for Managing NLP-related Text Annotations

Giuseppe Abrami and Alexander Mehler
Text-technology Lab, Goethe University Frankfurt
Robert-Mayer-Straße 10, 60325 Frankfurt am Main
{abrami, mehler}@em.uni-frankfurt.de

Abstract

NLP and automatic text analysis necessarily involve the annotation of natural language texts. The Apache *Unstructured Information Management applications* (UIMA) framework is used in several projects, tools and resources, and has become a *de facto* standard in this area. Despite the multiple use of UIMA as a document-based schema, it does not provide native database support. In order to facilitate distributed storage and enable UIMA-based projects to perform targeted queries, we have developed the *UIMA Database Interface* (UIMA DI). UIMA DI sets up an environment for a generic use of UIMA documents in database systems. In addition, the integration of UIMA DI into rights and resource management tools enables user and group-specific access to UIMA documents and provides data protection. Finally, UIMA documents can be made accessible for third party programs. UIMA DI, which we evaluate in relation to file system-based storage, is available under the GPLv3 license via GitHub.

Keywords: UIMA, database interface, text annotation, Neo4J, MongoDB, Web Services, REST

1. Introduction

NLP and automatic text analysis necessarily involve the annotation of natural language texts. In various projects (e.g. (Da Silva et al., 2006; Kano et al., 2009; Hemati et al., 2016)), especially in an NLP context (e.g. (Savova et al., 2010; Eckart de Castilho and Gurevych, 2014; Patterson et al., 2017; Kreimeyer et al., 2017; Rudzewitz et al., 2017; Kassner et al., 2017)), the Apache *Unstructured Information Management applications* (UIMA) (Ferrucci et al., 2009) is used as a standard architecture for text annotation. Accordingly, there are various tools for processing and producing UIMA compliant documents (e.g. (Ogren et al., 2008; Zeldes et al., 2009; Ferrucci et al., 2010; Hemati et al., 2016; Niekler et al., 2017)). UIMA documents are specified by means of *UIMA Type System Descriptor* (UIMA TSD) files in order to enable interchangeability. The usage of UIMA in numerous projects and tools shows that this framework can be seen as a *de facto* standard in the context of NLP (Wilcock, 2017). However, despite its widespread usage, it is surprising that native database support is rarely found for UIMA documents. An exception is (Fette et al., 2013) who introduced an approach to storing UIMA documents by means of relational databases. In this article we want to close the gap between UIMA and databases by looking at several database systems beyond relational ones. In order to make UIMA-based projects (as, for example, TREEANNOTATOR (Helfrich et al., 2018)) conform to the requirements of modern text technology, database support is essential. This becomes clear when comparing database usage with file system-based storage and retrieval of UIMA documents. Generally speaking, approaches using file system-based storage lead to several bottlenecks:

(a) **No redundancy:** The document-based schema of the UIMA framework, which is usually stored via XMI[1] files, means that the results of UIMA processes cannot be stored decentrally (e.g., in a cloud). However, exclusively storing results in one file system without being based on a suitable backup concept can lead to data loss in the event of a system failure.

(b) **No query options:** To extract information from UIMA documents, these documents must first be completely imported into the cache. However, it is not possible to select only those UIMA documents that are required for a given application.

(c) **No shared use:** In projects including several participants, sharing of results is only possible by copying files. This in turn means that data security and tracking of file versioning must be ensured not by the system, but by the users.

(d) **No data security:** UIMA documents are not bound to data security and cannot be made available individually to third parties.

To avoid all these bottlenecks, we developed the so-called *UIMA Database Interface* (UIMA DI). It enables the generic use of UIMA documents in database systems in the context of NLP. In this paper, we explain the architecture of UIMA DI and evaluate it against file storage systems: in Section 2. we give a brief overview of application scenarios of UIMA DI. These scenarios will be referred to throughout the paper to exemplify the use of UIMA DI. In Section 3., we explain the architecture and the current implementation of UIMA DI. Its evaluation is documented in Section 4. and Section 5.. Finally, Section 6. and 7. summarize and give an outlook on future work.

2. Application cases

UIMA DI is currently used in three different application scenarios:

TEXTIMAGER TEXTIMAGER (Hemati et al., 2016) is an application for analyzing and visualizing textual data based on UIMA. Further, a tool for annotating hierarchical text relations (RST), called TREEANNOTATOR, is currently under development that utilizes the preprocessing

[1] http://www.omg.org/spec/XMI/

methods of TEXTIMAGER. To meet the needs of automatic text processing (TEXTIMAGER) and manual text annotation (TREEANNOTATOR) we decided for UIMA TSD as the underlying data model, while UIMA DI is used to manage the resulting UIMA documents.

STOLPERWEGE STOLPERWEGE (Mehler et al., 2017) is an application for modeling and visualizing the biographies of victims of the Holocaust. For this purpose, geo-data has to be managed in order to be used in related search queries. Once more, UIMA DI has the task to perform this data management.

EHUMANITIES DESKTOP The EHUMANITIES DESKTOP (Gleim et al., 2012) provides a range of applications and services for humanities scholars. This includes the so-called *ResourceManager* (see Figure 1): using UIMA DI, the ResourceManager makes UIMA documents independent resources that can be integrated into larger contexts (e.g., projects or repositories) and, therefore, avoids Bottleneck (c) of Section 1. This also includes the *AuthorityManager*, which enables the management of resource access rights at the level of individual users or groups of them, thus enabling privacy and data security. Thus, by being based on UIMA DI, the *AuthorityManager* avoids Bottleneck (d).

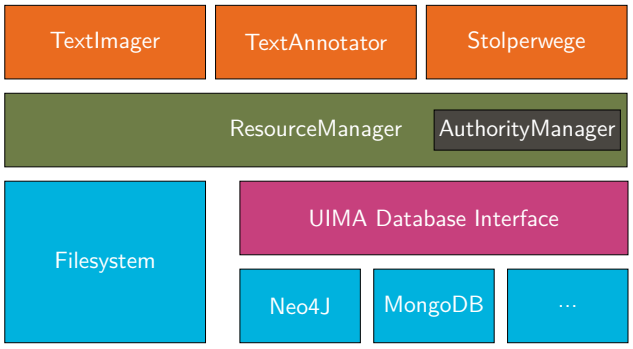

Figure 1: Application scenarios (orange) and software components (green) using UIMA DI.

3. UIMA Database Interface

UIMA TSD is an architecture that provides format and definition concepts for annotations stored in XMI files. Since UIMA TSD does not provide a built-in database solution, it is necessary to extend it. This is done by UIMA DI. To ensure uniqueness of documents managed by UIMA DI, it is necessary to reference exported XMI documents with their corresponding database records. For this reason, UIMA DI enables the generic storage and retrieval of UIMA documents by means of different databases.

3.1. Architecture

UIMA DI is an interface developed in Java and hosted at GitHub (GPLv3 license). In order to resolve dependencies, Apache Maven[2] is used. The core functions of UIMA DI address the following tasks:

(a) Analysis of UIMA TSD Analysis of the embedded UIMA TSD for generating meta information to enable the document to be stored in the used database which is implemented in UIMA DI. This is realized by analyzing the specified UIMA TSD. In addition, annotation templates can be created with the generated meta information, which can be used, for example, by TREEANNOTATOR and the project STOLPERWEGE (see Section 2.).

(b) (De-)Serialization of CAS UIMA documents are encoded in terms of CAS (UIMA COMMON ANALYSIS SYSTEM) (Götz and Suhre, 2004) representations. CAS files are serialized according to JSON[3] to be stored completely in the database. This enables lossless de-serialization of the entire UIMA documents.

(c) Dynamic ID insertion Each UIMA document and its content is mapped to a "Subject OF Analysis" (*sofa*) element. To prepare a UIMA document for database storage, an empty database entry is created and the resulting ID is stored as an additional sofa within the UIMA document. This sofa is identified during the import of previously exported UIMA documents: it enables a synchronization with the existing database entry. The generated database entry will then be updated with the previously generated meta information and the serialized CAS representation. Section 3.2. explains the corresponding data structures involved in the current implementation of UIMA DI regarding two databases: MongoDB and Neo4J. Figure 2 gives a visual depiction of the underlying process of document conversion.

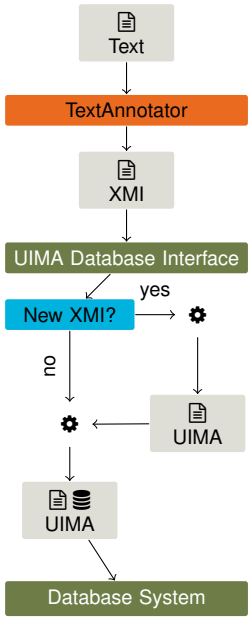

Figure 2: Process diagram of UIMA DI: A text is processed by TEXTANNOTATOR; then, the resulting XMI document x is sent to UIMA DI. If x is already stored in the database, the corresponding database entry is updated. If not, the XMI document is prepared for being processed by UIMA DI and stored in the underlying database system.

[2]https://maven.apache.org/

[3]http://www.json.org/

3.2. Current Implementations

The current version (1.0) of UIMA DI implements two database systems, namely MongoDB (3.2.1.) and Neo4J (3.2.2.). These databases utilize schema-free models and therefore allow for external database queries without presupposing knowledge about the database structure (as required, for example, by relational databases). Further databases can be easily adapted by the interface-based programming on which UIMA DI is based. Note also that our implementation includes a RESTful web service and analysis methods for the integrated UIMA TSD. The REST methods are implemented using Spark[4] while Swagger[5] is used for documenting the API of UIMA DI. Note that MongoDB and Neo4J allow for cloud-based storage enabling distributed data usage. Based on this analysis, we can state that our current implementation avoids Bottleneck (a) of Section 1. Furthermore, since the latter two databases provide highly specialized query languages, we also avoid Bottleneck (b).

3.2.1. MongoDB

We developed a UIMA Database Interface (UIMA DI) that allows for storing UIMA documents by means of the document-oriented database MongoDB[6]. When the CAS structure is serialized into JSON (see Section 3.1. above), three data fields are created as entries in MongoDB in the context of the applications enumerated in Section 2. (see Figure 3 for an example):

- **uima:** The serialized CAS representation of any input UIMA document is stored as such in MongoDB using the data field named *uima* (see Figure 3).
- **meta** is an array storing information about UIMA TSD-related data types instantiated by the input document. In this case, non-primitive data types are stored as references to objects.
- **geo** stores geospatial information (as needed, e.g., by the project STOLPERWEGE – see Section 2.). A geospatial index has been added for this purpose.

3.2.2. Neo4J

The graph database Neo4J[7] uses another storage structure than MongoDB (see Figure 3.2.2.): UIMA documents are now stored as nodes whose attributes denote simple data types and relations to other UIMA documents. The names of the attributes and relations are defined by the UIMA TSD associated with the document. The number of relationships to other objects is determined by the respective references in the UIMA document. As in the case of MongoDB, each node contains the complete serialization of the respective UIMA document (see Figure 4).
Note that we implemented UIMA DI by example of MongoDB and Neo4j to capture two paradigms of data modeling, that is, graph-oriented (Neo4j) and document-oriented modeling (MongoDB). In this way, UIMA DI already covers a wider range of databases. In the next section, we show how to evaluate this implementation.


```
{
   "_id" : ObjectId("5968
      ↪ e2fbb14e9e40e23b7e1e"),
   "geo" : {
      "coordinates" : [
         16.9327791,
         54.4322101
      ],
      "type" : "Point"
   },
   "meta" : {
      "firstName" : "Gustav",
      "lastName" : "Hoch",
      "deathDate" : "5968
         ↪ e2fbb14e9e40e23b7e1e",
      "id" : "5968e2fbb14e9e40e23b7e1e
         ↪ ",
      "type" : "org.hucompute.
         ↪ publichistory.datastore.
         ↪ typesystem.Person",
      "value" : "Gustav, Hoch",
      "birthDate" : "5968
         ↪ e2fbb14e9e40e23b7e1e"
   },
   "uima" : {
      "childNodes" : [
         {
            "_indexed" : 0,
            "_id" : 1,
            "sofaID" : "5968
               ↪ e2fbb14e9e40e23b7e1e
               ↪ ",
            "tagName" : "uima.cas.Sofa
               ↪ ",
            "sofaNum" : 2
         },
[...]
         {
            "_ref_deathDate" : 46,
            "_indexed" : 2,
            "firstName" : "Gustav",
            "lastName" : "Hoch",
            "end" : 0,
            "_id" : 8,
            "tagName" : "org.hucompute.
               ↪ publichistory.
               ↪ datastore.typesystem.
               ↪ Person",
            "_ref_sofa" : 1,
            "begin" : 0,
            "value" : "Gustav, Hoch",
            "_ref_birthDate" : 32
         }
      ]
   }
}
```


Figure 3: Excerpt from an entry of a UIMA document stored in MongoDB. The example is taken from the project STOLPERWEGE.

4. Experiments

Since there is no database solution for UIMA documents yet, our experiments are limited to measuring the reading, writing and querying time of UIMA documents in the respective database system and comparing it to the same op-

[4] http://sparkjava.com
[5] https://swagger.io/
[6] https://www.mongodb.com/
[7] https://neo4j.com/

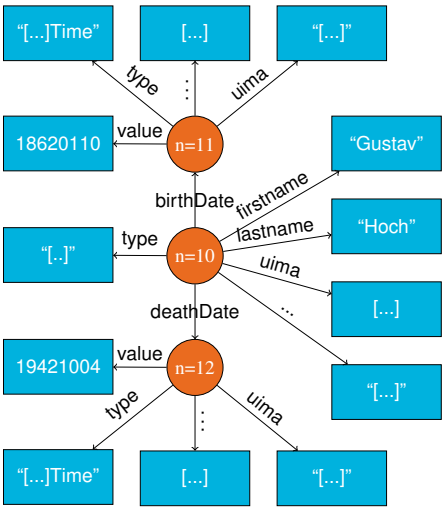

Figure 4: Schematic depiction of an entry of a UIMA document stored in Neo4J. The example is taken from the project STOLPERWEGE.

erations performed on the native file system. As test documents, we choose the German and English Wikipedia articles of six top candidates for the federal election in Germany 2017 (Angela Merkel, Martin Schulz, Katrin Göring-Eckardt, Sahra Wagenknecht and Alexander Gauland). The texts are preprocessed by means of TEXTIMAGER using the following taggers:

- Stanford NER Tagger[8]
- Stanford POS Tagger[9]
- Heideltime[10]

In this example, UIMA DI includes the following UIMA TSD as a result of using Maven as an interface to TEXTIMAGER data:

- Heideltime and
- DKPro-Core (contains the taggers listed above) (Eckart de Castilho and Gurevych, 2014).

After having preprocessed and stored the test documents using the systems to be evaluated (i.e., Neo4J, MongoDB and XMI files), we performed our evaluation. For this purpose, we selected all German and English texts separately to determine which storage system is the fastest to retrieve the corresponding UIMA documents. To this end, we did not optimize the (de-)serialization of the UIMA-CAS involved.

5. Results

Our results show that the read (see Figure 5) and write speed (see Figure 6) of both databases are slower than what is reached by the UIMA method of storing documents in XMI files. This is caused by the (de-)serialization of CAS elements into the JSON format. A second reason concerns the overhead induced by generating meta information. The more annotations a UIMA document contains, the more time is needed because different attributes and relations (based on the database) have to be created. However, for projects in which UIMA documents have to be shared, distributed document management is more important than reading and writing time. The same is true for projects which require sophisticated database query capabilities. Approaches being based on the file system require to import all XMI files to extract queried content. Obviously, this may induce a problematic memory load. In contrast to this, using UIMA DI as an interface to a database like Neo4j enables sophisticated content-related queries which can be faster than file-based approaches as shown in Figure 7.

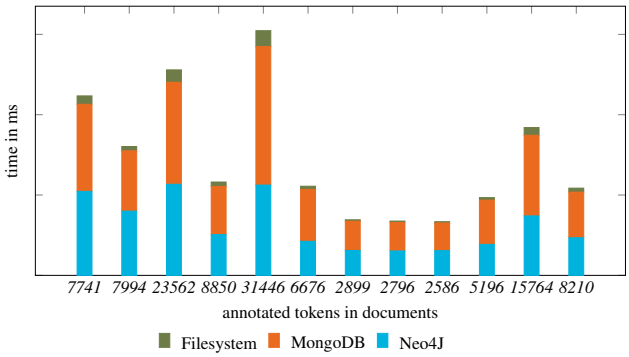

Figure 5: Measurement results with respect to reading.

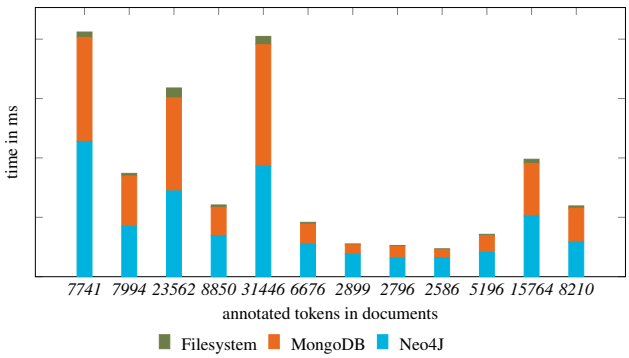

Figure 6: Measurement results with respect to writing.

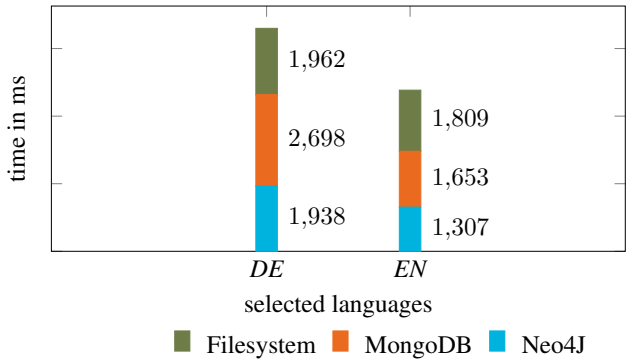

Figure 7: Measurement results with respect to querying.

[8]https://nlp.stanford.edu/software/CRF-NER.shtml
[9]https://nlp.stanford.edu/software/tagger.shtml
[10]https://github.com/HeidelTime/heideltime

6. Conclusion

We introduced UIMA DI as a generic interface to databases for managing UIMA documents. Our evaluation shows that our implementation of UIMA DI outperforms file-based systems when considering database queries. In projects related to digital humanities, querying UIMA documents in a sophisticated manner can be more important than optimizing reading and writing of UIMA documents as a whole. This is enabled by UIMA DI. Beyond that, several other bottlenecks of the file-based approach to processing UIMA documents are avoided by UIMA DI. Among other things, this concerns data security and protection. UIMA DI is available under license GPLv3 via GitHub[11] and is open to anyone interested in using or extending it.

7. Future Work

The future development of UIMA DI will primarily involve the implementation of further database systems like Elasticsearch[12] and Blazegraph[13]. With the help of Elasticsearch, we can avoid the document size limitation of MongoDB while ensuring a similar performance. Further, it is planned to accelerate the conversion from CAS to JSON by developing a converter specialized on this task.

8. Bibliographical References

Da Silva, P. P., McGuinness, D. L., and Fikes, R. (2006). A proof markup language for Semantic Web services. *Information Systems*, 31(4):381–395.

Eckart de Castilho, R. and Gurevych, I. (2014). A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11, Dublin, Ireland, August. Association for Computational Linguistics and Dublin City University.

Ferrucci, D., Lally, A., Verspoor, K., and Nyberg, E. (2009). Unstructured Information Management Architecture (UIMA) Version 1.0. OASIS Standard, Mar.

Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., et al. (2010). Building Watson: An overview of the DeepQA project. *AI magazine*, 31(3):59–79.

Fette, G., Toepfer, M., and Puppe, F. (2013). Storing UIMA CASes in a relational database. *Unstructured Information Management Architecture (UIMA)*, page 10.

Gleim, R., Mehler, A., and Ernst, A. (2012). SOA implementation of the eHumanities Desktop. In *Proceedings of the Workshop on Service-oriented Architectures (SOAs) for the Humanities: Solutions and Impacts, Digital Humanities 2012, Hamburg*.

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

Helfrich, P., Rieb, E., Abrami, G., Lücking, A., and Mehler, A. (2018). TreeAnnotator: Versatile Visual Annotation of Hierarchical Text Relations. In *Proceedings of the 11th Edition of the Language Resources and Evaluation Conference, May 7 - 12*, LREC 2018, Miyazaki, Japan.

Hemati, W., Uslu, T., and Mehler, A. (2016). TextImager: a Distributed UIMA-based System for NLP. In *Proceedings of the COLING 2016 System Demonstrations*. Federated Conference on Computer Science and Information Systems.

Kano, Y., Baumgartner, J. W. A., McCrohon, L., Ananiadou, S., Cohen, K. B., Hunter, L., and Tsujii, J. (2009). U-Compare: share and compare text mining tools with UIMA. *Bioinformatics*, 25(15):1997–1998.

Kassner, L., Hirmer, P., Wieland, M., Steimle, F., Königsberger, J., and Mitschang, B. (2017). The Social Factory: Connecting People, Machines and Data in Manufacturing for Context-Aware Exception Escalation. In *Proceedings of the 50th Hawaii International Conference on System Sciences*.

Kreimeyer, K., Foster, M., Pandey, A., Arya, N., Halford, G., Jones, S. F., Forshee, R., Walderhaug, M., and Botsis, T. (2017). Natural language processing systems for capturing and standardizing unstructured clinical information: A systematic review. *Journal of Biomedical Informatics*, 73(Supplement C):14–29.

Mehler, A., Abrami, G., Bruendel, S., Felder, L., Ostertag, T., and Spiekermann, C. (2017). Stolperwege: An App for a Digital Public History of the Holocaust. In *Proceedings of the 28th ACM Conference on Hypertext and Social Media*, HT '17, pages 319–320, New York, NY, USA. ACM.

Niekler, A., Wiedemann, G., and Heyer, G. (2017). Leipzig Corpus Miner - A Text Mining Infrastructure for Qualitative Data Analysis. *CoRR*, abs/1707.03253.

Ogren, P. V., Wetzler, P. G., and Bethard, S. (2008). ClearTK: A UIMA Toolkit for Statistical Natural Language Processing. In *Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP. Proceedings of LREC 2008*.

Patterson, O. V., Freiberg, M. S., Skanderson, M., J. Fodeh, S., Brandt, C. A., and DuVall, S. L. (2017). Unlocking echocardiogram measurements for heart disease research through natural language processing. *BMC Cardiovascular Disorders*, 17(1):151, Jun.

Rudzewitz, B., Ziai, R., De Kuthy, K., and Meurers, D. (2017). Developing A Web-based Workbook for English Supporting the Interaction of Students and Teachers. In *Proceedings of the Joint 6th Workshop on NLP for Computer Assisted Language Learning and 2nd Workshop on NLP for Research on Language Acquisition at NoDaLiDa, Gothenburg*, number 134, pages 36–46. Linköping University Electronic Press.

Savova, G. K., Masanz, J. J., Ogren, P. V., Zheng, J., Sohn, S., Kipper-Schuler, K. C., and Chute, C. G. (2010). 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.

Wilcock, G. (2017). The Evolution of Text Annotation Frameworks. In Nancy Ide et al., editors, *Handbook of*

[11] https://github.com/texttechnologylab/UIMADatabaseInterface
[12] https://www.elastic.co
[13] https://www.blazegraph.com/

Linguistic Annotation, pages 193–207. Springer Netherlands, Dordrecht.

Zeldes, A., Lüdeling, A., Ritz, J., and Chiarcos, C. (2009). ANNIS: a search tool for multi-layer annotated corpora. In *Proceedings of Corpus Linguistics*.