LDK 2023

Knowledge Storage Ecosystem: an Open Source Tool for NLP Results Management (Documents and Semantic Information)

Julian Moreno-Schneider and **Maria Gonzalez Garcia** and **Georg Rehm**
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Alt-Moabit 91c, 10559
Berlin, Germany
julian.moreno_schneider@dfki.de

Abstract

This paper presents the Knowledge Storage Ecosystem (KSE), a tool developed for the support of storage and management of knowledge, particularly linked data. The KSE can manage not only knowledge (the semantic information that is extracted from documents using different NLP procedures), but also original documents and full text indexes, allowing full text search in an efficient way, increasing the usability of extracted knowledge in a wide variety of applications. A graphical user interface has also been developed to facilitate the usability of the KSE, allowing this tool to reach a larger audience.

1 Introduction

The development of various NLP technologies over the last decades has resulted in a wide variety of tools, services and libraries to analyze texts, thus being able to generate an enormous amount of semantic information contained in these texts. Handling all this semantic information in knowledge bases has seen a surge in popularity in recent years, because this structured way of storing information enables inference and reasoning.

The widespread use of knowledge bases has fostered the development of tools or platforms that allow the storage and management of this type of semantic information (see Section 2). The main problem we encountered is that there is no platform that allows knowledge management in addition to the original documents on which NLP processes are carried out.

In this article we present a tool that allows the management and use of knowledge as well as documents, facilitating the joint management of these two modes of conveying information. This idea is not completely new, since the World Wide Web Consortium[1] (W3C) already defined this type of systems under the concept of the Linked Data Platform (Arwe et al., 2015). This concept only encompasses the operation rules, not stating anything about the information stored in such a system. Therefore, we go one step further by labeling the original documents as first class citizens inside our platform. The main problems that we have found in similar systems and that we are trying to solve with this platform are: (i) joint management of documents and related knowledge (especially semantic annotations); and (ii) synchronization of the stored information on CRUD (Create, Retrieve, Update and Delete) operations. In summary, the main contributions of this article are the following:

1. We have defined and implemented a platform, namely Knowledge Storage Ecosystem (KSE), that allows the joint management of knowledge, source documents and full text indexes.
2. We have designed and started the implementation of a graphical user interface that simplifies the management and usage of KSE.
3. We released the entire code of our tool (see Section 3).

2 Similar Systems

The management of semantic information (NLP annotation results) has been covered by many approaches from different perspectives. Some are more focused on the storage of linked data, platforms adhering to the Linked Data Platform standard, or combined systems including file storage or full texts. Many different tools that can be used to manage and store linked data have been developed, summarized in surveys such as those by (Zhang et al., 2021) and (Wylot et al., 2018). Platforms particularly focused on linked data are less abundant, but some alternatives exist. One example is Apache Marmota[2]. It is composed of

[1] https://www.w3.org

[2] https://marmotta.apache.org/index.html

several modules (for example, SPARQL module, LDP module, Reasoner module or security module among others), but apart from that, the project also develops some libraries that can be used separately such as KiWi Triple Store, LDClient or LDCache. OpenLink Virtuoso (Open-Source Edition)[3] is another tool that combines Relational, Graph, and Document Data management. In many cases, Linked Data Platforms have been developed to match a specific use case or domain, such as SeCold (Keivanloo et al., 2012), an open platform for sharing software datasets; QuerioCity (Lopez et al., 2012), a platform to manage (catalog, index and query) heterogenous information (special interest on stream integration) coming from cities; a platform that combines unstructured data from scientific literature and structured data from publicly available biological databases (Singh et al., 2020); or LinkedLab (Darari and Manurung, 2011), a Linked Data based solution for data management regarding research communities. A tool similar to ours is Trellis-LDP[4], a platform for building linked data applications that allows storage and management of linked data and documents, but the formats of documents is rather limited, and they do not included full text search as a feature. The main issue we have with Trellis is that it does not control duplicate documents. KIM (Popov et al., 2003) provides exactly the same functionality as our system (based on GATE[5], RDF Sesame[6] and Lucene[7]), even integrating the information extraction. Its issues as we perceived them are that it does not store the source documents, and it is a commercial product (only freely available for research). To the best of our knowledge, there is no open-source alternative that provides the functionalities that our system is offering.

3 Knowledge Storage Ecosystem

In this article we have designed and developed a tool that allows the management of semantic information together with source documents. This tool is called Knowledge Storage Ecosystem (KSE) and its main functionality is the management of different types of information (knowledge, source documents, full text indexes) that are related and interconnected between them.

[3]https://vos.openlinksw.com/owiki/wiki/VOS/
[4]https://www.trellisldp.org
[5]https://gate.ac.uk/
[6]https://metacpan.org/pod/RDF::Sesame::Repository
[7]https://lucene.apache.org/

The architecture of KSE (shown in Figure 1) is modular and composed of four components, apart from the graphical user interface, that is considered an external extension to the KSE.

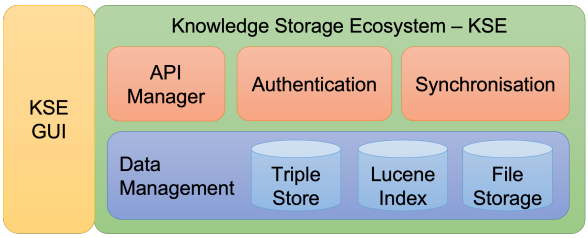

Figure 1: Architecture of the Knowledge Storage Ecosystem

With this first modular architecture of the KSE we cover the following requirements: (i) the storage of semantic information (knowledge) in a specific tool, namely triple store (see Section 3.1.2) allowing inference over the semantic information; (ii) indexing of full text using Lucene (see Section 3.1.3) to simplify search in source documents; (iii) handling of source documents (see Section 3.1.4) and linking them with semantic information through the document identifier in the triple store; and (iv) a first attempt to handle the synchronization of information inside the tool between information types (see Section 3.2).

The entire code, technical documentation and usage examples of KSE are available at `https://gitlab.com/speaker-projekt/knowledge-management/knowledge-storage-ecosystem`.

3.1 Data Management

The first and most important component of the KSE is the data management module, whose main functionality is the management (storage, recovery, modification and deletion) of information inside the system. The information stored in this system is organized in three different categories: source files or documents (PDF, DOCX, TXT, etc.), semantic information (knowledge as Linked Data) associated with the source document and full text obtained from the source document. For each category, the KSE has a specific information storage module, as described below.

3.1.1 Data Structures

The management of the information inside of KSE is made through specific data structures, that we have defined for this purpose. The

most relevant structures defined are `Collection`, `LDDocument` and `Triple`, as well as a `Converter` that allows us to convert these structures to files. `Collection` is a simple structure that has been defined to manage the set of documents that are grouped under the same collection. A collection consists of a *collection identifier*, a *name*, a *description*, and a *list of documents*. `LDDocument` is a more complex structure, because it has to group the three types of information related to a source: knowledge, source document and full text. A document is composed of the following variables: *document identifier*, *text*, a *list of triples* and *path* of the source document. `Triple` is a simple structure, and it is a set of three elements (subject, predicate and object) of a relationship or basic semantic unit. This structure has been defined to facilitate its internal management in the system. The `Converter` is responsible for (de)serializing data structures in/from files, so that they can be included in the KSE or exported from the KSE. It supports standardized semantic web formats such as `RDF`, `TURTLE` or `JSON-LD`.

When a document is created in the system (by uploading it via the REST API (Richardson and Ruby, 2007)), the system assigns it a unique identifier. This identifier is obtained from an encryption algorithm applied to the text of the document. The algorithm used is SHA-256 Cryptographic Hash Algorithm (Handschuh, 2011). There is a possibility that the text of the document is not provided by the user who adds it to the system, in which case an identifier is generated based on the timestamp in which the document was added. We are currently working on improving this process to use the binary content of the original document, thus being able to manage duplicates on the platform, referring to the same document and not generating a new one, as is the case with some alternatives.

3.1.2 Semantic Information Storage

The semantic information storage, or triple store, is a module that is responsible for the efficient management of knowledge (semantic information). There are many tools that are already implemented for performing this task, therefore we decided not to reinvent the wheel and use one of the available options.

We decided to use OpenLink Virtuoso (Open-Source Edition) because we already used it in several projects and the learning curve was shorter. Besides, Virtuoso offers the possibility to easily install as an independent module and use it through socket calls, which minimizes the potential of interconnection problem within modules.

In order to perform the CRUD operations with Virtuoso, we have defined specific SPARQL queries. Due to space limitation we only show one document creation example in Listing 1.

```
sparql insert into graph <col_1> {
    docURI sp_ont:documentId
    "docId" .
    <subject> <predicate>
    <object> .
}
```

Listing 1: Example of SPARQL query for creating a document in Virtuoso.

3.1.3 Full Text Index

This module allows the search for textual information in documents in an extremely efficient way, something that is supported in triple stores, but is inefficient if text gets longer. Therefore, we are using the well-known and extensively used and tested Lucene[8] (McCandless et al., 2010) tool. This is the basic Apache technology for full text search. Although in last years newer technologies have been developed (such as Solr or ElasticSearch), which include much more functionality, we decided to stay with the most basic technology in order to keep it simple and easy to use and integrate in our tool. Besides, the direct usage of Lucene allows us to redefine any component that we need, for example, the Document Parsers needed for the specific `LDDocument` structure.

We have defined a simple index containing three fields: *identifier* inside the Lucene index, *KSE document identifier* and *full text*. At indexing we use two different analysers to process these fields: A `Whitespace analyser` for the identifiers, and an `N-Gram Analyser` for the text. The N-Gram analyser converts the text in n-grams ($n = 3$) in order to index them as the minimal textual unit.

3.1.4 File Storage

This module is responsible for storing the original files within the platform. To do this, and in order to implement the module as simply as possible, we have used the file system. Original files are stored as files in a folder that is identified by the name of the collection the files belong to, for example, if we upload a file called *'Report.pdf'* and add it to the *'shared_documents'* collection, then the file system will be as shown in Listing 2.

[8] https://lucene.apache.org

```
kse_collections/
\--- shared_documents
     \--- Report.pdf
1 directory, 1 file
```

Listing 2: Folder structure of the file storage after including a file named 'Report.pdf' to the collection 'shared_documents'.

The main functionality of this module is to keep accessible the original documents on which the NLP analyses are performed. In this way one can reproduce experiments or display results directly on the source documents, for example, integrating entity highlights in `PDFs`.

3.2 Synchronization Module

The synchronization of the information is essential in our system, because when integrating other tools (Lucene, Virtuoso, etc.), it may happen that the semantic information related to a document is modified, while this document is included in the result of a textual search. Or even worse, that the document is deleted, but continues to be used in searches or statistics until it is permanently deleted from all tools. For this we have defined a synchronization mechanism that prohibits or blocks the use of a document if it is being used by some modification operation (update or delete). For this we use the synchronization mechanisms of Java (through three methods: `documentIsBlocked(docId){...}`, `blockDocument(docId){...}` and `unblockDocument(docId){...}`), together with a HashMap that stores the identifiers of all documents stored in the system (`HashMap blockedDocuments`).

3.3 API Manager

This module is responsible for the access to the entire tool functionality, from administrative control to information management through `HTTP REST API` endpoints.

The administrative control of the tool is done through configuration files, which are included directly in the source code including examples (available here). Nevertheless, we have included endpoints to manage these configuration parameters, being able to create, read, modify or delete them. All the endpoints defined for administrative tasks are listed in Figure 2.

The information management is completely done through endpoints that are accessible through

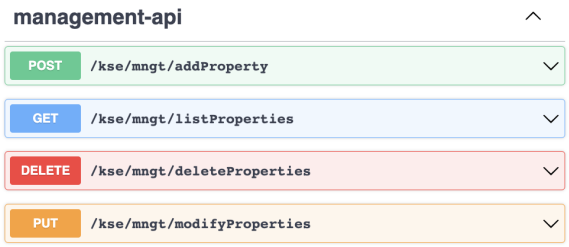

Figure 2: Administration endpoints.

`HTTP REST API`, and are divided into two categories: endpoints for CRUD operations (Create, Retrieve, Update and Delete) of information, regarding Collections and Documents (7 endpoints), and endpoints for information search: `SPARQL` for knowledge and full text search for document content. In both cases, the original documents can also be retrieved. The document content must be provide manually by users, because automated PDF scraping/content extraction is still not supported.

All the endpoints defined for information management are listed in Figure 3.

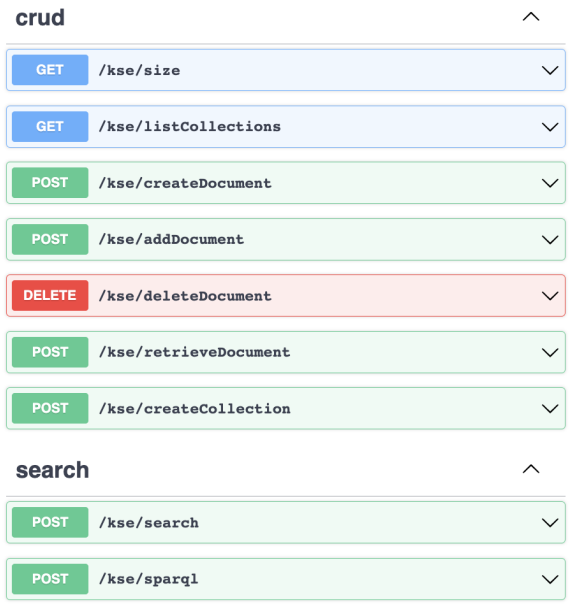

Figure 3: Information management endpoints.

3.4 Authentication

The authentication will not be limited to access the website, but it will be a much more detailed and resource-specific authentication policy. The basic authentication unit will be a *'user'*, which will be granted access to different resources: (i) websites in the graphical user interface that this user can access; (2) information resources (Collections,

Documents, Semantic annotation of documents or full text indexes) that this user can use, being able to specify if the user can read, write, etc. the resources.

Actually these roles have not been implemented, but we are planning to integrate Keycloak[9] as independent authentication module, which we will leave to future work.

3.5 Graphical User Interface

The system that we present in this article (Knowledge Storage Ecosystem) has been designed with its integration in larger software systems in mind, hence the access to it has been predetermined through the `HTTP REST API`. This way of accessing the system requires users to have knowledge of programming. To ease interfacing with the system, we additionally created a graphical user interface (GUI) that allows users without programming knowledge to use KSE as well.

The graphical user interface that we present here is a Web system that has been designed for managing all the functionalities of KSE that are accessible through `HTTP REST API` endpoints. Its main objective is to be functional and styling the interface is added to the list of future work items. The existing pages (shown in Figure 4) in the graphical interface are: (1) Dashboard: introductory page where KSE is presented and links to the other pages are provided; (2) Management/Configuration: management of configuration parameters; (3) Users: user management; (4) Collections: management of collections, as well as being able to create new collections; (5) Collection: management of an individual collection, as well as being able to add documents to it; (6) Document: management of individual documents; (7) Text Search: KSE can be searched textually. The results are displayed in document list format; and (8) Sparql Endpoint: SPARQL queries can be made to the KSE. The results are displayed in table format.

The code and technical documentation of the graphical user interface for KSE is available at https://gitlab.com/speaker-project/knowledge-management/kse-graphical-user-interface.

4 Conclusions and Future Work

The joint storage of documents and semantic information associated with them is not a resolved task. While there are solutions that have approached this problem from different angles, none of these solutions seem definitive, and there are unresolved issues. Besides, there are few existing tools that allow this functionality out-of-the-box. Therefore, we have implemented a system that performs this functionality in a simple way.

We implemented a solution that offers the user the desired functionality of CRUD operations over source documents and semantic information. The management of the information is done through `HTTP REST API` endpoints. To simplify that, we have also implemented and published a graphical user interface to use and manage the KSE system.

One of the important issues that we had to address in the implementation process is the synchronization of information between data storage tools.

There are several open issues that are kept for future work. The main items are:

- Integrating external Linked Data sources, such as Knowledge Bases (DBpedia, Wikidata, Yago, etc.) is foreseen. This is the first thing we plan to work on.
- Styling the interface so that aesthetics and ease of use are taken into account in the implementation.
- Implementing the authentication module by integrating Keycloak.
- Evaluating the system. The experiments to be carried out on this system are based on the evaluation of different user-related metrics that allow us to determine the usability, simplicity and performance of the system.

The link to the demonstration video is https://youtu.be/4T6ujG6MHe4.

Acknowledgments

The work presented in this article has received funding from the German Federal Ministry for Economic Affairs and Climate Action (BMWK) through the project SPEAKER (no. 01MK19011).

References

John Arwe, Steve Speicher, and Ashok Malhotra. 2015. Linked data platform 1.0. W3C recommendation, W3C. Https://www.w3.org/TR/2015/REC-ldp-20150226/.

[9] https://www.keycloak.org

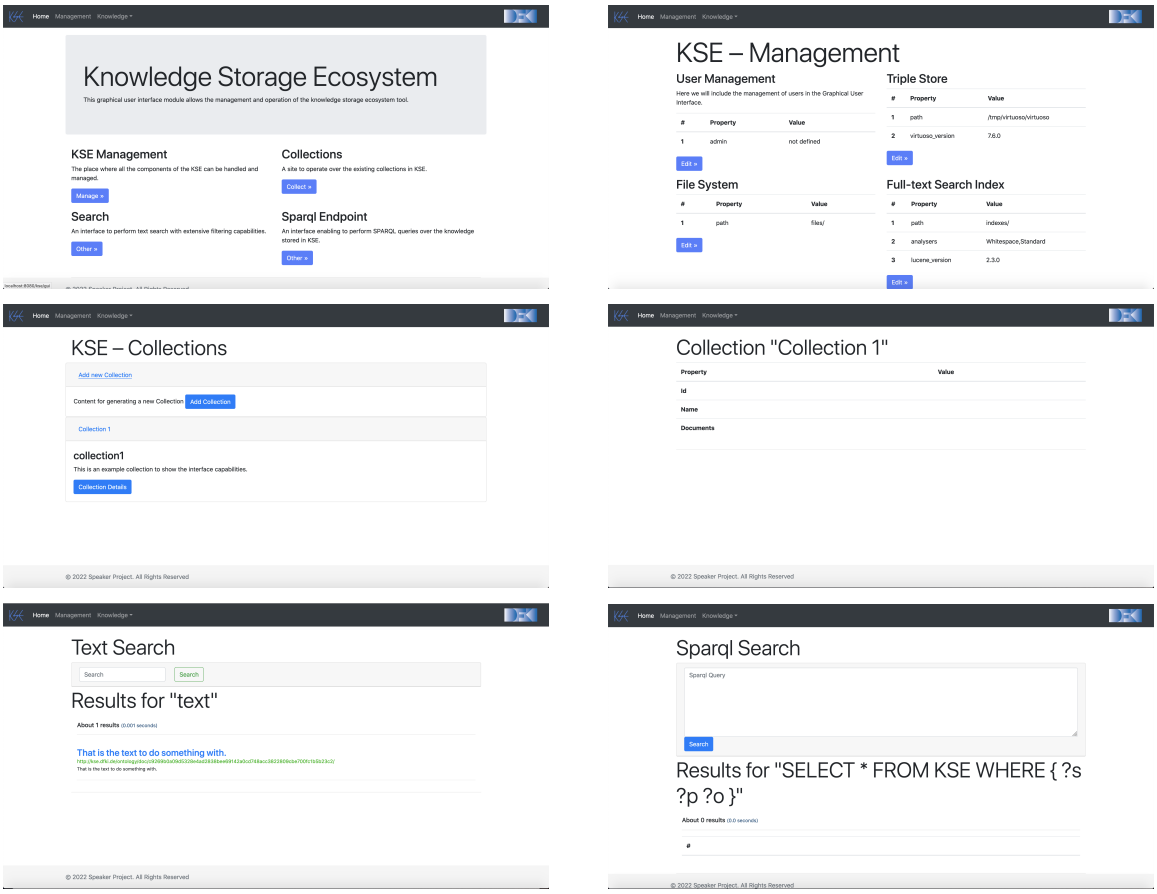

Figure 4: Screenshots of the different pages of the graphical user interface: top left is the dashboard, top right is the management and configuration, middle left is the collections management, middle right is the individual collection management, bottom left is the full text search and bottom right is the sparql endpoint.

Fariz Darari and Ruli Manurung. 2011. Linkedlab: A linked data platform for research communities. In *2011 International Conference on Advanced Computer Science and Information Systems*, pages 253–258.

Helena Handschuh. 2011. Sha-0, sha-1, sha-2 (secure hash algorithm). In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 1190–1193. Springer.

Iman Keivanloo, Christopher Forbes, Aseel Hmood, Mostafa Erfani, Christopher Neal, George Peristerakis, and Juergen Rilling. 2012. A linked data platform for mining software repositories. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 32–35.

Vanessa Lopez, Spyros Kotoulas, Marco Luca Sbodio, Martin Stephenson, Aris Gkoulalas-Divanis, and Pól Mac Aonghusa. 2012. Queriocity: A linked data platform for urban information management. In *The Semantic Web – ISWC 2012*, pages 148–163, Berlin, Heidelberg. Springer Berlin Heidelberg.

Michael McCandless, Erik Hatcher, and Otis Gospodnetic. 2010. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., USA.

Borislav Popov, Atanas Kiryakov, Angel Kirilov, Dimitar Manov, Damyan Ognyanoff, and Miroslav Goranov. 2003. Kim – semantic annotation platform. In *The Semantic Web - ISWC 2003*, pages 834–849, Berlin, Heidelberg. Springer Berlin Heidelberg.

Leonard Richardson and Sam Ruby. 2007. *RESTful Web Services*. O'Reilly, Beijing.

Gurnoor Singh, Arnold Kuzniar, Matthijs Brouwer, Carlos Martinez-Ortiz, Christian W. B. Bachem, Yury M. Tikunov, Arnaud G. Bovy, Richard G. F. Visser Finkers, and Richard. 2020. Linked data platform for solanaceae species. *Applied Sciences*, 10(19).

Marcin Wylot, Manfred Hauswirth, Philippe Cudré-Mauroux, and Sherif Sakr. 2018. Rdf data storage and query processing schemes: A survey. *ACM Comput. Surv.*, 51(4).

Fu Zhang, Qingzhe Lu, Zhenjun Du, Xu Chen, and Chunhong Cao. 2021. A comprehensive overview of rdf for spatial and spatiotemporal data management. *The Knowledge Engineering Review*, 36:e10.