OSTRICH: Versioned Random-Access Triple Store

Ruben Taelman
IDLab — Ghent University — imec
ruben.taelman@ugent.be

Miel Vander Sande
IDLab, Ghent University — imec
miel.vandersande@ugent.be

Ruben Verborgh
IDLab — Ghent University — imec
ruben.verborgh@ugent.be

ABSTRACT

The Linked Open Data cloud is evergrowing and many datasets are frequently being updated. In order to fully exploit the potential of the information that is available in and over historical dataset versions, such as discovering evolution of taxonomies or diseases in biomedical datasets, we need to be able to store and query the different versions of Linked Datasets efficiently. In this demonstration, we introduce OSTRICH, which is an efficient triple store with supported for versioned query evaluation. We demonstrate the capabilities of OSTRICH using a Web-based graphical user interface in which a store can be opened or created. Using this interface, the user is able to query *in*, *between*, and *over* different versions, ingest new versions, and retrieve summarizing statistics.

KEYWORDS

Linked Data; RDF; Versioning; OSTRICH; Triple Store

ACM Reference Format:
Taelman, R. and Vander Sande, M. and Verborgh, R. 2017. OSTRICH: Versioned Random-Access Triple Store. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23—27, 2018, Lyon, France*. ACM, New York, NY, USA, 4 pages. *http://dx.doi.org/10.1145/3184558.3186960*

1 INTRODUCTION

Many of the Linked Datasets [1] that are available on the Web change over time [2]. Many of these dataset publishers host separate snapshots of each of these versions, which can introduce storage overhead due to redundancies between them. Furthermore, separate snapshots can make it harder to execute queries for performing historical analyses.

RDF [3] provides a framework for representing Linked Data. Over the last couple of years, RDF archiving has been an active area of research [4][5][6][7][8]. Fernández et al. define an *RDF archive* [9] as a set of version-annotated triples, where a *version-annotated triple* is an RDF triple that is annotated with a label representing the version in which this triple holds. Three strategies [10] were identified on how RDF archives can be stored:

1. The **Independent Copies (IC)** approach creates separate instantiations of datasets for each change or set of changes.
2. The **Change-Based (CB)** approach instead only stores change sets between versions.
3. The **Timestamp-Based (TB)** approach stores the temporal validity of facts.

Additionally, several query types were introduced [9] to cover the retrieval demands in RDF archives, which are referred to as *query atoms*. In this work, we consider the following query atoms:

1. **Version materialization (VM)** retrieves data using queries targeted at a single version. Example: *Which books were present in the library yesterday?*
2. **Delta materialization (DM)** retrieves query result change sets between two versions. Example: *Which books were returned or taken from the library between yesterday and now?*
3. **Version query (VQ)** annotates query results with the versions in which they are valid. Example: *At what times was book X present in the library?*

Each of these storage strategies have their advantages and disadvantages in combination with certain query atoms. For instance, IC works well in combination with VM queries because it stores each version separately, so it can query each version separately as well. However, IC is less efficient for DM queries because it requires the differences between two dataset versions for the given query to be generated on-the-fly. Hybrid storage strategies, such as applied by TailR [11], can however provide different trade-offs between these strategies.

In this work, we describe and demonstrate *OSTRICH*, which is a hybrid IC-CB-TB storage technique, that offers efficient VM, DM and VQ triple pattern query support. This system is further discussed in Section 2, together with a preliminary evaluation in Section 3. After that, we give an overview of a demonstration of OSTRICH using a Web-based graphical user interface in Section 4. Finally, we discuss our conclusions and opportunities for future work in Section 5.

2 OVERVIEW OF THE OSTRICH SYSTEM

In this section, we give a brief overview of OSTRICH, the system on which this demonstration is built.

OSTRICH uses a *versioned triple store* format that allows VM, DM and VQ triple pattern queries to be resolved efficiently. Furthermore, these queries return a triple stream—triples can be consumed as they arrive, which supports efficient offsets. As certain systems, such as SPARQL query engines, typically optimize triple pattern join orders using estimated triple counts, OSTRICH provides efficient count estimation for VM, DM and VQ queries. Triple pattern queries, together with count estimation, form the basis for more sophisticated RDF/SPARQL query engines, such as the client-side Triple Pattern Fragments engine [12].

Internally, OSTRICH stores a versioned dataset in a *hybrid IC-CB-TB* way, using multiple indexes for supporting the different query types. The initial version of a dataset is stored as a fully materialized and immutable snapshot. This snapshot is stored as an HDT [13] file, which is a highly compressed, binary RDF representation. HDT also provides indexes that enable the efficient execution of triple pattern queries and count estimation. All other versions are *changesets*, i.e., lists of triples that need to be removed and lists of triples that need to be added. Changesets are stored in a custom indexing structure. These changesets are relative to the initial version, but merged in a timestamp-based manner to reduce redundancies between each version.

OSTRICH is implemented in C++, and is available as open source on GitHub *(https://github.com/rdfostrich/ostrich)*. Additionally, JavaScript bindings for Node.js have been implemented and are available on NPM *(https://www.npmjs.com/package/ostrich-bindings)*. These JavaScript bindings however lead to slightly slower

ACM 978-1-4503-5640-4/18/04.
DOI: http://dx.doi.org/10.1145/3184558.3186960

queries compared to the native C++ API. The C++ and JavaScript APIs allow OSTRICH stores to be queried using VM, DM and VQ triple pattern queries with a certain limit and offset. Additionally, their count estimates can be retrieved. Finally, new dataset versions can be ingested as changesets.

3 PRELIMINARY EVALUATION

For our preliminary evaluation, we have used the highly volatile BEAR-B-hourly dataset from the BEAR benchmark [9], which consists of the 100 most volatile resources from DBpedia Live [14]. This dataset contains 48.000 unique triples over 1.299 versions, and requires 8,314.86 MB when stored as N-Triples in changesets (466.35 MB gzipped). Fig. 1 shows the growth of an OSTRICH store after the ingestion of each consecutive version of this dataset. Using OSTRICH, this dataset requires only 450.59 MB to be stored, or 187.46 MB without the optimizing indexes. Compared to other systems in the BEAR benchmark, this is on average only 5,2% of IC strategies, 4,8% of TB strategies, but 514% of CB stategies. Furthermore, a less volatile dataset with an average of 17M triples over 10 versions requires 4.48 GB of storage with OSTRICH, and 3.03 GB if only the essential querying indexes are enabled. With OSTRICH, this dataset takes on average 35% of IC strategies, 10% of TB strategies, and 66% of CB stategies. For the tested datasets, OSTRICH requires significantly less storage space than IC and TB strategies. For datasets with a low volatility, OSTRICH requires less storage space than CB strategies. For highly volatile datasets, it requires more storage space, which is because OSTRICH enables more efficient version materialization than these CB strategies, and this comes at the cost of more required storage.

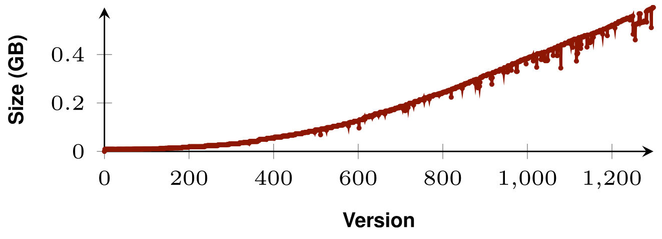


Fig. 1: Cumulative OSTRICH store sizes for each consecutive BEAR-B-hourly version in GB for an increasing number of versions.

Fig. 2, 3 and 4 show the average query evaluation times for VM, DM and VQ triple pattern queries on this BEAR-B-hourly dataset. In these figures, the evaluation times of OSTRICH are compared with versioning systems implemented based on HDT [13] and Jena [15], as provided by the BEAR benchmark. At the cost of ingestion times that are 1,38 to 125 times higher than in alternative solutions, OSTRICH is able to significantly reduce query times for VM, DM and VQ triple pattern queries. Results have shown that the average query times range between 0.1 and 1 milliseconds, which is lower than most alternative solutions. Only for VM queries on individual HDT copies, OSTRICH is slightly slower. This is because these HDT file are optimized for querying within each specific version, while OSTRICH chooses a different trade-off: storage space is significantly reduced and this makes VM queries only slightly slower compared to individual HDT copies. Additionally, this also makes DM and VQ queries within OSTRICH faster than with individual HDT copies, which makes OSTRICH a general-purpose versioned querying solution.

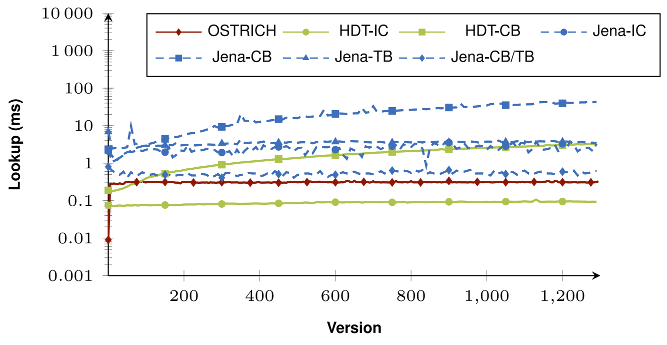


Fig. 2: Median BEAR-B-hourly VM query results for all triple patterns for all versions.

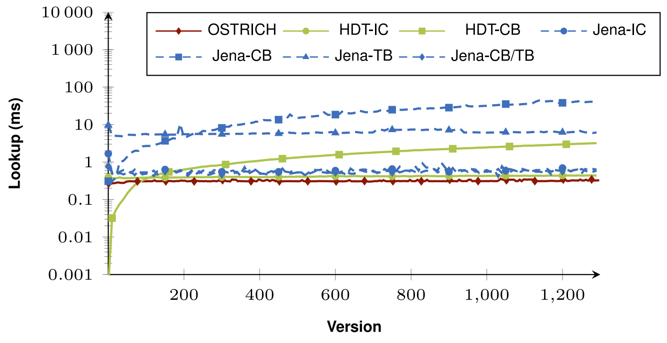


Fig. 3: Median BEAR-B-hourly DM query results for all triple patterns from version 0 to all other versions.

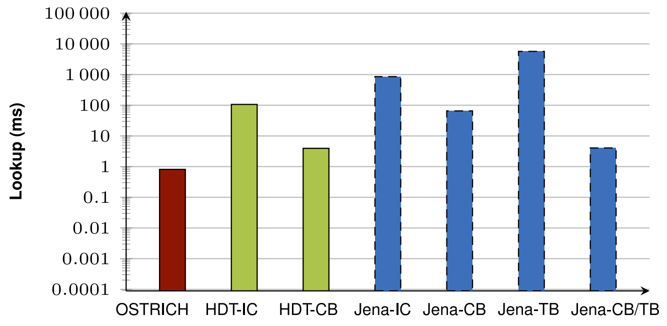


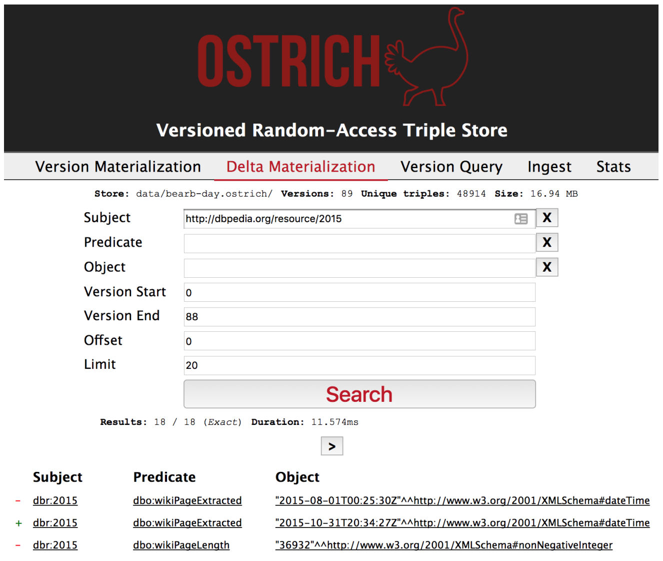
Fig. 4: Median BEAR-B-hourly VQ query results for all triple patterns.

4 DEMONSTRATION OVERVIEW

The goal of this demonstration is to show the capabilities of OSTRICH. This is done using a Web application (*OSTRICH Admin*) in which an OSTRICH store can be created, viewed, and updated. When starting the application, the path to a—possibly empty—OSTRICH store must be provided. This application has several features, including the ability to perform VM, DM and VQ queries, ingest new versions, and retrieve statistics about the store. These features will be elaborated on in the next sections. Finally, we introduce two example datasets to discover the interface.
OSTRICH Admin is implemented as a Node.js Web application using the Express framework *(https://expressjs.com/)*. This was done using the OSTRICH JavaScript bindings for Node.js. This application is available on GitHub *(https://github.com/rdfostrich/ostrich-admin)* under an open license. A screencast demonstrating the usage of this application can be found on Vimeo *(https://vimeo.com/246792247)*.

4.1 Query

OSTRICH Admin supports visual VM, DM and VQ triple pattern queries. These are usable by respectively following the *Version Materialization*, *Delta Materialization* or *Version Query* links as can be seen in Fig. 5. These pages show a form that corresponds to the OSTRICH API for these query types.
For instance, Fig. 5 shows the form for DM queries. The subject, predicate and object fields are used to provide URIs, literals or variables for the triple pattern query. A start and end version can be selected, which will define the versions over which the delta will be retrieved. Additionally, offset and limit values can be applied to the triple results.
Below the form, the triples matching the defined query are shown. In the case of DM queries, triples are annotated with a "+" or "-", which indicates if they are respectively an addition or deletion with respect to the given changeset. Furthermore, the number of results on this page is shown, together with the total count of this query, independent of the limit and offset. This total count can either be an *exact value*, or an *estimate* if calculating the exact value would take too much time. Finally, the triple pattern query execution time is shown.



Store: data/bearb-day.ostrich/ **Versions:** 89 **Unique triples:** 48914 **Size:** 16.94 MB

Subject	http://dbpedia.org/resource/2015
Predicate	
Object	
Version Start	0
Version End	88
Offset	0
Limit	20

Results: 18 / 18 (*Exact*) **Duration:** 11.574ms

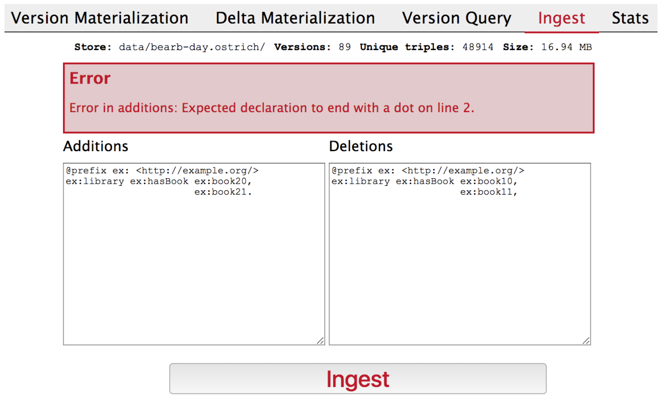
	Subject	Predicate	Object
–	dbr:2015	dbo:wikiPageExtracted	"2015-08-01T00:25:30Z"^^http://www.w3.org/2001/XMLSchema#dateTime
+	dbr:2015	dbo:wikiPageExtracted	"2015-10-31T20:34:27Z"^^http://www.w3.org/2001/XMLSchema#dateTime
–	dbr:2015	dbo:wikiPageLength	"36932"^^http://www.w3.org/2001/XMLSchema#nonNegativeInteger

Fig. 5: Delta Materialization interface for querying the differences between two versions by triple pattern with a certain offset and limit. The page shows all matching triples annotated with either the addition (+) or deletion symbol (–). Additionally, the total number of results and the query duration time is shown.

Similar pages exist for VM and VQ queries. For VM queries, the form does not have a version range, but only a single version field. For VQ queries, the form has no version fields, but results are annotated with version ranges.

4.2 Ingest

As ingesting new versions is an important feature in archiving solutions, OSTRICH Admin allows changeset-based version ingestion as can be seen in Fig. 6.
This form has a textbox for additions and deletions. This corresponds to the way the OSTRICH API accepts version ingestion, which is done using a stream containing additions and deletions. These textboxes accept triples in the Turtle serialization.
When ingestion is successful, the number of inserted triples will be displayed, together with the time it took to insert them.



Store: data/bearb-day.ostrich/ **Versions:** 89 **Unique triples:** 48914 **Size:** 16.94 MB

Error

Error in additions: Expected declaration to end with a dot on line 2.

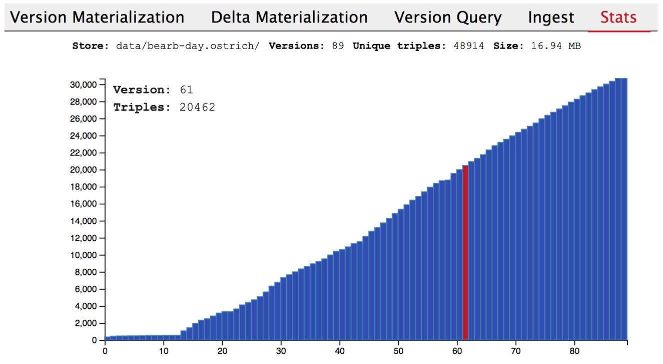
```
@prefix ex: <http://example.org/>
ex:library ex:hasBook ex:book20,
                      ex:book21.
```

```
@prefix ex: <http://example.org/>
ex:library ex:hasBook ex:book10,
                      ex:book11,
```

Fig. 6: Ingesting a new version is done using a changeset form for additions and deletions. The form accepts triples in turtle format, and will give user feedback in case invalid triples were provided.

4.3 Statistics

Finally, OSTRICH Admin allows basic statistics about the current OSTRICH store to be displayed as shown in Fig. 7. These statistics can be used for gaining a basic understanding of the dataset size and its growth rate.
On the top of this page—and all other pages as well—the path to the currently opened store is shown. Next to that, the total number of versions in this store is displayed. Finally, the total number of unique triples in this store, and the total size of this store is shown.
Additionally, this page shows a graph containing the number of triples in each version of the dataset. This graph is interactive, and allows the user to hover over each version bar to show the exact number of triples in the selected version.



Store: data/bearb-day.ostrich/ **Versions:** 89 **Unique triples:** 48914 **Size:** 16.94 MB

Version: 61
Triples: 20462

Fig. 7: The stats page shows an overview of the number of triples in each version. The user can hover over the version bars to see the exact number of triples in the top-left side of the graph.

4.4 Example Datasets

For our demonstration, we provide two example datasets that can be used to load into OSTRICH admin. All of these examples are publicly available *(https://linkedsoftwaredependencies.org/raw/ostrich/datasets/)*.

4.4.1 *Library*

The first dataset is a very small synthetic dataset about the availability of books in a library. The goal of this dataset is to explain the concept of changesets using books that are only available at specific moments in the library. In order to make it easily understandable, this dataset has only four versions, and 11 books that are available in specific versions of the dataset.

4.4.2 *DBpedia Live*

A larger real-world dataset based on DBpedia Live [14] contains more than 48K unique triples over 89 versions. This dataset has been derived from the BEAR RDF archiving benchmark [9]. It contains the 100 most volatile resources from DBpedia Live over the course of three months with an hourly granularity.

5 CONCLUSIONS

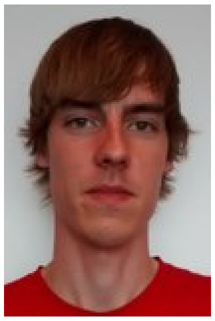
RDF archiving has been an active area of research over the last couple of years. OSTRICH is storage and querying system for RDF archives that supports various kinds of versioned queries. With OSTRICH, versioned datasets can be stored efficiently, while at the same time enabling efficient support for versioned queries. When OSTRICH is combined with techniques such as Triple Pattern Fragments, versioned Linked Datasets can be published at a low cost, and complex SPARQL queries can be evaluated *in*, *between*, and *over* the different versions. This lower the barrier towards historical analysis over datasets that evolve over time, such as biomedical patient information or certain taxonomies.

In the future, we will continue improving the performance of OSTRICH, and do an extensive performance evaluation. OSTRICH Admin will be kept up-to-date with the functionality of OSTRICH, so that OSTRICH datasets can be discovered and managed at a high-level using this Web application, without having to use the programmatic API for this.

ACKNOWLEDGEMENTS

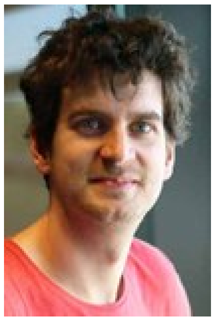
The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), and the European Union. Ruben Verborgh is a postdoctoral fellow of the Research Foundation – Flanders.

BIOGRAPHIES



Ruben Taelman is a PhD student at IDLab, Ghent University – imec, Belgium. His research concerns the server and client trade-offs for Linked Data publication and querying, with a particular focus on dynamic data, such as streams and versioning.

Miel Vander Sande is a post-doctoral researcher in Linked Data at Ghent University – imec. His main interest is low-cost Linked Data publishing infrastructures and Intelligent Web clients. In that regard, he executed numerous projects in Open Data legislation, digital publishing, e-learning, and data sharing.



Ruben Verborgh is a professor of Semantic Web technology at Ghent University – imec and a postdoctoral fellow of the Research Foundation Flanders. He explores the connection between Semantic Web technologies and the Web's architectural properties, with the ultimate goal of building more intelligent clients. Along the way, he became fascinated by Linked Data, REST/hypermedia, Web APIs, and related technologies.

REFERENCES

[1] Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts. 205–227 (2009).
[2] Umbrich, J., Decker, S., Hausenblas, M., Polleres, A., Hogan, A.: Towards dataset dynamics: Change frequency of Linked Open Data sources. 3rd International Workshop on Linked Data on the Web (LDOW). (2010).
[3] Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: Concepts and Abstract Syntax. W3C, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (2014).
[4] Vander Sande, M., Colpaert, P., Verborgh, R., Coppens, S., Mannens, E., Walle, R. Van de: R&Wbase: git for triples. In: Proceedings of the 6th Workshop on Linked Data on the Web (2013).
[5] Neumann, T., Weikum, G.: x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. Proceedings of the VLDB Endowment. 3, 256–263 (2010).
[6] Volkel, M., Winkler, W., Sure, Y., Kruk, S.R., Synak, M.: Semversion: A versioning system for RDF and ontologies. In: Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29–June 1, 2005. Proceedings (2005).
[7] Cerdeira-Pena, A., Farina, A., Fernández, J.D., Martínez-Prieto Miguel A: Self-indexing RDF archives. In: Data Compression Conference (DCC), 2016. pp. 526–535. IEEE (2016).
[8] Taelman, R., Vander Sande, M., Verborgh, R., Mannens, E.: Versioned Triple Pattern Fragments: A Low-cost Linked Data Interface Feature for Web Archives. In: Proceedings of the 3rd Workshop on Managing the Evolution and Preservation of the Data Web (2017).
[9] Fernández, J.D., Umbrich, J., Polleres, A., Knuth, M.: Evaluating Query and Storage Strategies for RDF Archives. In: Proceedings of the 12th International Conference on Semantic Systems. ACM (2016).
[10] Fernández, J.D., Polleres, A., Umbrich, J.: Towards efficient archiving of Dynamic Linked Open Data. In: Debattista, J., d'Aquin, M., and Lange, C. (eds.) Proceedings of te First DIACHRON Workshop on Managing the Evolution and Preservation of the Data Web. pp. 34–49 (2015).
[11] Meinhardt, P., Knuth, M., Sack, H.: TailR: a platform for preserving history on the web of data. In: Proceedings of the 11th International Conference on Semantic Systems. pp. 57–64. ACM (2015).
[12] Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web. Journal of Web Semantics. 37–38, (2016).
[13] Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF Representation for Publication and Exchange (HDT). Web Semantics: Science, Services and Agents on the World Wide Web. 19, 22–41 (2013).
[14] Morsey, M., Lehmann, J., Auer, S., Stadler, C., Hellmann, S.: DBpedia and the live extraction of structured data from wikipedia. Program. 46, 157–181 (2012).
[15] McBride, B.: Jena: A semantic web toolkit. IEEE Internet computing. 6, 55–59 (2002).