Continuous Knowledge Graph Quality Assessment through Comparison using ABECTO

Jan Martin Keil

Heinz Nixdorf Chair for Distributed Information Systems, Institute for Computer Science, Friedrich Schiller University Jena, Germany

German Aerospace Center (DLR), Institute of Data Science, Mälzerstraße 3-5, 07745 Jena, Germany

Abstract

Maintaining accuracy and completeness of RDF knowledge graphs is an important but challenging task. As constraints checking can only spot outliers, knowledge graphs would need to be checked against reference data to obtain a reliable assessment. But this is often impossible due to the lack of suitable reference data. An alternative is the comparison with other, overlapping knowledge graphs that possibly contain incorrect data, too. This can spot potentially incorrect values in the maintained knowledge graph. As knowledge graphs might evolve over time, this comparison must be done regularly. However, the regular comparison of data is an exhausting and error prone task, if done manually.

We present ABECTO, a command line tool for the automatic comparison of multiple possibly incorrect RDF knowledge graphs to monitor their accuracy and completeness. In our demonstration, we will showcase its application in a Continuous Integration scenario for the regular automatic check of the quality of a maintained knowledge graph.

Keywords

Continuous Integration, Knowledge Graph Engineering, Knowledge Graph Quality, Ontology Engineering, Ontology Quality

1. Introduction

In more and more application areas, knowledge graphs (KGs) provide domain-specific background information for applications. These applications rely on the quality of the provided data. Accuracy and completeness are two important quality criteria for the maturity of KGs for these applications. However, maintaining the accuracy and completeness of RDF KGs is a challenging task. State-of-the-art approaches, like the Shapes Constraint Language (SHACL) [1] for A-Box statements or the OntOlogy Pitfall Scanner! (OOPS!) [2] for T-Box statements, use constraints to spot incompleteness or wrong data. However, these approaches can only detect data outside of the range of plausible values, as for example a negative age or a missing birthdate of a person. They fail to detect plausible but still wrong values or completely missing resources. The detection of these issues requires the comparison with reference data. Unfortunately, a reliable reference dataset is typically not available. Otherwise, the reference dataset should have been used to construct the KG in the first place. Moreover, even authoritative datasets might contain errors, as their creation process can not rely on a reliable reference dataset, either. This results in a classical chicken-egg problem. In consequence, methods to detect wrong or incomplete data must not rely on the freedom from error of other data.

As an alternative, we proposed [3] the comparison with other, overlapping KGs that possibly contain incorrect data, too. This approach can spot potentially incorrect and missing values as well as missing resources in the maintained KG. Even if it might not find all issues due to incorrect and incomplete data in the compared KGs, this can contribute to further improve the accuracy and completeness of the maintained KG. Thereby, it complements constraint based methods. Further, not requiring an ideal reference dataset makes it more likely to find data to compare against.

SEMANTiCS 2024 EU: 20th International Conference on Semantic Systems, September 17-19, 2024, Amsterdam, Netherlands

jan_martin.keil@dlr.de (J. M. Keil)

0000-0002-7733-0193 (J. M. Keil)

If done manually, the comparison of data in KGs is an exhausting and error prone task. Therefore, the comparison must be automated. Moreover, to unlock the full potential of automated quality checks for the improvement of a KG, they must run regularly. Otherwise, the data quality could decrease during the evolution of the KG and incorrect data could be used in a productive environment, even though they would have been automatically detectable.

In software engineering, the practice of quick integration and regular automatic quality checking of all changes to a software is called Continuous Integration (CI). Ideally, each proposed change to the software by internal and external contributors is immediately checked automatically. From the field of software engineering it is known that projects using CI have an increased number of defects spotted internally without having an increasing number of defects spotted externally [4] and release their software more often [5]. CI is not yet widely used in knowledge graph engineering. However, we expect positive effects on productivity and quality in this field, too.

Therefore, we developed ABECTO, a command line tool for the comparison of multiple possibly incorrect RDF KGs to monitor their accuracy and completeness. It enables KG maintainers to regularly check their KG automatically and is designed to be used in a CI environment. With ABECTO, we extend the range of tools for the automated quality assessment of KGs to enable the precise orchestration of quality assurance processes for KGs. In our demonstration, we will showcase its application in a CI scenario for the regular automatic check of the quality of the maintained KG.

2. Comparison Framework

For the realization of the comparison of knowledge graphs, we propose a framework [3, 6] of four consecutive processing phases and a reporting phase. The sequence of phases is visualized in Figure 1.

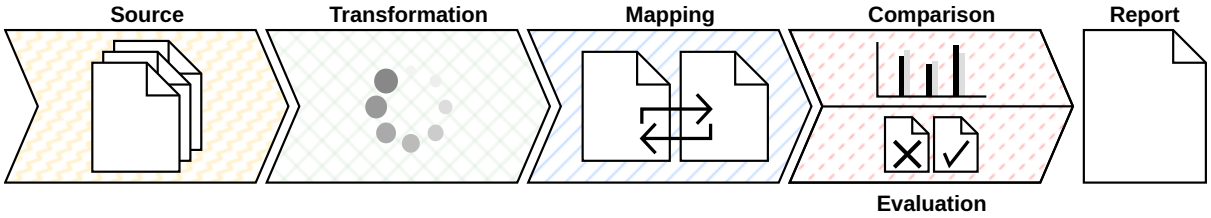

Figure 1: Schematic of the knowledge graph comparison framework.

During the **source phase**, the data of the knowledge graphs to compare get loaded. The data might be loaded from local files, files retrieved via URL, SPARQL endpoints or further sources.

During the **transformation phase**, additional statements can be generated based on the existing statements. Tools like SPARQL construct queries, reasoners or custom sets of rules might get applied for this. The transformation enables, for example, the deduction of implicit encoded data, the application of domain knowledge not encoded in the knowledge graph or the alignment of literal datatypes. It provides high flexibility with regards to the original structure of the data to enable the applicability of knowledge graph comparisons to a wide range of fields.

During the **mapping phase**, the correspondency of resources gets determined. A wide range of approaches could be applied for the mapping: Starting from simple rules like equal or similar values of corresponding properties up to the application of sophisticated algorithms for ontology matching or entity linking. In addition, users might manually include or exclude correspondencies. However, mapping approaches are not the main concern of our work. Instead, results from dedicated third mapping tools could also be reused, if provided as RDF input.

During the **comparison and evaluation phase**, the corresponding resources get compared. This includes the determination of missing resources or values, as well as the detection of value deviations. Further, the count, coverage and completeness of resources and values will be measured.

During the **reporting phase**, the collected information about the knowledge graphs get exported into files of different formats.

3. The ABox Evaluation and Comparison Tool for Ontologies (ABECTO)

ABECTO is a Java command line tool for the comparison and evaluation of two or more KGs to assess their accuracy and completeness. It implements the comparison framework introduced in Section 2 and is based on the RDF framework Apache Jena[1]. The source code is publicly available on GitHub[2] and Zenodo [7] under the permissive open source software license Apache 2.0[3]. In addition, ABECTO is available as a Docker image in the GitHub Container Registry to ease the use of ABECTO in CI environments. Parameters may enable a failure exit status code in case of detected issues in a specific KG to signal a failure to the calling environment.

ABECTO is developed as a command line tool because programs with graphical user interfaces cannot be used effectively in an automated manner. In early drafts [6] ABECTO was implemented as HTTP REST service with Jupyter Notebook[4] as user interface. However, this prevented it from being used as a CI tool and therefore the architecture was refactored. Beyond that, the current version features several new or improved processors, reports and measures.[5]

3.1. Configuration and Application of ABECTO

The comparison in ABECTO is configured by a *plan* that is a pipeline of several *steps*, described using the ABECTO vocabulary[6]. Each step executes a *processor* with a particular configuration. ABECTO provides multiple built in processors to (a) load resources from different *source*s like files or SPARQL endpoints, (b) *transform* statements with e.g. SPARQL CONSTRUCT queries in preparation of future steps, (c) *map* resources based on value equality, string similarity, functional dependencies and the re-use of mappings provided by the KGs them self, and to (d) *compare* corresponding resources regarding population completeness and property value accuracy and completeness. The data to compare are described with *aspects* and aspect *variables* defined by one SPARQL SELECT query per compared KG. The output primary data and metadata of each step are stored in in-memory RDF graphs, using the ABECTO vocabulary.

ABECTO can reveal several types of issues: *Resource omissions* spot resources from one KG without corresponding resource in another KG. *Value omissions* spot variable values of resources from one KG without an equivalent value for corresponding resources and the same variable in another KG. However, if both KGs have a value for the same variable of corresponding resources, but their values are not equivalent, ABECTO detects a *value deviation*. Further, ABECTO can provide measurements of the number of resources and values, the overlap between the KGs, and their estimated completeness.

The results together with the plan definition and provenance data can be stored as RDF in a TRiG[7] file. Further, ABECTO provides multiple built in reports for the export in CSV or Markdown[8] format to ease the review and analysis of measurements, omissions, deviations and mappings.

The command line interface of ABECTO provides several parameters to control its input and output. An RDF dataset file that contains a plan configuration must be selected as input. If paths for an RDF result file or reports are provided, these will we generated. The RDF result file also contains the plan configuration and can be used as input, too. A parameter allows to skip the plan execution to only load and process existing results for later generation of additional reports. For the use in CI environments the KG of primary interest can be defined. Then, the reports will only contain issues that concern the KG of primary interest. Moreover, it can be configured which type of issues –if applicable, only from the KG of primary interest– cause a failure exit status code to signal a problem to the CI environment.

[1] https://jena.apache.org/
[2] https://github.com/fusion-jena/abecto
[3] https://opensource.org/licenses/Apache-2.0
[4] https://jupyter.org/
[5] https://github.com/fusion-jena/abecto/blob/main/CHANGELOG.md
[6] http://w3id.org/abecto/vocabulary
[7] https://www.w3.org/TR/trig/
[8] https://daringfireball.net/projects/markdown/

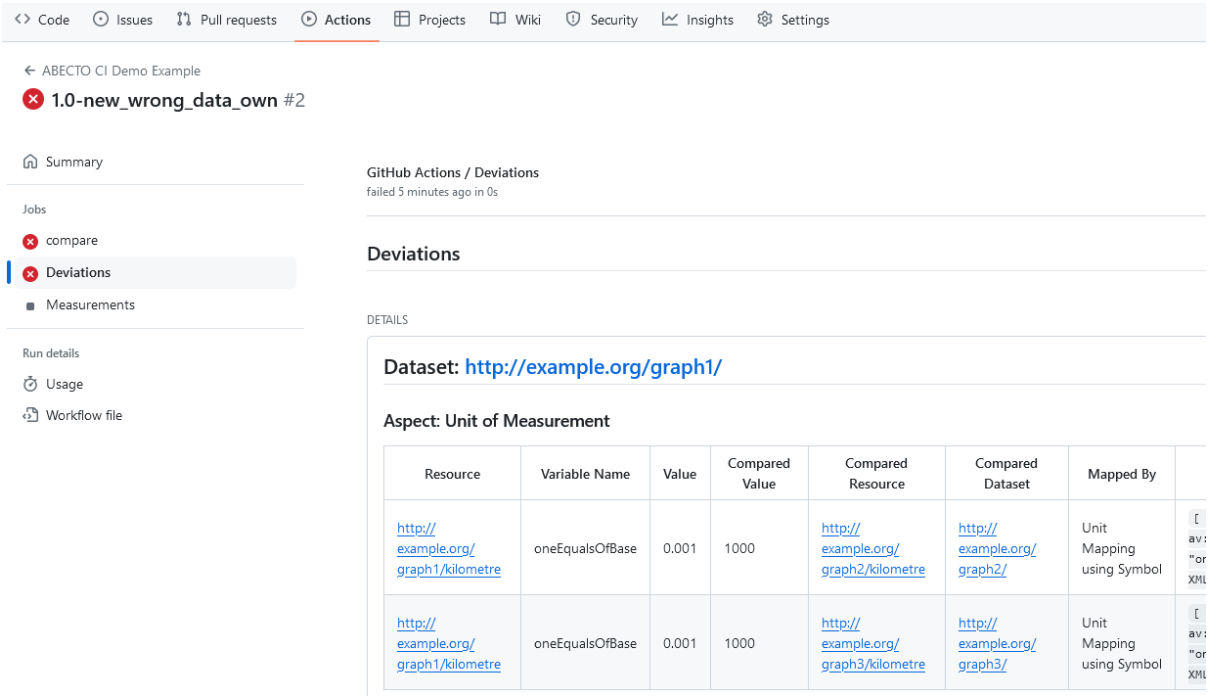

Figure 2: Screenshot of the presentation of deviations in the GitHub user interface. It shows the deviations between the value of one resource from the maintained KG and the values of two corresponding resources from two different compared KGs.

3.2. Example Projects on Real-World Knowledge Graphs

We provide two example comparison projects for real world KGs.[9] The first project is a **Comparison and Evaluation of Unit Ontologies** [8]. In the project we compare *unit of measurement* and *quantity kind* data from OM 2, QUDT 2, as well as the according subsets of SWEET 3 and Wikidata. The comparison includes in total more than 11 600 resources and takes in total about 13 min. Out of these, 5 min are spend on the loading of all relevant data from the Wikidata SPARQL endpoint, alone.

The second project is a **Comparison of Space Travel Data in Wikidata and DBpedia** [9]. In the project we compare *astronaut*, *spacecraft* and *space mission* data from DBpedia and Wikidata. The comparison includes in total more than 18 200 resources and takes in total about 14 min. Again, 5 min of this time are spend on loading all relevant data from the Wikidata SPARQL endpoint, alone.

Findings have been reported to the according maintainers. Both comparison projects already caused several improvements[10] in all involved knowledge graphs.

4. Demonstration

We showcase the use of ABECTO in a CI pipeline to monitor the accuracy of a KG. The demonstration [10] starts with an incomplete and flawed example KG maintained in a Git repository. The repository is equipped with a ready for operation pipeline configuration that will compare the example KG to two other incomplete and flawed example KGs. For the purpose of the demonstration the KGs in comparison will also be contained in the repository. In a real world scenario they would of course origin from external sources.

Requirements for the demo execution and preparation instructions are available in the ReadMe

[9] DBpedia: https://www.dbpedia.org/, OM: https://github.com/HajoRijgersberg/OM, QUDT: https://qudt.org, SWEET: https://github.com/ESIPFed/sweet, Wikidata: https://www.wikidata.org

[10] OM: https://github.com/HajoRijgersberg/OM/issues?q=abecto, QUDT: https://github.com/qudt/qudt-public-repo/issues?q=abecto, SWEET: https://github.com/ESIPFed/sweet/issues?q=abecto, DBpedia & Wikidata: https://www.wikidata.org/wiki/User:Jmkeil/ABECTO_Provoked_Edits#Based_on_the_Comparison_of_Unit_Ontologies

file[11] of the demonstration. During the demonstration, the initial KGs will be updated with some prepared changes. These changes might (a) add new accurate facts, (b) add new faulty facts, (c) fix existing faulty facts, (d) update the KGs in comparison in a way affecting the comparison, (e) mark deviating facts from the KGs in comparison as wrong, or (f) restore the initial repository status to restart the demonstration. After each change, the CI pipeline will be executed automatically. An example repository with pre-executed changes[12] is available online. The results become available through the GitHub user interfaces and present the detected problems in the KG to the user, as shown in Figure 2. During the demonstration, we will look up the results of the automated execution of ABECTO after each change. This allows visitors to experience hands-on how KG quality can be monitored and improved using ABECTO. Moreover, since ABECTO is available as free and open source software, visitors have the opportunity to continue exploring the capabilities afterwards and to use it for their own KG.

Acknowledgments

Many thanks to the author's supervisor Birgitta König-Ries as well as to the anonymous reviewers for very helpful comments on earlier drafts of this manuscript.

References

[1] RDF Data Shapes Working Group, Shapes Constraint Language (SHACL), in: H. Knublauch, D. Kontokostas (Eds.), W3C Recommendation, 2017. URL: https://www.w3.org/TR/2017/REC-shacl-20170720/.

[2] M. Poveda Villalón, A. Gómez Pérez, M. C. Suárez Figueroa, OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation, International Journal on Semantic Web and Information Systems 10 (2014) 7–34. doi:10.4018/ijswis.2014040102.

[3] J. M. Keil, Ontology ABox Comparison, in: The Semantic Web: ESWC 2018 Satellite Events - ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers, 2018, pp. 240–250. doi:10.1007/978-3-319-98192-5_43.

[4] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, V. Filkov, Quality and productivity outcomes relating to continuous integration in github, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, ACM, New York, NY, USA, 2015, p. 805–816. doi:10.1145/2786805.2786850.

[5] M. Hilton, T. Tunnell, K. Huang, D. Marinov, D. Dig, Usage, costs, and benefits of continuous integration in open-source projects, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE '16, ACM, New York, NY, USA, 2016, p. 426–437. doi:10.1145/2970276.2970358.

[6] J. M. Keil, Abecto: An abox evaluation and comparison tool for ontologies, in: A. Harth, V. Presutti, R. Troncy, M. Acosta, A. Polleres, J. D. Fernández, J. X. Parreira, O. Hartig, K. Hose, M. Cochez (Eds.), The Semantic Web: ESWC 2020 Satellite Events, volume 12124 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 140–145. doi:10.1007/978-3-030-62327-2_24.

[7] J. M. Keil, ABox Evaluation and Comparison Tool for Ontologies (ABECTO) v2.2.2 (2024). doi:10.5281/zenodo.11522479.

[8] J. M. Keil, Units of Measurement Data Comparison with ABECTO, 2023. doi:10.5281/zenodo.7843835.

[9] J. M. Keil, Wikidata and DBpedia Space Travel Data Comparison with ABECTO, 2023. doi:10.5281/zenodo.7843823.

[10] J. M. Keil, Demonstration: Continuous Knowledge Graph Quality Assessment through Comparison using ABECTO v1.0.0 (2024). doi:10.5281/zenodo.10796062.

[11] https://github.com/fusion-jena/abecto-ci-demo/blob/main/README.md
[12] https://github.com/fusion-jena/abecto-ci-demo-example/actions