ABECTO: Assessing Accuracy and Completeness of RDF Knowledge Graphs

Jan Martin Keil

Heinz Nixdorf Chair for Distributed Information Systems,
Institute for Computer Science, Friedrich Schiller University Jena, Germany
E-mail: jan-martin.keil@uni-jena.de; ORCID: https://orcid.org/0000-0002-7733-0193

Abstract. Accuracy and completeness of RDF knowledge graphs are crucial quality criteria for their fitness for use. However, assessing accuracy and completeness of knowledge graphs requires a basis for comparison. Unfortunately, in general, a gold standard to compare with does not exist. As an alternative, we propose the comparison with other, overlapping RDF knowledge graphs of arbitrary quality. We present ABECTO, a command line tool that implements a pipeline based framework for the comparison of multiple RDF knowledge graphs. For these knowledge graphs, it provides quality annotations like value deviations and quality measurements like completeness. This enables knowledge graph curators to monitor the quality and potential users to select an appropriated knowledge graph for their purpose. With two example applications of ABECTO we demonstrate the usefulness of ABECTO for the improvement of knowledge graphs.

Keywords: Data Quality, Knowledge Graph Evaluation, Knowledge Graph Quality, Ontology Evaluation, Ontology Quality

1. Motivation

Accuracy and completeness of knowledge graphs are important criteria in the selection or evaluation of a knowledge graph for an application. With *knowledge graph* we denote in this paper any kind of RDF dataset, typically but not necessarily with a focus on actual instances (A-Box) instead of schema definitions (T-Box), and including RDF datasets that are typically denoted as *ontology*. Assessing accuracy and completeness of knowledge graphs requires data to compare with. Constraints on property values can only detect outliers, but can not detect wrong values that are in a plausible range. The literature regularly recommends to use a gold standard for the evaluation [1]. However, a gold standard does in general not exist. An alternative is the comparison with other, overlapping knowledge graphs of arbitrary quality [2].

In an earlier comparison of unit ontologies [3] we experienced the need for automation of the comparison of knowledge graphs. However, the automation scripts for that specific comparison were not generally applicable. Therefore, we outlined a comparison framework [2] and developed the ABox Evaluation and Comparison Tool for Ontologies (ABECTO) [4], a generic comparison tool for knowledge graphs.

ABECTO provides a pipeline based framework for the comparison of multiple knowledge graphs. It enables their curators to monitor the quality or users to select an appropriated knowledge graph to use. In this paper, we introduce the tool and two example applications. The paper is structured as follows: In Section 2 we give an overview about related work, followed by a collection of requirements for the tool in Section 3. ABECTO is introduced in Section 4 and in Section 5 we name limitations of the tool. The intended workflow with ABECTO is introduced in Section 6. In Section 7 we present two example applications and then close with our conclusion in Section 8.

2. Related Work

The term *comparison* is ambiguously used in the context of semantic web technologies [2]. We will use the term to describe the comparison of entire knowledge graphs regarding certain aspects to evaluate or select knowledge graphs.

Wang and Strong define the quality dimension *accuracy* as "the extent to which data are correct, reliable, and certified free of error" [5]. In context of knowledge graphs, this can be divide into (a) syntactic validity of RDF documents and of literals, which is the compliance to syntactic rules, and (b) semantic validity of triples, which determines whether the value is true [1]. In this paper, we focus on semantic validity.

The quality dimension *completeness* is defined by Wang and Strong as "the extent to which data are of sufficient breadth, depth, and scope for the task at hand" [5]. For knowledge graphs, this can be divided into (a) schema completeness, which assesses missing classes and properties, (b) column completeness, which assesses missing property values, and (c) population completeness, which assesses missing resources. In this paper, we focus on column completeness and population completeness.

The quality dimensions accuracy and completeness have in common that their assessments requires a basis for comparison [1]. This basis for comparison is typically called a gold standard. However, the general lack of a gold standard for the assessment of accuracy and completeness of knowledge graphs was constantly considered as a problem since early work [6] on ontology evaluation until more recent work [1]. Nevertheless, only a few tools have recently been developed to make use of the comparison of knowledge graphs with each other to address this issue. All of these tools are dedicated to specific knowledge graphs:

The library authority data initiatives International Standard Name Identifier (ISNI)[1] and Virtual International Authority File (VIAF)[2] regularly run processes for integration and consolidation of the data of their providers and among each other [7]. Manual result review processes then reveal errors in the source data that get communicated to the providers to enable error correction.

Bianchini et al. compared data from Wikidata[3] and VIAF regarding the contained data about persons [8]. Among others, they measured the reciprocal coverage to assess the suitability for entity identification purposes. Further, they provide several exemplary cases in which the interlinking reveals errors in both datasets. They conclude, that "VIAF and Wikidata can be constantly improved through reciprocal comparison, which allows discovery of errors in both".

The machine learning based tool *soweego*[4] enables the supervised linking of Wikidata items to items from some other catalogs. The linked items can get compared to propose changes for Wikidata.

The web service *Wikidata Mismatch Finder*[5] provides a platform for reporting deviations between Wikidata and other data sources. The goal is to provide a convenient way to report and collaboratively handle possible errors in Wikidata and interlinked data sources.

3. Requirements

During an earlier comparison of unit ontologies [3] we experienced a couple of challenges that led to some requirements for a generic comparison tool which we considered during the development of ABECTO:

R1. Handling of Schema Heterogeneity: Different knowledge graphs might use heterogeneous modeling approaches for the same aspect of a domain. For example, (a) properties might correspond to chains of properties, (b) IRIs might correspond to blank nodes, (c) OWL data properties might correspond to OWL annotation properties, or (d) corresponding resources might have different types like RDFS class, OWL individual, SKOS concepts, or

[1] https://isni.org
[2] https://viaf.org
[3] https://www.wikidata.org
[4] https://github.com/Wikidata/soweego
[5] https://www.wikidata.org/wiki/Wikidata:Mismatch_Finder

Wikidata instance. Therefore, it is necessary to enable a comparison beyond a simple one-to-one mapping between resources and properties and without restriction to specific RDF vocabularies.

R2. Result Provenance: The comparison of knowledge graphs might require the transformation of data from heterogeneous schemata. To be able to trace revealed issues back to the root cause, it is necessary to enable a provenance tracking of all (intermediate) results.

R3. No Repeated Review of Deviations: The re-evaluation of a knowledge graph will reveal the same deviations, if issues found in one knowledge graph did not get fixed, which might be out of control of the user. As the manual review of deviations is time consuming, it is necessary to enable the exclusion of known deviations from the results, to increase the visibility of new results.

R4. Integration into Ontology Quality Models: An comprehensive quality assessment of a knowledge graph would likely consider several quality measurements from different tools. Therefore, it is necessary to enable the integration of comparison results into ontology quality models, to make them interoperable with other tools for aggregation.

In an early version of ABECTO [4] we used —in difference to the current version— a HTTP based client-server-architecture for pipeline configuration as well as result access. Trials with this early version revealed that this is cumbersome to use, especially in automated quality assurance processes. That led to the following additional requirement:

R5. Integration into Quality Assurance Environments: As known from the field of software engineering, automation is key factor for successful quality assurance. Therefore, it is necessary to enable the easy integration of a comparison tool into standard quality assurance environments.

4. Tool Description

ABECTO is a tool for the comparison and evaluation of two or more knowledge graphs to assess their accuracy and completeness. It is a Java command line tool based on the RDF framework Apache Jena[6]. The source code is publicly available on GitHub[7] and Zenodo [9] under the permissive open source software license Apache 2.0[8]. In addition, ABECTO is available as a Docker image on GitHub. Parameters may enable exit codes unequal to zero in case of detected issues in a specific knowledge graph to signal a failure to the calling environment. This enables the use in Continuous Integration (CI) environments complying to **R5**.

Figure 1 gives a simplified overview of important components in ABECTO. A *plan* of interdependent steps describes the comparison of several knowledge graphs. This plan is configured in the default graph of an RDF multi graph document using the ABECTO vocabulary, which will be introduced in Section 4.1. A *step* describes the execution of one processor with the named RDF graphs returned by its direct and indirect predecessors as input. Each step generates up to (a) one *primary data graph* containing original or transformed data from one associated knowledge graph, (b) per knowledge graph one *associated metadata graph* for quality annotations and quality measurements, and (c) one *general metadata graph* for mapping data. Further, *predefined metadata graphs* might be added as input of a step to manually provide annotations or mappings. Execution metadata, like the provenance of the result graphs, will be stored in the default graph to comply to **R2**. For example, this enables to retrace the step that generated the mapping of two resources. A *processor* has one of four types: source processor, transformation processor, mapping processor, or comparison (and evaluation) processor. Each type is designated to one phase in the comparison framework depicted in Figure 2. However, the strict compliance of the plan to these phases is not enforced by ABECTO. Available processors are introduced in Section 4.2. An *aspect* describes the resources to compare.

[6] https://jena.apache.org/
[7] https://github.com/fusion-jena/abecto
[8] https://opensource.org/licenses/Apache-2.0

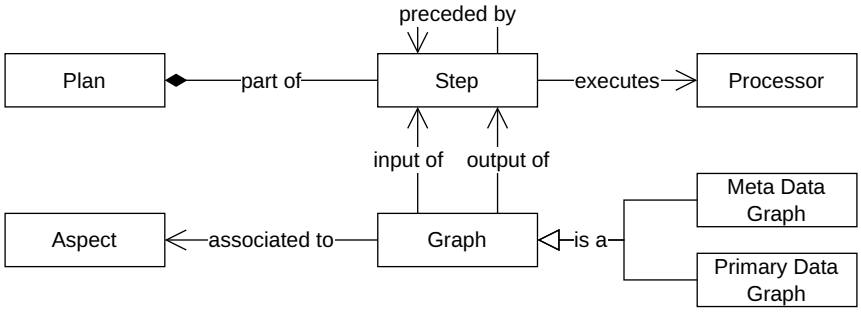

Fig. 1. Diagram of important components in ABECTO.

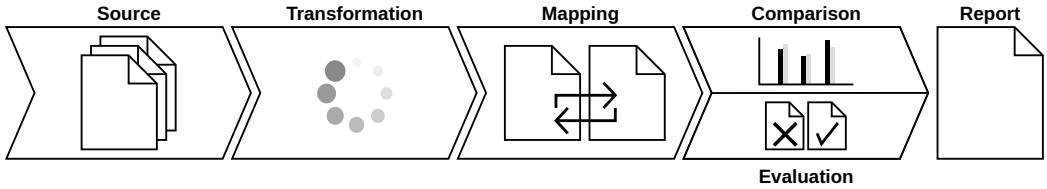

Fig. 2. Schematic of the comparison framework implemented in ABECTO.

One SPARQL query per knowledge graph and aspect specifies the resources to compare, represented by a key variable, and their related values, represented by further *variables*, that could get used by the processors. This enables the comparison of data with different modeling approaches, complying to **R1**. Figure 3 shows an example of the SPARQL queries specifying an aspect for two knowledge graphs with the key variable `person` and the related values `name` and `birthdate`.

```
PREFIX exo:<http://example.org/>
SELECT ?person ?name ?birthdate
WHERE {
  ?person exo:name ?name .
  OPTIONAL {
    ?person exo:birthday ?birthdate .
  }
}
```

```
PREFIX exc:<http://example.com/>
SELECT ?person ?name ?birthdate
WHERE {
  ?person exc:givenname ?firstname ;
          exc:surname   ?surname ;
          exc:birthdate ?birthdate .
  BIND(CONCAT(?firstname, ?surname) AS ?name)
}
```

Fig. 3. Two SPARQL queries specifying an aspect with the key variable `person` for two knowledge graphs with different schema.

The results can get stored in an RDF multi graph document and can get exported into several reports. Available reports are introduced in Section 4.3.

4.1. The ABECTO Vocabulary

The ABECTO Vocabulary[9] enables the description of the plan with aspects and steps, the execution results, and the results provenance. We use the following prefix to abbreviate the namespace in the vocabulary IRIs:

`av: <http://w3id.org/abecto/vocabulary#>`

To enable easy integration with other systems and thereby comply to **R4**, the ABECTO Vocabulary reuses several vocabularies:

- `dcat: <http://www.w3.org/ns/dcat#>`, a vocabulary for data set descriptions

[9] http://w3id.org/abecto/vocabulary

- dqv: <http://www.w3.org/ns/dqv#>, a vocabulary for data quality
- ldqd: <http://www.w3.org/2016/05/ldqd#>, a vocabulary for quality dimensions
- oa: <http://www.w3.org/ns/oa#>, a vocabulary for annotations
- om: <http://www.ontology-of-units-of-measure.org/resource/om-2/>, a vocabulary for units of measurement
- p-plan: <http://purl.org/net/p-plan#>, a vocabulary for workflows
- prov: <http://www.w3.org/ns/prov#>, a vocabulary for provenance data
- rdfg: <http://www.w3.org/2004/03/trix/rdfg-1/>, a vocabulary for RDF graphs
- sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#>, a vocabulary for statistical data and metadata

An overview of the vocabulary of plan description and results provenance is shown in Figure 4. It enables the description of the concepts introduced above, i.e. plan, step, primary data graph, meta data graph, processor, aspect, and variable.

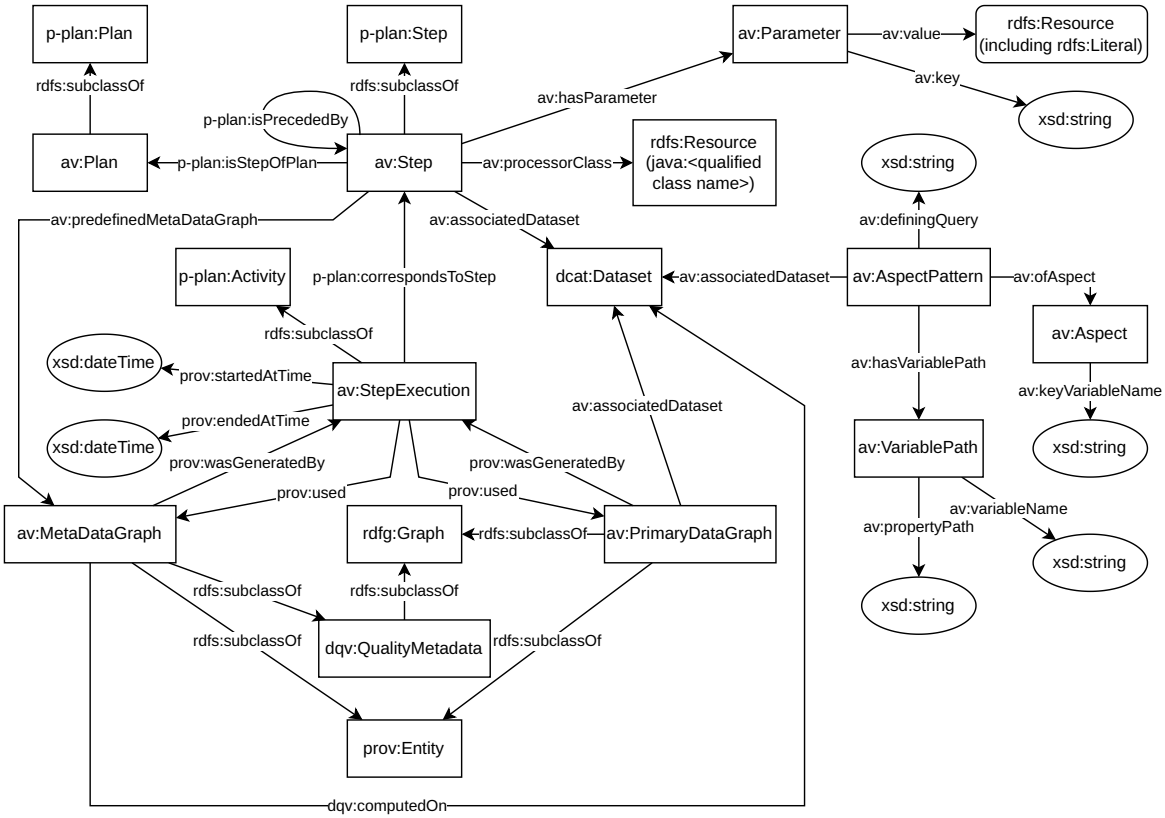

Fig. 4. The ABECTO vocabulary for the description of the plan and the result provenance.

Further resources are provided for the use inside the metadata graphs. The properties av:relevantResource, and av:correspondsToResource, av:correspondsNotToResource are available for the representation of the belonging of a resource to an aspect and for mapping results. The classes av:Deviation, av:ValueOmission, av:ResourceOmission, av:WrongValue, and av:Issue are used to annotate the compared knowledge graphs or contained resources. They are used together with the properties av:affectedAspect, av:affectedVariableName, av:affectedValue, av:comparedToValue, av:comparedToResource, and av:comparedToDataset, as shown in Figure 5 on the example of av:Deviation. The class av:QualityMeasurement together with the properties dqv:computedOn,

`av:affectedAspect`, `av:affectedVariableName`, `av:comparedToDataset`, `dqv:value`, and `sdmx-attribute:unitMeasure` is used to represent measurements on knowledge graphs, as shown in Figure 6.

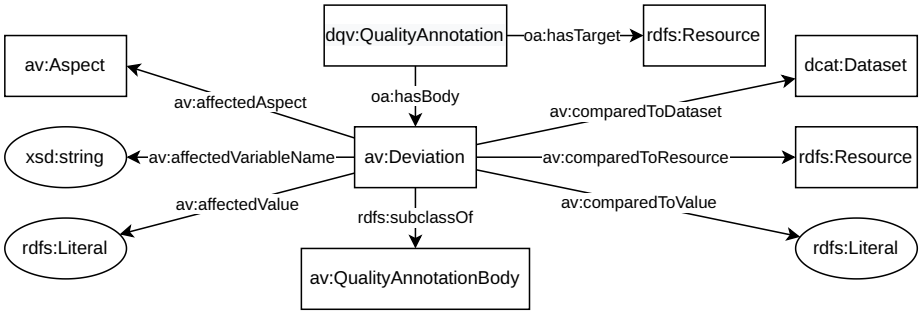

Fig. 5. The vocabulary for the annotation of deviating values of resources.

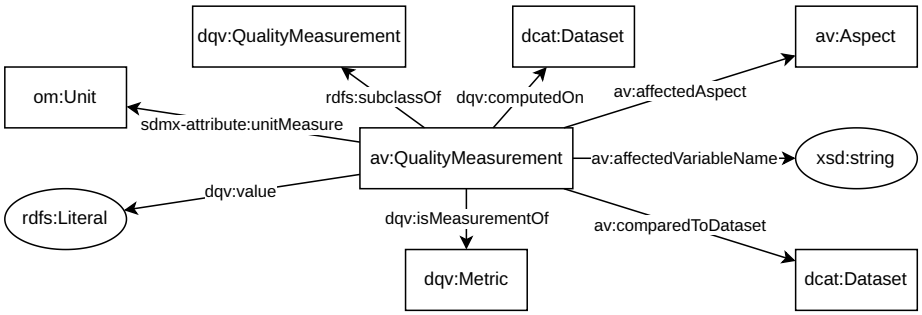

Fig. 6. The vocabulary for the representation of measurements.

4.2. Processors

ABECTO has a couple of build-in processors. In the pipeline description, processors get denoted by an IRI of the unofficial scheme `java` and a path equal to the canonical name of the processors class. This way to represent Java classes is used also in Apache Jena. We use the following prefix to abbreviate the namespace in the processor IRIs:

```
abecto: <java:de.uni_jena.cs.fusion.abecto.processor.>
```

4.2.1. Source Processors

Source Processors load RDF data from different sources and store them in the internal triple store for further processing.

The **File Source Processor** (`abecto:FileSourceProcessor`) loads RDF data from one or multiple locale files of one of the following formats: RDF/XML, TriG, N-Quads, Turtle, N-Triples, JSON-LD, SHACL Compact Syntax, TriX, and RDF Thrift. The format is automatically detected. The processor has the following parameter:

path One or multiple paths of RDF files that will be loaded. A path may either be absolute, or relative to the configuration file.

The **SPARQL Source Processor** (`abecto:SparqlSourceProcessor`) loads RDF data from a SPARQL endpoint. This makes ABECTO independent from the availability of knowledge graphs RDF dumps and may avoid the handling of large dump files, if only a small share of the data is needed. The resources of interest get defined by a SPARQL query, a list, or both. The processor partitions these resources into chunks and loads all statements containing these resource as subject or object. Depending on the given parameters, the other non-predicate resources

of loaded statements will also get loaded until a certain distance. Further parameters enable fine-grained control of statements to load and for the handling of errors like endpoint time outs. The processor has the following parameters:

service URL of the SPARQL endpoint to use.
list List of the relevant resources.
query SELECT query to retrieve a list of the relevant resources. All variables will be taken into account. None IRI values will be ignored. ORDER BY, LIMIT and OFFSET might become overwritten.
chunkSize Maximum number of resources to retrieve in one request. Default: 500
chunkSizeDecreaseFactor Factor to reduce the `chunkSize` after failed request to the source SPARQL endpoint. Default: 0.5
chunkSizeIncreaseFactor Factor to increase the `chunkSize` after successful request to the source SPARQL endpoint until the initial value got restored. Default: 1.5
maxDistance Maximum distance of loaded associated resources. Associated resources share a statement as an object with a retrieved resource as a subject, in case of any property, and vice versa, in case of followed inverse properties (see `followInverse`). Default: 0
followInverse Properties to track in inverse direction to compile a list of associated resources to load. That means that the subject of a statement whose property is in this list and whose object is a loaded resource will become an associated resource.
followUnlimited Properties that represent a hierarchy. Resources associated to a loaded resource by a followUnlimited property will be loaded unlimited, but will not cause retrieval of further resources not connected by a followUnlimited property or a followInverseUnlimited property. Default: rdfs:subClassOf, rdf:first, rdf:rest
followInverseUnlimited Properties that represent a hierarchy. Resources associated to a loaded resource by the inverse of a followInverseUnlimited property will be loaded unlimited, but will not cause retrieval of further resources not connected by a followUnlimited property or a followInverseUnlimited property.
ignoreInverse Properties to ignore in inverse direction. Statements with one of these properties will neither get loaded nor will their subjects become an associated resource.
maxRetries Total maximum number of retries of failed request to the source SPARQL endpoint. Default: 128

The **URL Source Processor** (`abecto:UrlSourceProcessor`) loads RDF data from one or multiple remote files of one of the following formats: RDF/XML, TriG, N-Quads, Turtle, N-Triples, JSON-LD, SHACL Compact Syntax, TriX, and RDF Thrift. The format is automatically detected. The processor has the following parameter:

url One or multiple URLs of RDF files that will be loaded.

4.2.2. Transformation Processors

Transformation processors derive additional primary data from the existing primary data. For example, this enables the derivation of implicit statements or the adjustment of value formating for the mapping or comparison.

The **Forward Rule Reasoning Processor** (`abecto:ForwardRuleReasoningProcessor`) applies forward rules to derive additional primary data. The processor has the following parameter:

rules The rules to apply on the primary data using the Apache Jena rule syntax[10].

The **SPARQL Construct Processor** (`abecto:SparqlConstructProcessor`) applies a SPARQL construct query on the primary data of a knowledge graph to derive additional primary data. The query execution will be repeated until a configured limit of execution or no new statements have been produced. The processor has the following parameters:

query The SPARQL construct query to apply on the primary data.
maxIterations Maximum number of executions of the query. Default: 1

[10] https://jena.apache.org/documentation/inference/#RULEsyntax

4.2.3. Mapping Processors

Mapping Processors provide correspondences and correspondence exclusions between resources in the knowledge graphs. The pipeline of a comparison plan may contain multiple complementary mapping processors. In case of contradicting results of mapping processors, the processor executed first takes precedence and contradicting correspondences or correspondence exclusions will not be added. That way, it is also possible to provide manual adjustments to the mapping by providing correspondences or correspondence exclusions in a predefined metadata graph in the configuration. A rule reasoner is used to derive implicit correspondences and correspondence exclusions. The reasoning applies immediately on new correspondences to consider them during the further mapping processor execution. Additionally, the inferences get persisted after a mapping processor execution succeeded.

The **Equivalent Value Mapping Processor** (`abecto:EquivalentValueMappingProcessor`) provides correspondences between resources of one aspect in different knowledge graphs, if they have equivalent values for all given variables. This is similar to the inferences of a OWL reasoner on inverse functional properties. Values are treated as equivalent if they are equivalent literals or if they are resources that are already known to correspond. If multiple values exist for one variable, only one pair of values must be equivalent. Unbound variables are treated as not equivalent. The processor has the following parameters:

aspect The aspects for which the correspondences get generated.
variables One or multiple variables that will be compared to determine the correspondence of resources.

The **Functional Mapping Processor** (`abecto:FunctionalMappingProcessor`) provides correspondences based on links from resources of another aspect. If corresponding resources from different knowledge graphs link with a given variable two resources, these resources will considered to correspond. This is similar to the inferences of a OWL reasoner on functional properties. The processor has the following parameters:

referringAspect The aspect of the resources linking the resources to map.
referringVariable The variable linking the resources to map.
referredAspect The aspect of the resources to map.

The **Jaro-Winkler Mapping Processor** (`abecto:JaroWinklerMappingProcessor`) provides correspondences based on the Jaro-Winkler Similarity [10] of `string` values using our implementation for efficient bounded Jaro-Winkler similarity based search [11]. Two resources are considered to correspond if for one variable in both directions the other variable value is the most similar value from the other knowledge graph and if the similarity score exceeds a threshold. The processor has the following parameters:

aspect The aspects for which the correspondences are generated.
variables One or multiple variables used to search corresponding resources.
threshold The similarity threshold the variable values of two resources must comply.
caseSensitive Determines, if case is taken into account during the search for corresponding resources.

The **Use Present Mapping Processor** (`abecto:UsePresentMappingProcessor`) provides correspondences based on existing links between resources in variable values. The processor has the following parameters:

aspect The aspects for which the correspondences get generated.
variable The variable that links a resource to a corresponding resource.

4.2.4. Comparison Processors

Comparison processors compare the primary data of the knowledge graphs using the correspondences provided by the mapping processors. They provide annotations on specific values, resources, and knowledge graphs or determine measurements on the knowledge graphs.

The **Completeness Processor** (`abecto:CompletenessProcessor`) is a comparison processor, that provides on the one hand `av:Issue` annotations for resource duplicates and `av:ResourceOmission` annotations. On the other hand, it provides per knowledge graph x measurements of (a) the duplicate-free count n_x of resources of an aspect, (b) the absolute coverage $m_{x,y}$ of resources of an aspect in another knowledge graph y, (c) the relative coverage $\frac{m_{x,y}}{n_y}$ of resources of an aspect in another knowledge graph y, and (d) the estimated completeness $\hat{C}_x = \frac{n_x}{\hat{N}}$ of resources of an aspect determined by a mark and recapture method as proposed by Razniewski et.al. [12]. We

use the mark and recapture method defined by Thomas [13], which permits multiple samples of different sample sizes: With T samples of sizes $n_1, \ldots, n_T$, and the sum of pairwise overlaps $M = \sum_{x=1}^{T-1} \sum_{y=x+1}^{T} m_{x,y}$, the estimated population size is $\hat{N} = \left(\sum_{x=1}^{T-1} \sum_{y=x+1}^{T} n_x n_y \right) \frac{1}{M}$. The processor has the following parameter:

aspects One or multiple aspects for which measurements and annotations will be generated.

The **Literal Value Comparison Processor** (`abecto:LiteralValueComparisonProcessor`) is a comparison processor, that provides `av:Deviation`, `av:ValuesOmission`, and `av:Issue` annotations on literal values for one variable of corresponding resources. Given a pair of corresponding resources from the same or different knowledge graphs, the processor will compare their values of one variable using the following procedure:

1. Skip non-literal values and add an `av:Issue` annotation for each of them to mark the invalid value.
2. Skip values to ignore due to their language tag.
3. Skip values already annotated with a `av:WrongValue` annotation, complying to **R3**.
4. Skip equivalent values from both resources.
5. If no value remained for only one resource, add a `av:ValuesOmission` for this resource and each remaining value of the other resource.
6. If at least one value remained for each resource, add a `av:Deviation` annotate for each pair of the remaining values.

Values are considered equivalent, if they are semantically equivalent with the following exceptions: (a) Numerical values are additionally considered equivalent even if a binary floating point literal and a literal with a type (derived from) `xsd:decimal` get compared and both values have the same value in the value space, but not necessarily if they have equal lexical representations [14]. (b) Depending on the parameters, temporal values of the types `xsd:date` and `xsd:dateTime` might get considered equivalent, if they have equivalent date parts. (c) Depending on the parameters, string values of the type `xsd:string` or `rdf:langString` might get considered equivalent, even if they have equal lexical representations but different language tags. The processor has the following parameters:

aspect The aspects for which annotations will be generated.
variables One or multiple variables that will be compared to determine the annotations.
languageFilterPatterns Zero, one or multiple language patterns to filter compared literals. If not empty, only string literals will be loaded, that match at least on of these patterns. String literals without language tag will match with "", all string literals with language tag match with "*". Default: empty
allowTimeSkip If true, two literals of the types `xsd:date` and `xsd:dateTime` with equal year, month and day part will match.
allowLangTagSkip If true, literals of the type `xsd:string` or `rdf:langString` with equal lexical value but different language tag will match.

The **Resource Value Comparison Processor** (`abecto:ResourceValueComparisonProcessor`) is a comparison processor, that provides `av:Deviation`, `av:ValuesOmission`, and `av:Issue` annotations on non-literal values for one variable of corresponding resources. Given a pair of corresponding resources from the same or different knowledge graphs, the processor will compare their values of one variable using the following procedure:

1. Skip literal values and add an `av:Issue` annotation for each of them to mark the invalid value.
2. Skip values already annotated with a `av:WrongValue` annotation, complying to **R3**.
3. Skip equivalent values from both resources.
4. If no value remained for only one resource, add a `av:ValuesOmission` for this resource and each remaining value of the other resource.
5. If at least one value remained for each resource, add a `av:Deviation` annotate for each pair of the remaining values.

Values are considered equivalent, if the IRIs are equal or the resources are considered to correspond. The processor has the following parameters:

aspect The aspects for which annotations will be generated.
variables One or multiple variables that will be compared to determine the annotations.

4.3. Reports

Reports are defined by one SPARQL query on the result multi graph and one Apache FreeMarker[11] template. ABECTO provides the following built-in reports:

The **Deviations Report** `deviations` contrast the variable value of one resource with the deviating value of a corresponding resource in CSV format. In addition, it provides the aspect and the knowledge graphs of the resources, the step that mapped the resources, and an annotation snippet to mark the second value as wrong. Each entry is intended to be handled by one of the following options: (a) fixing the value in the own knowledge graph, (b) fixing the value in the other knowledge graph, (c) manually fixing the mapping, or (d) annotating the other knowledge graphs value as wrong in the comparison configuration using the provided snippet.

The **Mapping Review Report** `mappingReview` provides an overview of all mappings as well as all missing resources in CSV format. It provides the aspect, the two affected knowledge graphs, the resource IRIs and labels (if applicable) the processor that provide the mapping or resource omission. The aim of this report is to enable manual revision and adjustment of the mapping.

The **Measurements Markdown Report** `measurementsMarkdown` provides a tabular display of measurements on the knowledge graphs in Markdown[12] format.

The **Resource Omission Report** `resourceOmissions` lists all missed resources per knowledge graph in CSV format. In addition, it provides the labels of the missing resource and the knowledge graph that contained the missing resources.

The **Wikidata Mismatch Finder Report** `wdMismatchFinder` provides encountered deviations in the Mismatch Finder CSV import file format[13], provided that they can be displayed in the format.

5. Limitations

In the current version of ABECTO, all input and output data are managed in the main memory. Therefore, the scalability of the tool with regards to the knowledge graph size is limited. This might get improved by enabling the optional use of Apache Jenas disk storage component TDB2[14].

6. Workflow

For the monitoring of a knowledge graphs quality, ABECTO is intended to be used in a cyclic workflow, that is shown in Figure 7. It can be aligned to different iterating development processes. The workflow consists of four steps:

The **Plan Execution** provides all necessary data for the further process. Ideally, it is triggered automatically. For example, the plan could be executed after each commit into a version control repository to provide reports based on each version of an RDF file and automatically warn on revealed deviations. After this step, all reports are available for analysis.

During the **Result Analysis** a knowledge graph maintainer checks based on the reports whether it is necessary to correct or extend the knowledge graph or the ABECTO plan. An detected deviation makes at least one of the following actions necessary:

- change or addition of a value in the own knowledge graph to correct the own knowledge graph,
- annotation of a value in another knowledge graph as wrong value to avoid the future appearance of the deviation in the reports,

[11] https://freemarker.apache.org/
[12] https://daringfireball.net/projects/markdown/
[13] https://github.com/wmde/wikidata-mismatch-finder/blob/main/docs/UserGuide.md#creating-a-mismatches-import-file
[14] https://jena.apache.org/documentation/tdb2/

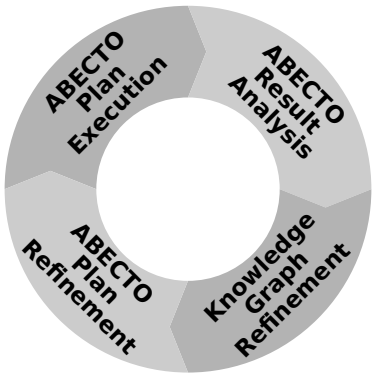

Fig. 7. Illustration of the workflow to use ABECTO in knowledge graph development.

- change or addition of a value in another knowledge graph to avoid the future appearance of the deviation in the reports,
- manually adjustment of the mapping to avoid the future appearance of deviations between not corresponding resources,
- correction of values relevant for the mapping to correct the own knowledge graph and to avoid the future appearance of deviations between not corresponding resources.

In the **Knowledge Graph Refinement** phase and the **Plan Refinements** phase, necessary changes on the knowledge graph and the ABECTO plan will be performed.

7. Adaption

To demonstrate the usefulness of ABECTO, we present two comparison projects that use ABECTO.

7.1. Comparison of Units of Measurement Data from Four Knowledge Graphs

In the first comparison project, we compared units of measurement related data from four knowledge graphs[15]. It covers the aspects units of measurement and quantity kinds including conversion factors and dimension vectors. The data origin from OM 2[16], QUDT 2[17], SWEET 3[18], loaded via RDF files, and Wikidata, loaded via SPARQL endpoint. We used the UCUM[19] code as well as matches stated in the knowledge graphs to map the units. The quantity kinds were mapped using Jaro-Winkler Similarity of the labels. The mappings were supplemented by several manual adjustments. We compared the units regarding their symbol, associated quantity kind, and conversion values and the quantity kinds regarding their symbol and dimension vector. Further, we computed completeness measures for both aspects. The measurement results for resource count and completeness are shown in Table 1. For a correct interpretation please note, that in contrast to the other knowledge graphs, Wikidata also features a large amount of historical units of measurement.

The comparison revealed more then 600 deviations between the knowledge graphs, which point to potential errors in the knowledge graphs, including more then 300 deviations related to conversion factors and offsets. The maintainers of OM, QUDT and SWEET were notified about the comparison results and for a portion of the deviations, we have proposed changes to the knowledge graphs. In Wikidata, we directly corrected a couple of errors

[15] https://github.com/fusion-jena/abecto-unit-ontology-comparison
[16] https://github.com/HajoRijgersberg/OM
[17] https://qudt.org
[18] https://github.com/ESIPFed/sweet
[19] https://ucum.org

Table 1
The number of unit and quantity kind resources and the estimated completeness per knowledge graph.

Knowledge Graph	Units Count	Units Completeness	Quantity Kinds Count	Quantity Kinds
OM	1429	14 %	108	6 %
QUDT	1709	16 %	328	17 %
SWEET	136	1 %	-	-
Wikidata	5807	55 %	1769	92 %

revealed by the deviations. That way, the comparison already caused and will hopefully also cause in future several improvements[20] of the knowledge graphs.

7.2. Comparison of Space Flight Data in Wikidata and DBpedia

In the second comparison project, we compared the data about space flight related resources[21]. It covers the aspects astronauts and spacecrafts and space missions. The data origin from Wikidata and DBpedia and are loaded in both cases from the respective SPARQL endpoint. We used the URLs of the related Wikipedia articles to map the resources of all aspects. The mapped astronauts were compared regarding their birth data, death date labels, and time in space. For spacecrafts we compared the COSPAR ID (an identifier for artificial objects in space), crew members, labels, landing date, launch date, and the "Satellite Catalog Number" (another identifier for artificial objects in space). Space missions were compared regarding their duration, inclination, labels, landing date, landing site, launch date, launch site, launch vehicle, mass, member, next mission, number of orbits, previous mission, and vehicle. Further, we also computed completeness measures for both aspects, but only to get the number of resources per aspect, shown in Table 2. As every Wikipedia article is associated with one Wikidata resource (but not vice versa) and every DBpedia resource is based on at least one Wikipedia article, the completeness measure is not reliably.

Table 2
The number of astronaut, spacecraft and space mission resources per knowledge graph.

Knowledge Graph	Astronauts Count	Spacecraft Count	Space Missions Count
DBpedia	739	4708	3405
Wikidata	795	7457	145

The comparison revealed more then 800 deviations between Wikidata and DBpedia resources regarding their birth date, COSPAR IDs, crew members, death date, duration, label, landing date, launch date, Satellite Catalog Number and time in space properties. As values in DBpedia origin from Wikipedia, this also points to errors in the English Wikipedia. As far as they were already representable in the according format, the results were provided to the Wikidata Mismatch Finder[22].

8. Conclusion

We presented ABECTO, the first generic tool for the comparison of knowledge graphs for the assessment of their accuracy and completeness. It provides a pipeline based framework for the comparison of multiple knowledge graphs. ABECTO does not depend on the existence of a gold standard. Thereby we overcome an more then 20 years old problem in the field of ontology engineering. Design decisions were led by requirements learned in earlier

[20] 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, Wikidata: https://www.wikidata.org/wiki/User:Jmkeil/ABECTO_Provoked_Edits#Based_on_the_Comparison_of_Unit_Ontologies

[21] https://github.com/fusion-jena/abecto-space-travel-comparison

[22] https://mismatch-finder.toolforge.org/store/imports, uploaded on August 18th 2022

knowledge graph comparison projects. This enables the use of ABECTO for the comparison of a wide range of knowledge graphs, the interoperability with other tools, and the integration into automated processes.

In the two comparison projects, we demonstrated the usefulness of ABECTO for the improvement of real world knowledge graphs. We hope, that ABECTO will be adapted by knowledge graph curators to keep track on the accuracy and completeness of facts in their work. The capability to use ABECTO in continuous integration processes allows them to regularly and automatically compare their knowledge graph with other knowledge graphs. That way, it provides novel opportunities to strengthen the reliability of a knowledge graph. Further, ABECTO empowers users of knowledge graphs to easily compare available knowledge graphs. They will not longer have to blindly trust the represented facts or to perform a tedious manual review of axioms. The framework will highlight questionable facts. That way, users will be able to take a more educated decision on the selection of ontologies.

Acknowledgements

Many thanks to Alsayed Algergawy and Franziska Zander and the author's supervisor Birgitta König-Ries for very helpful comments on earlier drafts of this manuscript.

References

[1] M. Färber, F. Bartscherer, C. Menne and A. Rettinger, Linked data quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO, *Semantic Web* **9**(1) (2018), 77–129. doi:10.3233/SW-170275.

[2] J.M. Keil, Ontology ABox Comparison, in: *The Semantic Web: ESWC 2018 Satellite Events*, A. Gangemi, A.L. Gentile, A.G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, J.Z. Pan and M. Alam, eds, Lecture Notes in Computer Science, Vol. 11155, Springer, 2018, pp. 240–250. ISBN 978-3-319-98191-8. doi:10.1007/978-3-319-98192-5_43.

[3] J.M. Keil and S. Schindler, Comparison and evaluation of ontologies for units of measurement, *Semantic Web* **10**(1) (2019), 33–51. doi:10.3233/SW-180310.

[4] J.M. Keil, ABECTO: An ABox Evaluation and Comparison Tool for Ontologies, in: *The Semantic Web: ESWC 2020 Satellite Events*, A. Harth, V. Presutti, R. Troncy, M. Acosta, A. Polleres, J.D. Fernández, J.X. Parreira, O. Hartig, K. Hose and M. Cochez, eds, Lecture Notes in Computer Science, Vol. 12124, Springer, 2020. ISBN 978-3-030-62326-5. doi:10.1007/978-3-030-62327-2_24.

[5] R.Y. Wang and D.M. Strong, Beyond Accuracy: What Data Quality Means to Data Consumers, *J. Manag. Inf. Syst.* **12**(4) (1996), 5–33. doi:10.1080/07421222.1996.11518099.

[6] P.R.S. Visser and T.J.M. Bench-Capon, A Comparison of Four Ontologies for the Design of Legal Knowledge Systems, *Artif. Intell. Law* **6**(1) (1998), 27–57. doi:10.1023/A:1008251913710.

[7] A. Angjeli, A.M. Ewan and V. Boulet, ISNI and VIAF – Transforming ways of trustfully consolidating identities, 2014. http://library.ifla.org/id/eprint/985/.

[8] C. Bianchini, S. Bargioni and C.C. Pellizzari di San Girolamo, Beyond VIAF, *Information Technology and Libraries* **40**(2) (2021). doi:10.6017/ital.v40i2.12959.

[9] J.M. Keil, ABox Evaluation and Comparison Tool for Ontologies (ABECTO) v1.0.1, Zenodo, 2022. doi:10.5281/zenodo.7009167.

[10] W.E. Winkler, String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage, in: *Proceedings of the Section on Survey Research*, American Statistical Association, 1990, pp. 354–359. http://eric.ed.gov/?id=ED325505.

[11] J.M. Keil, Efficient Bounded Jaro-Winkler Similarity Based Search, in: *BTW 2019*, T. Grust, F. Naumann, A. Böhm, W. Lehner, T. Härder, E. Rahm, A. Heuer, M. Klettke and H. Meyer, eds, Gesellschaft für Informatik, Bonn, 2019, pp. 205–214. doi:10.18420/btw2019-13.

[12] S. Razniewski, F.M. Suchanek and W. Nutt, But What Do We Actually Know?, in: *Proceedings of the 5th Workshop on Automated Knowledge Base Construction, AKBC@NAACL-HLT 2016*, J. Pujara, T. Rocktäschel, D. Chen and S. Singh, eds, The Association for Computer Linguistics, 2016, pp. 40–44. ISBN 978-1-941643-53-2. doi:10.18653/v1/W16-1308.

[13] P. Thomas, Generalising multiple capture-recapture to non-uniform sample sizes, in: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*, S. Myaeng, D.W. Oard, F. Sebastiani, T. Chua and M. Leong, eds, ACM, 2008, pp. 839–840. ISBN 978-1-60558-164-4. doi:10.1145/1390334.1390531.

[14] J.M. Keil and M. Gänßinger, The Problem with XSD Binary Floating Point Datatypes in RDF, in: *The Semantic Web: ESWC 2022*, P. Groth, M. Vidal, F.M. Suchanek, P.A. Szekely, P. Kapanipathi, C. Pesquita, H. Skaf-Molli and M. Tamper, eds, Lecture Notes in Computer Science, Vol. 13261, Springer, 2022, pp. 165–182. doi:10.1007/978-3-031-06981-9_10.