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 most cases, there is no gold standard to compare against. 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, ABECTO provides quality annotations like value deviations and quality measurements like completeness. This enables knowledge graph curators to monitor the quality and can help potential users to select an appropriated knowledge graph for their purpose. We demonstrate the usefulness of ABECTO for the improvement of knowledge graphs with two example use case applications.

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

1. Introduction

Knowledge graphs provide domain knowledge for a wide range of applications. The faultless operation of these applications depends on the quality of the used knowledge graph. Useful criteria for the selection or evaluation of knowledge graphs include the quality dimensions accuracy and completeness. For example, inaccurate knowledge about measurement unit conversions could cause wrong values in an integrated measurement dataset from sources with different measurement units after an alignment of the measurement units [1]. Likewise, the use of inaccurate or incomplete knowledge about publications would, for example, cause the calculation of wrong bibliometric values.

With *knowledge graph* we denote any kind of RDF graph, typically but not necessarily with a focus on actual instances (A-Box) instead of schema definitions (T-Box), and including RDF graphs that are typically denoted as *ontology*. State-of-the-art approaches for the validation of knowledge graphs, like SHACL [2] for A-Box statements or OOPS! [3] for T-Box statements, use constraints on statements to detect missing or wrong statements. Therefore, they can only detect outliers out of the range of plausible values or completely missing properties. However, they can not detect wrong values in a plausible range or an incomplete set of statements, like a measurement unit conversion factor of 0.001 instead of 1000, or one out of five publication authors missing. Assessing accuracy and completeness of knowledge graphs requires data to compare with. The literature [4] regularly recommends to use a gold standard for the evaluation. However, a gold standard does in most cases not exist. An alternative is the comparison with other, overlapping knowledge graphs of arbitrary quality [5].

In an earlier comparison of unit ontologies we identified a large number of issue in these ontologies [1]. At the same time, we identify a high 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 [5] and developed the ABox Evaluation and Comparison Tool for Ontologies (ABECTO) [6], 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 and can help users to select an appropriated knowledge graph to use. The source code is publicly available on GitHub[1] and Zenodo [7] under the permissive open source software license Apache 2.0[2]. In this paper, we introduce the tool and two example applications.

The paper is structured as follows: We give an overview about related work in Section 2, provide requirements for the tool in Section 3, introduce ABECTO in Section 4, explain the intended workflow with ABECTO in Section 5, present two example applications in Section 6, and conclude the paper in Section 7.

2. Related Work

The quality of data can be expressed regarding several dimension. Wang and Strong [8] identified 20 general data quality dimension. ISO 25012:2008 [9] provides an general data quality model. The defined quality dimensions have also been used for the assessment of RDF data. In a systematic survey, Zaveri et al. [10] provide an comprehensive overview of 18 quality dimensions and measures relevant for knowledge graphs. Based on that and on the W3C Data Quality Vocabulary (QDV) [11], Radulovic et al. [12] built a knowledge graph specific quality model and provided guidelines for its application. Likewise, but mainly based on the quality dimensions by Wang and Strong, Faerber et al. [4] did an extensive investigation on five freely available large knowledge graphs. Also based on the work by Zaveri et al., Debattista et al. [13] applied 27 quality measurements from several quality dimensions on all available datasets in the Linked Open Data Cloud and determined the 11 most informative quality indicators using a Principal Component Analysis (PCA). Our work focuses on the quality dimensions *accuracy* and *completeness*.

Wang and Strong define the quality dimension *accuracy* as "the extent to which data are correct, reliable, and certified free of error" [8]. 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 [4, 10]. In this work, we focus on semantic validity, which is also referred to as consistency with the represented reality [14] and must not be confused with inferential consistency.

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" [8]. For knowledge graphs, this can be divided into (a) *schema completeness*, which assesses the degree to which relevant classes and properties are present, (b) *population completeness*, which assesses the degree to which relevant objects of the represented reality are present, (c) *property completeness* or column completeness, which assesses the degree to which relevant property values are present, and (d) *interlinking completeness*, which assesses the degree of resources interlinking to other knowledge graphs [4, 10]. In addition, Issa et al. [15] propose the three completeness types currency completeness, metadata completeness, and labelling completeness. But in our point of view, *currency completeness* actually belongs to either schema completeness, property completeness, or the currency dimension, but is to vaguely defined for a clear judgment; *metadata completeness* actually belongs to the belivability dimension; and *labelling completeness* actually belongs to the understandability dimension [10]. In this work, we focus on population completeness and property completeness.

2.1. Assessment of Accuracy and Completeness

For the two quality dimensions of accuracy and completeness applies that their assessments requires some source for domain knowledge as a basis for comparison [4, 10, 12, 13]. The term "comparison" is ambiguously used in the context of semantic web technologies [5]. We will use the term to describe the comparison of entire knowledge graphs regarding certain aspects to evaluate or select knowledge graphs. Other notions of the term describe the comparison of single or a few resources in recommendation, matching or merging tasks or the comparison of knowledge graph versions [5]. The 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

[1] https://github.com/fusion-jena/abecto

[2] https://opensource.org/licenses/Apache-2.0

a problem since early work [16] on ontology evaluation until more recent works [4, 10]. Moreover, other recent extensive studies [12, 13] did not apply accuracy and completeness measurements in their experiments.

An alternative to the comparison with a gold standard is the comparison of multiple knowledge graphs with each other [5]. Nevertheless, only a few tools have recently been developed to make use of the comparison of knowledge graphs to address this issue. All of these tools are dedicated to specific knowledge graphs: The library authority data initiatives International Standard Name Identifier (ISNI)[3] and Virtual International Authority File (VIAF)[4] regularly run processes for integration and consolidation of the data of their providers and among each other [17]. Subsequent manual reviews of the results reveal errors in the source data that get communicated to the providers to enable error correction. Bianchini et al. compared data from Wikidata[5] and VIAF regarding the contained data about persons [18]. 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*[6] 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*[7] 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.

Rashid et al. describe another alternative approach for the assessment of completeness based on the comparison of knowledge graph versions [14]. Among others, they propose measures for persistency (the continuous presence of instances) and completeness (the continuous presence of instance properties) per class. These measures are derived from the elementary measures *instances per class*, *property frequency per class and property*, and *number of used properties per class* measured on two consecutive knowledge graph versions. The elementary measures can be measured independently and efficiently on each knowledge graph versions. That way, their measures allows an efficient monitoring of a knowledge graph regarding the unintended removal of information. In case of the detection of a potential quality problem, further manual investigations are required to identify the actual cause of the problem. However, the absence of information that were nether present in the knowledge graph can not be detected.

Approaches without need of a basis for comparison use heuristics for the approximation of completeness or constraints on statements to detect missing or wrong statements. State-of-the-art constraints based approaches are SHACL [2] for A-Box statements or OOPS! [3] for T-Box statements. However, these approaches can only detect outliers out of the range of plausible values or completely missing properties. Radulovic et al., for example, propose to approximate interlinking completeness based on the ratio of interlinked and total subjects [12]. An overview of more completeness assessment approaches is provided by Issa et al. [15].

2.2. Relation of Knowledge Graph Comparison to Link Discovery

Closely related to the comparison of knowledge graph is the work in the field of *Link Discovery*. It aims to enrich knowledge graph with links to other knowledge graphs, to apply the Linked Data paradigms[8]. Silk [19], an early work in this field applied hierarchically numerically aggregated similarity measures between resources from different knowledge graphs to identify resources to interlink up to a certain similarity threshold. The state-of-the-art Link Discovery framework LIMES [20] applies link specifications defined by boolean filters that consist of similarity or distance measures and a threshold, and that get aggregated by boolean operators. These are complemented by an execution engine for optimized execution and several supervised and unsupervised machine learning approaches that return link specifications for specific knowledge graph pairs.

---

[3] https://isni.org
[4] https://viaf.org
[5] https://www.wikidata.org
[6] https://github.com/Wikidata/soweego
[7] https://www.wikidata.org/wiki/Wikidata:Mismatch_Finder
[8] https://www.w3.org/DesignIssues/LinkedData

The comparison of knowledge graphs differs from Link Discovery in that knowledge graphs comparison analyses property values of interlinked resources, whereas Link Discovery produces an interlinking based on property values. That way, their relation in a knowledge graph construction process is ambiguous. Knowledge graph comparison could be considered as (a) a preceding step of Link Discovery, to enable an more extensive interlinking, or (b) a follow up step of Link Discovery, to use an extended interlinking for more extensive comparison results. The Link Discovery approach COLIBRI [21], which is based on an earlier version of LIMES, addressed this with an entanglement of both steps: For two or more knowledge graphs, it alternately attempted (a) to generate an interlinking based on the previous repair step, and (b) to repair property values based on the previous interlinking step using a voting mechanism. However, the work had its main emphasis in Link Discovery, and it was only applicable for properties with an 1:1 correspondence between the knowledge graphs.

3. Requirements

During an earlier comparison of unit ontologies [1] we identified challenges that led to the following requirements for a generic knowledge graph 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 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. This aligns with general requirements on knowledge graph construction tools [22].

R2. Result Provenance: The comparison 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 provide the provenance of all (intermediate) results. This aligns with general requirements on knowledge graph construction tools [22].

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

R4. Integration into Ontology Quality Models: An comprehensive quality assessment of a knowledge graph would likely consider several different quality measures that have been measured by multiple tools. Therefore, it is necessary to enable the integration of the comparison results into ontology quality models for aggregation with measurements of other quality measures by other tools, as also outlined by Radulovic et al. [12].

R5. Suitable Result Representation: To ease the manual review of the results or enable an result review with other tools, it is necessary to provide a suitable representation of the results.

In an early version of ABECTO [6] we used —in difference to the current version— a HTTP based client-server-architecture for the configuration and result presentation result. Trials revealed that this is cumbersome to use, especially in automated quality assurance processes. That led to the following additional requirement:

R6. 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.

Further, the following general requirement is also relevant for ABECTO:

R7. Scalability: "It should be possible to integrate a large number of data sources as well as a high amount of data (data scalability)." ([22])

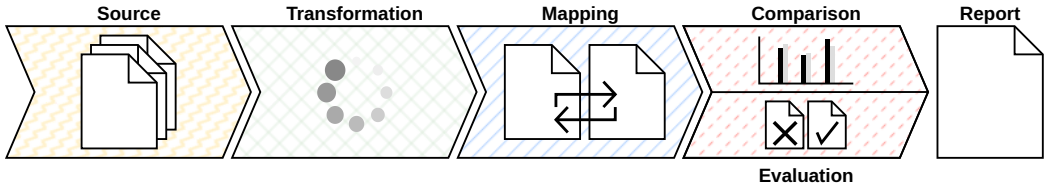

Fig. 1. Schematic of the comparison framework implemented in ABECTO. (Revised visualization from [5].)

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

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[9]. The source code is publicly available on GitHub[1] and Zenodo [7] under the permissive open source software license Apache 2.0[2]. In addition, ABECTO is available as a Docker image on GitHub. Parameters may enable a failure exit status code 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 **R6**.

The comparison of several knowledge graphs in ABECTO is described by a *plan* of interdependent steps. This plan is configured in the *default graph* [23] of an *RDF dataset* [23] document using the ABECTO vocabulary, which will be introduced in Section 4.1. The vocabulary diagram in Figure 2 provides an overview of the components of ABECTO. A *step* describes the execution of one processor with the *named graphs* [23] returned by its direct and indirect predecessors as input. The processors can be configured with *parameters* defined per step. Each step generates up to (a) one *primary data graph*, a named graph containing original or transformed data from one associated knowledge graph, (b) per knowledge graph one *associated metadata graph*, a named graph for quality annotations and quality measurements, and (c) one *general metadata graph*, a named graph for mapping data. Further, named graphs called *predefined metadata graphs* can be added in the plan RDF dataset document as input of a step and its successor to provide manual annotations or mappings. Execution metadata, like the provenance of the result graphs, will be stored in the default graph to comply with **R2**. For example, this enables to retrace the step that generated the mapping of two resources. We assume a "graph name denotes the named graph" dataset semantic [24]. That way, the possible wrong or inconsistent data in the primary data graphs do not participate in the truth of the dataset, but the graph name can be used to identify the named graph to describe its provenance. 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 1. However, the strict compliance of the plan to these phases is not enforced by ABECTO. Available processors will be introduced in Section 4.2.

ABECTO does not automatically match the schemas of the knowledge graphs. An *aspect* describes resources that must be compared. A user must provide one SPARQL query per knowledge graph and aspect that specifies the resources to compare. The resource is represented by a *key variable*, which is a query variable with a certain name that is specified per aspect. Property values related to the resource are represented by further query *variables* and can be used by the processors. This enables the comparison of data modeled in different ways, 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`. The aspect SPARQL queries can also be used to align the scope of the compared knowledge graphs to get meaningful comparison results. For example, a knowledge graph about space flight might not contain data about persons except of astronauts. To compare them with a general purpose knowledge graph, persons from that knowledge graphs must be restricted to astronauts.

By now ABECTO manages all input and output data in the main memory. Therefore, the scalability (**R7**) of the tool with regards to the knowledge graph size is limited. This might be improved by implementing the optional use of Apache Jenas disk storage component TDB2[10].

[9] https://jena.apache.org/
[10] https://jena.apache.org/documentation/tdb2/

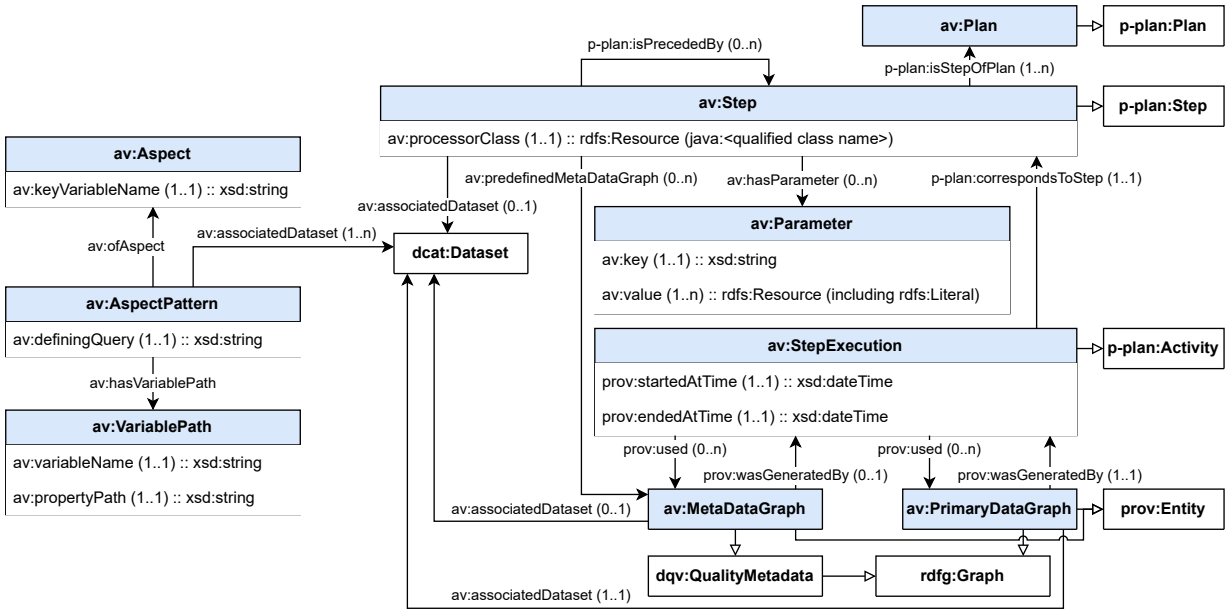

Fig. 2. The ABECTO vocabulary for the description of the plan and the result provenance.

```
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 schemas.

The results of an ABECTO plan execution can be stored in an RDF dataset document for processing by further tools, like a quality data dashboard, comply with **R5**. To enable an easy integration into automated processes, complying to **R6**, we decided to optimize ABECTO for the use via command line interface instead of providing an graphical user interface. To nevertheless provide human readable presentations of the results, complying with **R5**, the results can be exported into several reports. Reports are defined by one SPARQL query on the result dataset and one Apache FreeMarker[11] template. ABECTO provides the following built-in reports: (a) The *Deviations Report* contrast the property 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. (b) The *Mapping Review Report* provides an overview of all mappings as well as all missing resources in CSV format. The aim of this report is to enable manual revision and adjustment of the mapping. (c) The *Measurements Markdown Report* provides a tabular display of measurements on the knowledge graphs in Markdown[12] format. (d) The *Resource Omission Report* lists all missed resources per knowledge graph together with the labels and the source knowledge graph of the missing resource in CSV format. (e) The *Wikidata Mismatch Finder Report* provides encountered deviations in the Mismatch Finder CSV import file format[13], as far as it is possible to represent them in the format.

[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

4.1. The ABECTO Vocabulary

The ABECTO Vocabulary[14] enables the description of the plan with aspects and steps, the execution results, and the results provenance. To enable easy integration with other systems and thereby comply with **R4**, the ABECTO Vocabulary reuses the vocabularies DQV[15], and LDQD[16] for the representation of quality data, as well as P-Plan[17], and PROV-O[18] for the representation of workflows and result provenance. The vocabulary consists of two parts. The first part, which is visualized in Figure 2, enables the description of the ABECTO concepts: `av:Plan` for plans, `av:Step` together with `av:Parameter` for steps, `av:PrimaryDataGraph` for primary data graphs, `av:MetaDataGraph` for meta data graphs, `av:Aspect` together with `av:AspectPattern` and `av:VariablePath` for aspect. Processors are represented by an IRI of the unofficial scheme `java` and a path equal to the canonical name of the processors class, an approach also used in Apache Jena. This permits the use of a prefix for processor IRIs[19]. `dcat:Dataset` is reused for the representation of knowledge graphs and the actual execution of a step is represented with `av:StepExecution`. The property `av:associatedDataset` is used to connect steps, aspect patterns and graphs to a knowledge graph, `av:predefinedMetaDataGraph` assigns predefined metadata graphs to steps.

The second part serves to describe the gained quality data and mappings and should be used inside of metadata graphs. The properties `av:correspondsToResource`, and `av:correspondsNotToResource` are available for the representation of mapping results. We could not reuse the similar properties `owl:sameAs`, `owl:equivalentClass` and `owl:equivalentProperty` from OWl according to **R1**, as these imply a certain type of resource, which might differ between knowledge graphs. Further, we could not reuse `skos:exactMatch`, as it must not link resources from the same scheme. To annotate the compared knowledge graphs and contained resources, several classes of quality annotation bodies (`av:QualityAnnotationBody`) are provided: `av:Deviation` annotates corresponding resources with deviation of variable values, `av:ValueOmission` annotates resources without an equivalent variable values found at a corresponding resource, `av:ResourceOmission` annotates knowledge graphs without a corresponding resource of a resource found in another knowledge graph, `av:WrongValue`, which is intended for use in predefined metadata graphs, annotates resources with a value known to be wrong, and `av:Issue` annotates resources or knowledge graphs with further issues like corresponding resources in the same knowledge graph, i.e. duplicates. Further, `av:QualityMeasurement` together with the provided `dqv:Metric` instances is used to represent measurements in an interoperable way (**R4**) by reusing DQV. For the instances of `av:QualityAnnotationBody` or `av:QualityMeasurement` the following properties are provides: `av:affectedAspect` to link the aspect of the investigated resource, `av:affectedVariableName` to link the name of the investigated variable, and `av:comparedToDataset` to link the knowledge graph that has been used for comparison during the investigation. Further, for the instance of `av:QualityAnnotationBody` the following additional properties are provided: `av:affectedValue` to link the property value that has been investigated, `av:comparedToResource` to link the resource that has been used for comparison during the investigation, and `av:comparedToValue` to link the property value that has been used for comparison during the investigation.

4.2. Processors in ABECTO

ABECTO has a couple of built-in processors. In the following sections, we give an overview of the available processors. A more detailed descriptions of the processors and their mandatory and optional parameters is provided in the ReadMe file[1].

---

[14] `av: <http://w3id.org/abecto/vocabulary#>`
[15] `dqv: <http://www.w3.org/ns/dqv#>`
[16] `ldqd: <http://www.w3.org/2016/05/ldqd#>`
[17] `p-plan: <http://purl.org/net/p-plan#>`
[18] `prov: <http://www.w3.org/ns/prov#>`
[19] `abecto: <java:de.uni_jena.cs.fusion.abecto.processor.>`

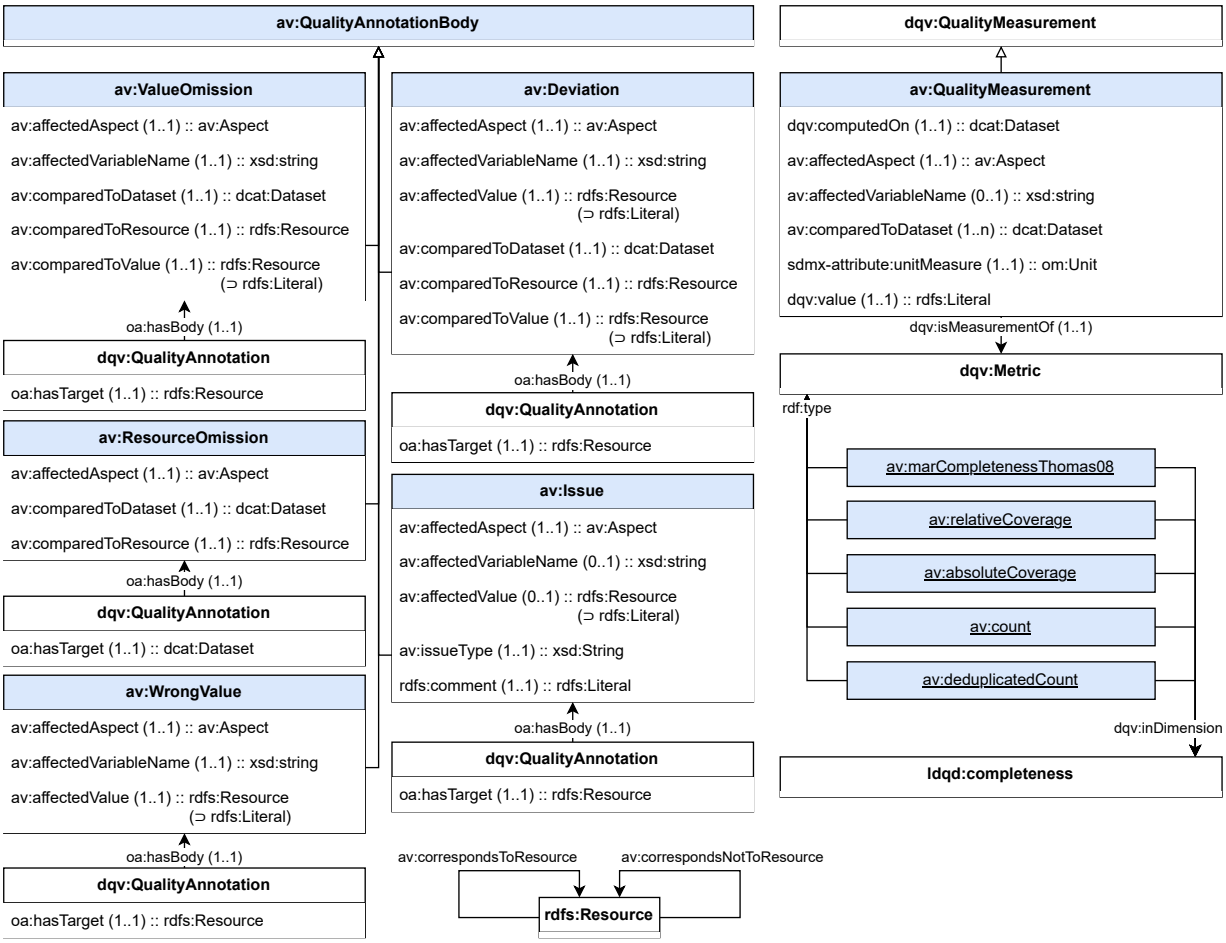

Fig. 4. The ABECTO vocabulary for gained quality data and mappings.

4.2.1. Source Processors

Source Processors load RDF data from different sources and store them in the internal triple store for processing. The **File Source Processor** (`abecto:FileSourceProcessor`) loads RDF data from one or multiple locale files, the **URL Source Processor** (`abecto:UrlSourceProcessor`) does the same for remote files. They support the formats RDF/XML, TriG, N-Quads, Turtle, N-Triples, JSON-LD, SHACL Compact Syntax, TriX, and RDF Thrift. The format is automatically detected.

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

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 formatting for the mapping or comparison. The **Forward Rule Reasoning Processor** (`abecto:ForwardRuleReasoningProcessor`) applies forward rules to derive additional primary data.

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 no new statements are produced or a configured maximum number of executions.

4.2.3. Mapping Processors

Mapping Processors provide correspondences and correspondence exclusions between resources of the same aspect in the knowledge graphs. In case of contradicting results of consecutively executed mapping processors, the first takes precedence and contradicting correspondences or correspondence exclusions will be ignored. That way, it is also possible to provide manual adjustments to the mapping in the ABECTO plan by providing correspondences or correspondence exclusions in a predefined metadata graph in the configuration. A rule reasoner is used to derive further implicit correspondences and correspondence exclusions. However, ABECTO is not supposed to be a comprehensive mapping tool. It provides a couple of mapping processors to enable simple mapping strategies. For more sophisticated mapping strategies, dedicated tools for ontology matching or link discovery should be used.

The results of external mapping tools and mappings provided by the investigated knowledge graphs can be used with the **Use Present Mapping Processor** (`abecto:UsePresentMappingProcessor`). Beside that, the **Equivalent Value Mapping Processor** (`abecto:EquivalentValueMappingProcessor`) provides correspondences between resources of one aspect in different knowledge graphs, if they have equivalent values for a given set of variables, similar to the inferences of an OWL reasoner on inverse functional properties. The **Functional Mapping Processor** (`abecto:FunctionalMappingProcessor`) provides correspondences based on incoming links from resources of another aspect, similar to the inferences of an OWL reasoner on functional properties. The **Jaro-Winkler Mapping Processor** (`abecto:JaroWinklerMappingProcessor`) provides correspondences using the Jaro-Winkler Similarity [25], which is a common choice for label based mappings [20]. For scalability (**R7**), we apply our implementation for efficient bounded Jaro-Winkler similarity based search [26].

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 **Population Comparison Processor** (`abecto:PopulationComparisonProcessor`) provides on the one hand `av:ResourceOmission` annotations and `av:Issue` annotations for resource duplicates. On the other hand, it provides per knowledge graph k_x measurements of (a) the count n'_x (`av:count`) and (b) the duplicate-free count n_x (`av:deduplicatedCount`) of resources of an aspect, (c) the estimated population completeness $\hat{C}_x = \frac{n_x}{\hat{N}}$ of resources of an aspect determined by a mark and recapture method (`av:marCompletenessThomas08`) as proposed by Razniewski et al. [27], (d) the absolute coverage $m_{x,y}$ (`av:absoluteCoverage`) and (e) the relative coverage $\frac{m_{x,y}}{n_y}$ (`av:relativeCoverage`) of resources of an aspect in another knowledge graph k_y. For the estimated population completeness we use the mark and recapture method defined by Thomas [28], 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 $\hat{N} = \left(\sum_{x=1}^{T-1} \sum_{y=x+1}^{T} n_x n_y\right) \frac{1}{M}$.

In contrast to the original application of mark and recapture methods on animals, duplicates might occur in a sample during the assessment of knowledge graphs. The population comparison algorithm with duplicate handling is shown in Figure 5. It starts in line 4 to 9 with the initialization of the pairwise overlaps $m_{x,y}$ for all knowledge graph pairs, and for each knowledge graph the set of unmatched resources U_x of the aspect a, the resources counts with duplicates n'_x and without duplicates n_x. The unmatched resources and the count without duplicates are initialized with all resources of the aspect in the knowledge graph, including duplicates, and will be reduces during the following iteration trough all sets of corresponding resources of the aspects in line 10 to 28. For these sets R^c, the subset per knowledge graph C_x is retrieved and removed from U_x. Based on the size of C_x, it is determined, if the entity represented by the resources in the set R^c is in a knowledge graph missing (line 15) or present (line 19) and whether the entity is present multiple times (line 23). Based on that, $m_{x,y}$ and n_x are updated accordingly and duplicates are annotated. Finally, in line 29 to 44, the total pairwise overlap M, the estimated population size $\hat{N}$, the estimated completeness measurements $\hat{C}_x$ and the relative coverage measurements $\frac{m_{x,y}}{n_y}$ are calculated and all measurements, as well as the resource omission annotations are stored.

```
1  FUNCTION comparePopulation(a, // aspect
2                             K = {k_1,...,k_T}, // knowledge graphs
3                             C) // sets of corresponding resources
4  FOR x = 1 TO T
5    FOR y = 1 TO T
6      m_{x,y} = 0 // initialise pairwise overlaps
7    U_x = getResources(a,k_x) // init. unmatched resources
8    n'_x = |U_x| // count with duplicates
9    n_x = |U_x| // initialise count without duplicates
10 FOREACH R^c IN C // corresponding resources set
11   FOR x = 1 TO T
12     C_x = U_x ∩ R^c // corresponding resources of k_x
13     U_x = U_x \ C_x
14   FOR x = 1 TO T
15     IF |C_x| = 0 // entity not covered by k_x
16       FOR y = 1 TO T
17         FOREACH r IN C_y
18           store(av:ResourceOmission,k_x,a,k_y,r)
19     IF |C_x| > 0 // entity covered by k_x
20       FOR y = 1 TO T
21         IF |C_y| > 0 // entity covered by k_y
22           m_{x,y} = m_{x,y} + 1
23     IF |C_x| > 1 // duplicates contained in k_x
24       n_x = n_x - |C_x| + 1
25       FOREACH r1 IN C_x
26         FOREACH r2 IN C_x
27           IF r1 ≠ r2
28             store(av:Issue,r1,k_x,a,,,k_y,"Duplicate",r2)
29 M = Σ_{x=1}^{T-1} Σ_{y=x+1}^{T} m_{x,y}
30 IF M ≠ 0
31   N̂ = (Σ_{x=1}^{T-1} Σ_{y=x+1}^{T} n_x n_y) 1/M // est. population size
32   FOR x = 1 TO T
33     Ĉ_x = n_x / N̂ // estimated population completeness
34     store(av:marCompletenessThomas08,k_x,a,,K \ {k_x},Ĉ_x)
35 FOR x = 1 TO T
36   store(av:count,k_x,a,,∅,n'_x)
37   store(av:deduplicatedCount,k_x,a,,∅,n_x)
38   FOR y = 1 TO T
39     IF x ≠ y
40       store(av:absoluteCoverage,k_x,a,,{k_y},m_{x,y})
41       IF n_y ≠ 0
42         store(av:relativeCoverage,k_x,a,,{k_y},m_{x,y}/n_y)
43       FOREACH r IN U_y
44         store(av:ResourceOmission,k_x,a,k_y,r)
```

Fig. 5. Pseudo code of the population comparison algorithm.

The **Property Comparison Processor** (`abecto:PropertyComparisonProcessor`) investigates the property values of corresponding resources for aspect variables v and provides `av:Deviation`, `av:Issue`, and `av:ValuesOmission` annotations on them. A *property deviation* is a case of a pair of corresponding resources with each having a value not present in the other resource. A *property omission* is a case of a pair of corresponding resources with one having a value not present in the other, but not vice versa. Further it measures per knowledge graph k_x (a) the count n'_x (`av:count`), (b) the duplicate-free count n_x (`av:deduplicatedCount`) and (c) the estimated property completeness $\hat{C}_x$ (`av:marCompletenessThomas08`) of property values of an aspect variable, (d) the absolute coverage $m_{x,y}$ (`av:absoluteCoverage`) and (e) the relative coverage $\frac{m_{x,y}}{n_y}$ (`av:relativeCoverage`) of property values of an aspect variable in another knowledge graph k_y, based on deduplicated resources and property values. The considered `xsd:string` and `rdf:langString` literals can be restricted to specific languages with a processor parameter. The estimated property completeness measure is based on the same mark and recapture method [27, 28] as the estimated population completeness measure. However, the measurement is a bit more complicated due to two additional problems.

In addition to the equivalence of entities, the equivalence of property values must be considered too. The entity mapping is reused to determine the equivalence of object property values. Data property values are considered equivalent, if they are semantically equivalent, but with the following exceptions to comply with **R1**: (a) Numeric literals are additionally considered as equivalent even if they have incompatible datatypes, i.e. pairs out of the three groups `xsd:float`, `xsd:double` and (datatypes derived from) `xsd:decimal`, if both literals either represent the same special value (`INF`, `-INF`, `NaN`) or represent equal numbers in the value space, but not necessarily if they have equal lexical representations [29]. For example, `"0.5"^^xsd:decimal` and `"0.5"^^xsd:float` are additionally considered as equivalent. However, `"0.1"^^xsd:decimal` and `"0.1"^^xsd:float`, which actually represents the number 0.100 000 001 4..., are not considered as equivalent. (b) Depending on processor parameters, temporal values of the types `xsd:date` and `xsd:dateTime` might be considered equivalent, if they have equivalent date parts. (c) Depending on the parameters, string values of the type `xsd:string` or `rdf:langString` might be considered equivalent, even if they have equal lexical representations but different language tags.

The pseudo code of the property comparison algorithm for one aspect variable is shown in Figure 6. It starts in line 5 to 15 with the initialization of the pairwise overlaps $m_{x,y}$ for all knowledge graph pairs, and for each knowledge graph with the retrieval of resources R_x and property values per resource $V_{x,r}$, as well as the measurement of the resources counts with duplicates n'_x and the initialization of the counts without duplicated n_x. The values are deduplicated based on the equivalency of property values described above. Strings with excluded languages are filtered from the values $V_{x,r}$. In an actual implementation, it might be beneficial to not keep all R_x and $V_{x,r}$ sets in the memory, but to retrieve them on demand to substantially reduce the memory consumption, which is necessary for

```
1  FUNCTION compareProperties(a, // aspect
2                             v, // variable
3                             K = {k_1,...,k_T}, // knowledge graphs
4                             C) // sets of corresponding resources
5  FOR x = 1 TO T
6    n'_x = 0 // initialise count with duplicates
7    n_x = 0 // initialise count without duplicates
8    FOR y = x + 1 TO T // only once per pair
9      m_{x,y} = 0 // initialise pairwise overlaps
10   FOREACH r IN getResources(a, k_x)
11     R_x = getResources(a, k_x)
12     V_{x,r} = getValues(a, v, k_x, r)
13     removeExcluded(V_{x,r})
14     n'_x = n'_x + |V_{x,r}|
15     n_x = n_x + |deduplicated(V_{x,r})|
16 FOREACH R^c IN C
17   FOR x = 1 TO T
18     V^+ = [] // init. values list (with duplicates)
19     FOREACH r IN R^c
20       V^+ = V^+ + deduplicated(V_{x,r}) // append to list
21       removeKnownWrongValues(a, v, k_x, r, V_{x,r})
22     n_x = n_x - (|V^+| - |deduplicated(V^+)|)
23   // compare values
24   FOR x = 1 TO T - 1
25     FOR y = x + 1 TO T // only once per pair
26       // create common value-resource look-up
27       V_unique = ∅
28       FOREACH r IN R^c
29         FOREACH v_next IN V_{x,r} ∪ V_{y,r} // next value to map
30           FOREACH v_mapped IN V_unique
31             IF equivalentValue(v_mapped, v_next)
32               R_{v_mapped} = R_{v_mapped} ∪ {r}
33               // equiv. values point to one set
34               R_{v_next} ↦ R_{v_mapped}
35               BREAK // continue with next value
36           R_{v_next} = {r}
37           V_unique = V_unique ∪ {v_next}
38       FOREACH v_unique IN V_unique
39         IF R_{v_unique} ∪ R_x ≠ ∅ AND R_{v_unique} ∪ R_y ≠ ∅
40           m_{x,y} = m_{x,y} + 1
41       FOREACH r_x IN R^c ∩ R_x // resources from k_x
42         FOREACH r_y IN R^c ∩ R_y // resources from k_y
43           U_x = ∅ // init. unmateched values from r_x
44           U_y = ∅ // init. unmateched values from r_y
45           FOREACH v_x IN V_{x,r_x}
46             IF r_y ∉ R_{v_x}
47               U_x = U_x ∪ {v_x}
48           FOREACH v_y IN V_{y,r_y}
49             IF r_x ∉ R_{v_y}
50               U_y = U_y ∪ {v_y}
51           IF U_x = ∅
52             FOREACH u_y IN U_y
53               store(av:ValueOmission, k_x, r_x, a, v, k_y, r_y, u_y)
54           ELSE IF U_y = ∅
55             FOREACH u_x IN U_x
56               store(av:ValueOmission, k_y, r_y, a, v, k_x, r_x, u_x)
57           ELSE
58             FOREACH u_x IN U_x
59               FOREACH u_y IN U_y
60                 store(av:Deviation, k_x, r_x, u_x, a, v, k_y, r_y, u_y)
61                 store(av:Deviation, k_y, r_y, u_y, a, v, k_x, r_x, u_x)
62 M = Σ_{x=1}^{T-1} Σ_{y=x+1}^{T} m_{x,y}
63 IF M ≠ 0
64   N̂ = (Σ_{x=1}^{T-1} Σ_{y=x+1}^{T} n_x n_y) 1/M // est. population size
65   FOR x = 1 TO T
66     Ĉ_x = n_x/N̂ // estimated property completeness
67     store(av:marCompletenessThomas08, k_x, a, v, K \ {k_x}, Ĉ_x)
68 FOR x = 1 TO T
69   store(av:count, k_x, a, v, ∅, n'_x)
70   store(av:deduplicatedCount, k_x, a, v, ∅, n_x)
71   FOR y = x + 1 TO T // only once per pair
72     store(av:absoluteCoverage, k_x, a, v, {k_y}, m_{x,y})
73     store(av:absoluteCoverage, k_y, a, v, {k_x}, m_{x,y})
74     IF n_y ≠ 0
75       store(av:relativeCoverage, k_x, a, v, {k_y}, m_{x,y}/n_y)
76     IF n_x ≠ 0
77       store(av:relativeCoverage, k_y, a, v, {k_x}, m_{x,y}/n_x)
```

Fig. 6. Pseudo code of the property comparison algorithm.

the algorithms scalability (**R7**). The actual comparison is done during an iteration trough all sets of corresponding resources of the aspects in line 16 to 61. First, the algorithm adjusts the deduplicated count n_x by subtracting duplicated property values across duplicated resources in one knowledge graph (line 22). Simultaneously, it skips values already annotated with a `av:WrongValue` annotation, complying to **R3** (line 21). Next, it iterates over all pairs of knowledge graphs to (1) create a common value-resource look-up and a common set of non-equal property values of resources in the property set V_{unique} in line 27 to 37, and then (2) update in line 40 the pairwise overlap $m_{x,y}$ based on the value-resource look-up and V_{unique}. After that, (3) the algorithm iterates over all pairs of corresponding resources of both knowledge graphs to determine unmatched property values U_x, U_y, and (4) determine omissions and deviations based on U_x, U_y: If no value remains for only one resource (line 51 or 54), a `av:ValuesOmission` is stored for this resource and each remaining value of the other resource. If at least one value remained for each resource (line 57), a `av:Deviation` annotate is stored for each pair of the remaining values. Finally, in line 62 to 77, the total pairwise overlap M, the estimated population size $\hat{N}$, the estimated completeness measurements $\hat{C}_x$ and the relative coverage measurements $\frac{m_{x,y}}{n_y}$ are calculated and all measurements are stored.

5. Workflow

To monitor the quality of a knowledge graph, ABECTO should be used in an incremental workflow, because the re-execution of ABECTO after an improvement of the monitored knowledge graph or the ABECTO plan might reveal new problems that have been hidden by the fixed problem. This workflow can be aligned to different iterative development processes and consists of four steps:

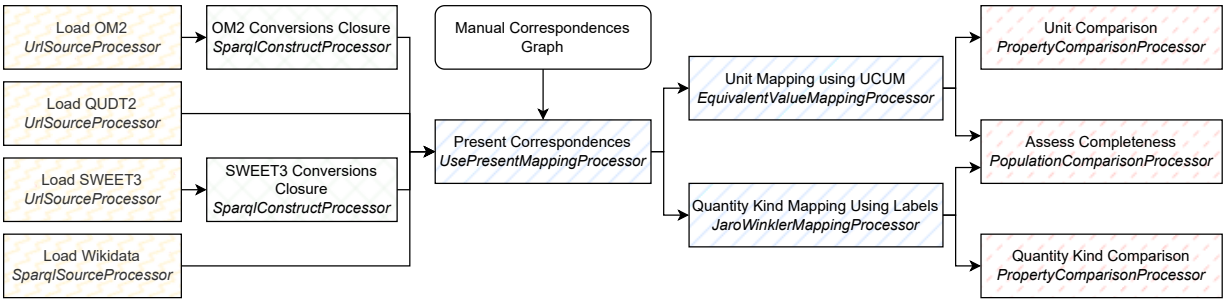

Fig. 7. The ABECTO pipeline for the comparison of measurement units data.

The **Plan Execution** step executes ABECTO and provides the necessary data for the next steps. Ideally, this step is triggered automatically, e.g. as part of a continuous integration pipeline that is executed after each commit into a version control repository, to provide reports based on each version of an RDF file and immediately warn on new problem. After this step, all reports are available for analysis.

During the **Result Analysis** a maintainer of the monitored knowledge graph uses the reports to check whether it is necessary to correct or add values in the monitored knowledge graph or to adapt the ABECTO plan. A deviation makes it necessary to perform at least on of the following actions: (a) change a wrong value or add a missing value in the monitored knowledge graph to resolve the deviation, (b) add a wrong value annotation for a compared knowledge graph into the ABECTO plan to suppress the future appearance of the deviation in the reports, (c) propose to change a wrong value or to add a missing value in a compared knowledge graph to resolve the deviation, (d) manually adjust the mapping in the ABECTO plan to avoid the future appearance of a deviation between not corresponding resources, or (e) correct values that are relevant for the mapping in the monitored knowledge graph to correct it and to avoid the future appearance of a deviation between not corresponding resources.

In the **Knowledge Graph Refinement** phase and the **Plan Refinements** phase, necessary changes identified in the Result Analysis step on the monitored knowledge graph and the ABECTO plan have to be performed.

6. Use Case Applications

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

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

In the first comparison project [30], we compared units of measurement related data from four knowledge graphs, which are indispensable for FAIR and digital data in science, engineering and beyond. Ongoing activities in the meteorological community, as the CODATA Task Group on Digital Representation of Units of Measurement[20], a CGPM 2022 resolution "On the global digital transformation and the International System of Units"[21], and the resulting CIPM Task Group on the SI Digital Framework[22], which all aim for an FAIR compliant and machine actionable encoding of units, documents the demand for high-quality representations of units. We investigate the state-of-the-art measurement units representations OM 2[23], QUDT 2[24], as well as the according subsets of SWEET 3[25] and Wikidata, loaded via RDF files or SPARQL endpoint. The comparison covers the aspects *units of measurement* and *quantity kind* including conversion factors and dimension vectors. We used the UCUM[26] code, as well as cor-

[20] https://codata.org/initiatives/task-groups/drum/
[21] https://www.bipm.org/en/cgpm-2022/resolution-2
[22] https://www.bipm.org/en/committees/ci/cipm/wg/cipm-tg-dsi
[23] https://github.com/HajoRijgersberg/OM
[24] https://qudt.org
[25] https://github.com/ESIPFed/sweet
[26] https://ucum.org

Table 1

The count and deduplicated count of unit resources, their coverage of resources in other knowledge graphs and the estimated completeness per knowledge graph compared to all other knowledge graphs, as well as the count, deduplicated count and completeness of quantity kind resources.

	Units							Quantity Kinds		
Knowledge Graph	Count	Dedup. Count	Covered in OM	QUDT	SWEET	Wikidata	est. Completeness	Count	Dedup. Count	est. Completeness
OM	1444	1429	-	432 (27 %)	87 (64 %)	916 (15 %)	14 %	116	108	5 %
QUDT	1619	1578	432 (3 %)	-	96 (71 %)	483 (8 %)	15 %	424	397	19 %
SWEET	140	136	87 (6 %)	96 (6 %)	-	74 (1 %)	1 %	-	-	-
Wikidata	6014	6009	916 (64 %)	483 (31 %)	74 (54 %)	-	58 %	1897	1831	87 %

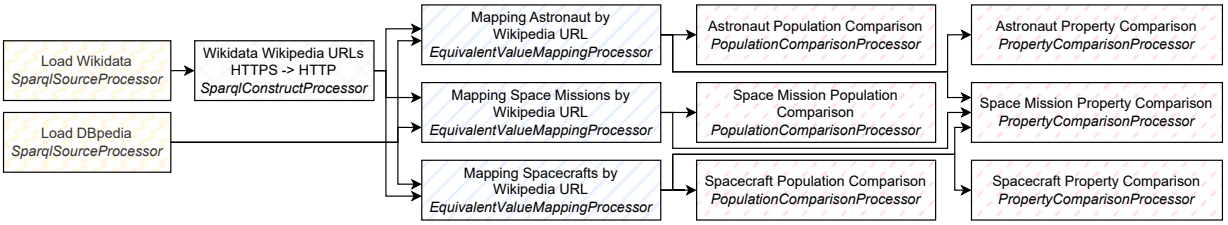

Fig. 8. The ABECTO pipeline for the comparison of space travel data in Wikidata and DBpedia.

respondences 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. It is important to note that Wikidata also features a large amount of historical units of measurement, which are out of scope of the other knowledge graphs.

The comparison revealed more then 600 deviations, 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 revealed by the deviations. That way, the comparison already caused several improvements[27] of the knowledge graphs and enables further improvements. The comparison project has been taken over as continuous integration task in the OM repository.

6.2. Comparison of Space Travel Data in Wikidata and DBpedia

In the second comparison project [31], we compared the data about space travel related resources. With this project, we aim to support the development end establishment of the Wikidata Mismatch Finder in the Wikidata community. The comparison covers the aspects astronauts, 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 date, 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, members, next mission, number of orbits, previous mission, and vehicle. Further, we also computed the population of for all aspects to get the number of resources per aspect and their coverage, as shown in Table 2. Every Wikipedia article is associated with

[27] 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

Table 2
The count of astronaut, spacecraft and space mission resources per knowledge graph and the coverage of resources in the other knowledge graph.

	Astronauts		Spacecraft		Space Missions	
Knowledge Graph	Count	Covered in other KG	Count	Covered in other KG	Count	Covered in other KG
DBpedia	738	629 (77 %)	4716	1361 (18 %)	3436	610 (68 %)
Wikidata	819	629 (85 %)	7601	1361 (29 %)	901	610 (18 %)

one Wikidata resource (but not vice versa) and every DBpedia resource is based on at least one Wikipedia article. Therefore, the completeness measure is not meaningful and is skipped from the table. The deduplicated count is skipped, as duplicates can not be identified due to the mapping strategy.

The comparison revealed more then 800 deviations between Wikidata and DBpedia resources regarding their birth and death date, COSPAR IDs, crew members, duration, label, launch and landing date, Satellite Catalog Number and time in space properties. In particular, it revealed many errors in dates. 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 are regularly[28] provided to the Wikidata Mismatch Finder.

7. Conclusion

We presented ABECTO, the first generic tool for the comparison of knowledge graphs to assess 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 a more then 20 years old problem in the field of ontology engineering. Design decisions were led by requirements identified in earlier knowledge graph comparison projects. Thus, ABECTO is suited for the comparison of a wide range of knowledge graphs, the integration into automated processes, and is interoperable with other tools.

In two comparison projects, we demonstrated the usefulness of ABECTO for improving real world knowledge graphs. ABECTO can be used by knowledge graph curators to keep track on the accuracy and completeness of facts. The capability to use ABECTO in continuous integration processes allows knowledge graph curators to regularly and automatically compare their knowledge graph with other knowledge graphs. That way, it provides a novel opportunity to strengthen the reliability of a knowledge graph. Further, ABECTO empowers users of knowledge graphs to easily compare available knowledge graphs. They will no longer have to blindly trust the represented facts or to perform a tedious manual review of axioms: The framework highlights questionable facts. Thus, users will be able to take a better educated decision on the selection of knowledge graphs.

Acknowledgements

Many thanks to Alsayed Algergawy, Franziska Zander, Samira Babalou and the author's supervisor Birgitta König-Ries, as well as the reviewer Edgard Marx and two anonymous reviewers for very helpful comments on earlier drafts of this manuscript.

References

[1] 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.

[2] RDF Data Shapes Working Group, Shapes Constraint Language (SHACL), H. Knublauch and D. Kontokostas, eds, 2017. https://www.w3.org/TR/2017/REC-shacl-20170720/.

[28] https://mismatch-finder.toolforge.org/store/imports

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

[4] 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.

[5] 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.

[6] 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.

[7] J.M. Keil, ABox Evaluation and Comparison Tool for Ontologies (ABECTO) v2.1.0, Zenodo, 2023. doi:10.5281/zenodo.7840767.

[8] 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.

[9] I.J.S.. Software and systems engineering, ISO/IEC 25012:2008 Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Data quality model, Technical Report, ISO/IEC, 2008. https://www.iso.org/standard/35736.html.

[10] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann and S. Auer, Quality assessment for Linked Data: A Survey, *Semantic Web* **7**(1) (2016), 63–93. doi:10.3233/SW-150175.

[11] D. on the Web Best Practices Working Group, Data on the Web Best Practices: Data Quality Vocabulary, A.I. Riccardo Albertoni, ed., 2016. https://www.w3.org/TR/2016/NOTE-vocab-dqv-20161215/.

[12] F. Radulovic, N. Mihindukulasooriya, R. García-Castro and A. Gómez-Pérez, A comprehensive quality model for Linked Data, *Semantic Web* **9**(1) (2018), 3–24. doi:10.3233/SW-170267.

[13] J. Debattista, C. Lange, S. Auer and D. Cortis, Evaluating the quality of the LOD cloud: An empirical investigation, *Semantic Web* **9**(6) (2018), 859–901. doi:10.3233/sw-180306.

[14] M. Rashid, M. Torchiano, G. Rizzo, N. Mihindukulasooriya and O. Corcho, A quality assessment approach for evolving knowledge bases, *Semantic Web* **10**(2) (2019), 349–383. doi:10.3233/sw-180324.

[15] S. Issa, O. Adekunle, F. Hamdi, S.S. Cherfi, M. Dumontier and A. Zaveri, Knowledge Graph Completeness: A Systematic Literature Review, *IEEE Access* **9** (2021), 31322–31339. doi:10.1109/ACCESS.2021.3056622.

[16] 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.

[17] 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/.

[18] 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.

[19] J. Volz, C. Bizer, M. Gaedke and G. Kobilarov, Silk - A Link Discovery Framework for the Web of Data, in: *LDOW*, 2009. http://ceur-ws.org/Vol-538/ldow2009_paper13.pdf.

[20] A.-C.N. Ngomo, M.A. Sherif, K. Georgala, M.M. Hassan, K. Dreßler, K. Lyko, D. Obraczka and T. Soru, LIMES: A Framework for Link Discovery on the Semantic Web, *KI - Künstliche Intelligenz* **35**(3–4) (2021), 413–423. doi:10.1007/s13218-021-00713-x.

[21] A.-C.N. Ngomo, M.A. Sherif and K. Lyko, Unsupervised Link Discovery through Knowledge Base Repair, in: *Lecture Notes in Computer Science*, Springer International Publishing, 2014, pp. 380–394. doi:10.1007/978-3-319-07443-6_26.

[22] M. Hofer, D. Obraczka, A. Saeedi, H. Köpcke and E. Rahm, Construction of Knowledge Graphs: State and Challenges, arXiv, 2023. doi:10.48550/ARXIV.2302.11509.

[23] R. Cyganiak, D. Wood and M. Lanthaler (eds), RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[24] RDF 1.1 Working Group, RDF 1.1: On Semantics of RDF Datasets, A. Zimmermann, ed., 2014. https://www.w3.org/TR/2014/NOTE-rdf11-datasets-20140225/.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

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