SN Computer Science (2023) 4:16
https://doi.org/10.1007/s42979-022-01429-x

ORIGINAL RESEARCH

A Knowledge Graph Perspective on Knowledge Engineering

Umutcan Simsek[1] · Elias Kärle[2] · Kevin Angele[1,2] · Elwin Huaman[1] · Juliette Opdenplatz[1,2] · Dennis Sommer[1] · Jürgen Umbrich[2] · Dieter Fensel[1]

Received: 12 April 2022 / Accepted: 16 September 2022 / Published online: 17 October 2022

Abstract
For over 50 years researchers and practitioners have searched for ways to elicit and formalize expert knowledge to support AI applications. Expert systems and knowledge bases were all results of these efforts. The initial efforts on knowledge bases were focused on defining a domain and task intensionally with rather complex ontologies. The increasing complexity of knowledge and knowledge-based systems eventually led to the development of knowledge engineering methodologies. Knowledge graphs, in contrast to the traditional knowledge bases, represent knowledge more extensionally with a very large set of explicit statements and rather simpler and smaller ontologies. This paradigm change calls for a new take on knowledge engineering that focuses on the curation of ABox statements. In this paper, we introduce various aspects of the knowledge graphs lifecycle namely creation, hosting, curation and deployment. We define each task, give example approaches from the literature and explain our approach with a running example. Additionally, we present the German Tourism Knowledge Graph that is being implemented with our methodology.

Keywords Knowledge graphs · Knowledge engineering · Knowledge graph lifecycle

Introduction

The vision of AI research that aimed to give computers the ability to "solve problems that are meant for humans" was implemented in the form of a General Problem Solver (GPS) [1]. The main idea behind GPS was that any problem can be solved by a computer if it can be formally defined. The GPS was an important milestone for AI and had important successes in solving many problems. However, it was soon realized that real-world problems need specific tasks and domain knowledge as a heuristic to tackle with AI. The "knowledge principle" coined by Feigenbaum [2] is still valid today. Is a conversational AI useful, when it can "understand" a user's wish "to have a romantic dinner at a restaurant" in multiple languages via large-scale matrix multiplications, but does not know what dinner is or that it is eaten in a restaurant? Would an autonomously driving car not be safer, if it could cross-check its statistical inferences with a knowledge base?

This article is part of the topical collection "Web Information Systems and Technologies 2021" guest edited by Joaquim Filipe, Francisco Domínguez Mayo and Massimo Marchiori.

✉ Umutcan Simsek
umutcan.simsek@sti2.at

✉ Dieter Fensel
dieter.fensel@sti2.at

Elias Kärle
elias.kaerle@onlim.at

Kevin Angele
kevin.angele@sti2.at

Elwin Huaman
elwin.huaman@sti2.at

Juliette Opdenplatz
juliette.opdenplatz@sti2.at

Dennis Sommer
dennis.sommer@sti2.at

Jürgen Umbrich
juergen.umbrich@onlim.com

[1] Semantic Technology Institute Innsbruck, University of Innsbruck, Technikerstrasse 21a, 6020 Innsbruck, Tyrol, Austria

[2] Onlim GmbH, Weintraubengasse 22, 1020 Vienna, Austria

All scenarios above demonstrate the need for knowledge for AI to be useful, and not for the first time. Researchers and practitioners have looked for different ways to engineer or acquire expert knowledge and formalize them for computers in some form such as knowledge bases for many years. Knowledge graphs are the most recent attempt to achieve this knowledge acquisition goal. Knowledge Graphs are very large semantic nets that integrate heterogeneous data sources [3]. When looked closely, there are significant differences between traditional knowledge bases and knowledge graphs:

- Traditional knowledge bases are more focused on having complex and expressive TBoxes[1] and smaller ABoxes.[2] The aim is to make the implicit knowledge described by TBox explicit with reasoning. Knowledge graphs in comparison have much larger ABoxes and relatively small and less complex TBoxes. The aim is to represent knowledge extensionally. The large ABox also means that knowledge graphs are typically several orders of magnitude larger than knowledge bases.
- Knowledge bases are typically tailored for specific domains and tasks and curated at a much smaller scale. Also, the separation of the TBox and ABox is much clearer. Knowledge graphs on the other hand are more flexible: adding a new type or a new instance is just adding a node to the graph. The flexible data model allows knowledge graphs to integrate heterogeneous sources and expand rapidly.

The differences between knowledge bases and knowledge graphs (particularly the size and heterogeneity) call for a new perspective on knowledge engineering. Unlike traditional knowledge engineering methodologies, building knowledge graphs focus on large ABoxes and not complex TBoxes. Due to the heterogeneity of data sources, quality assessment, and improvement via correctness and enrichment is also under focus.

In this paper, we present our methodology for building knowledge graphs (Fig. 1) that consists of:

- a Knowledge Creation step that focuses on the efficient creation of large ABoxes and lightweight ontology engineering based on Schema.org for TBoxes
- a Knowledge Hosting step that considers provenance of the created knowledge from heterogeneous sources and contextualization of knowledge

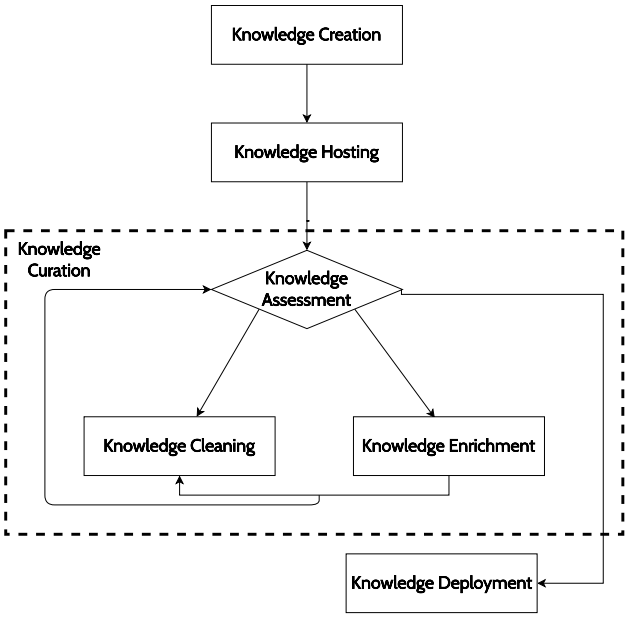

Fig. 1 A methodology for building knowledge graphs

- a Knowledge Curation step that aims to assess, correct, and enrich a knowledge graph. Quality issues in knowledge graphs are inevitable due to the large size and heterogeneity. The assessment task evaluates the quality in various dimensions. Then, the quality is improved particularly in the correctness and completeness dimensions by cleaning and enriching the knowledge graph, respectively.
- a Knowledge Deployment step that puts the knowledge graph in use

This paper is a compiled and extended version of our previous work such as [3, 4]. The main contribution of the paper is to take a look on traditional knowledge engineering from knowledge graph perspective. We identify challenges that emerge due to the aforementioned characteristics of knowledge graphs and propose various solutions to different knowledge graph lifecycle tasks. The remainder of the paper first presents a subset of the knowledge graph which we use as a running example (Section "Running Example"). We then present each step of our methodology starting from Section "Knowledge Creation" until Section "Knowledge Deployment". We introduce the German Tourism Knowledge Graph (GTKG) that is being built with the knowledge graph lifecycle in Section "Building the German Tourism KnowledgeGraph". We discuss the related work in Section "Related Work" and conclude with a summary and future work in Section "Conclusion and Future Work".

[1] Terminological Box (TBox) defines the terminology (e.g.,types, properties) and other logical formulas used in a knowledge graph.

[2] Assertional Box (ABox) refers to the facts that are described by the TBox in the knowledge graph.

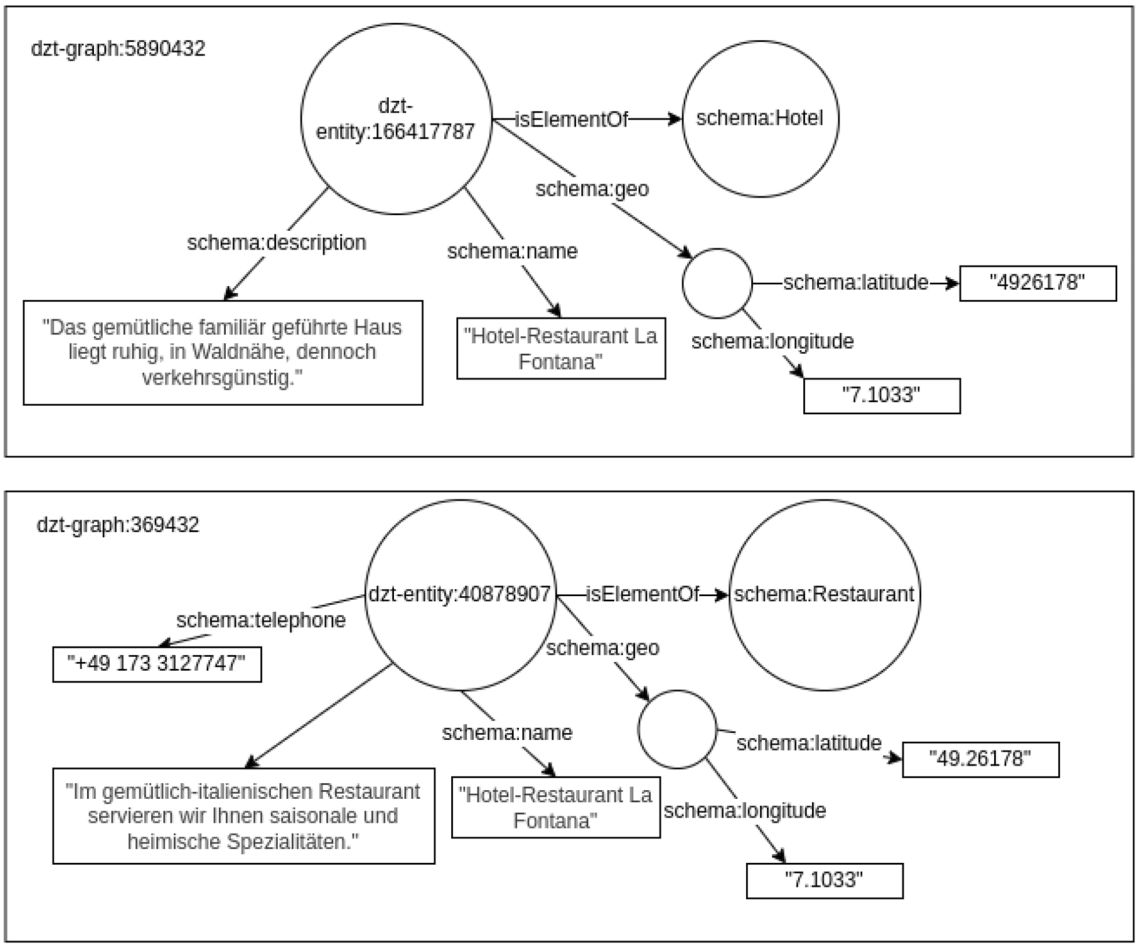

Fig. 2 An excerpt from the GTKG

Running Example

As a running example to demonstrate different steps of our methodology, we use a subset of the German Tourism Knowledge Graph (GTKG). A detailed description of the knowledge graph will be given in Section 7.

Figure 2 shows two named graphs identified with URIs dzt-graph:5890432 and dzt-graph:369432[3,4] The first graph contains a schema:Hotel instance (dzt-entity:166417787) that is described with the properties schema:description, schema:name and schema:geo. The second named graph contains a schema:Restaurant instance (dzt-entity:40878907) that is described with the same properties as the schema:Hotel instance, and additionally with the schema:telephone property. In the remainder of the paper, we will refer to this running example to explain the various processes in the knowledge graph lifecycle.

Knowledge Creation

Knowledge creation refers to semantically annotating content, data, and services from heterogeneous sources with an ontology. This step can be split into two major tasks, given the nature of knowledge graphs:

- *Bottom-up* A lightweight ontology engineering task called domain specification that creates domain-specific patterns.
- *Top-down*: A large scale instance generation task as an application of the domain-specific patterns.

In the following, we will explain these two tasks and our approach to knowledge creation (Fig. 3).

[3] Assume dzt-graph is the namespace of the GTKG.

[4] The rectangular boxes show literal values. Empty circles represent entities without identifiers. The edges represent the relationship between two entities.

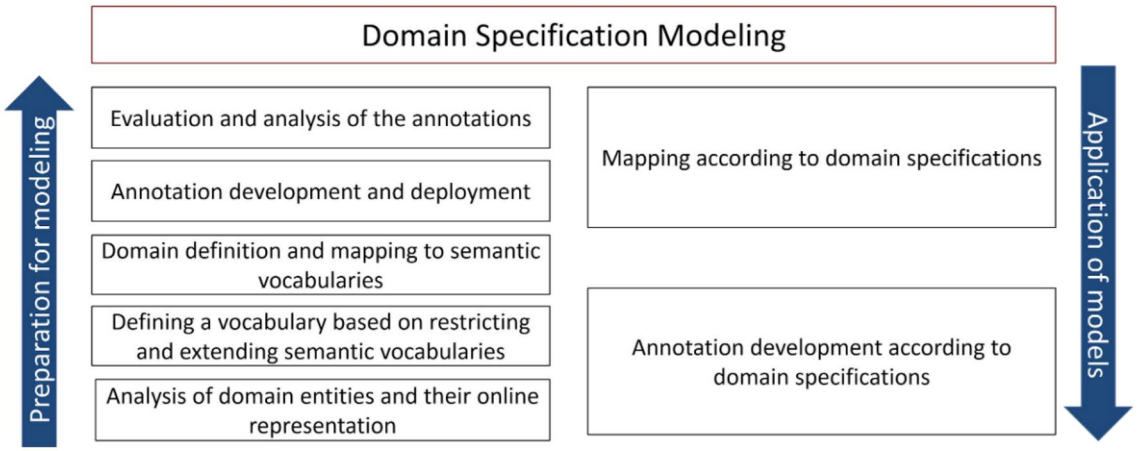

Fig. 3 Knowledge creation approach

Bottom-Up: Domain Specification Modeling

We take Schema.org as a reference ontology for knowledge graphs as it is a de-facto industrial standard for annotations on the web. These annotations are natural building blocks for knowledge graphs and usage of such a widespread ontology would increase the impact of a knowledge graph. Before the knowledge generation process starts, the domain(s) the knowledge graph is supposed to describe must be analyzed to figure out the domain entities and their relationships and then mapped to Schema.org. This process is quite challenging due to the nature of the vocabulary. Schema.org covers many domains with hundreds of types and properties, however, its coverage for specific domains is very shallow. This situation calls for an adaptation of Schema.org for specific domains and tasks. We call this adaptation process *domain specification* [5].

The domain specification process consists of two major operations. The **reduction** operation aims to eliminate the types and properties that are not relevant to a specific domain. Afterward, the ranges of the remaining properties can be also restricted when necessary. The **extension** operation aims to extend the remaining part of the vocabulary with domain-specific types and properties. The result of this process is a set of SHACL shapes that are called domain specifications. These operators are then used to guide the instance generation task.

Top-Down: Application of Domain Specifications

This task involves instance generation via the application of the generated domain-specific patterns to heterogeneous sources. The instance generation process can be done manually, semi-automatically via supervised methods and declarative mappings, or fully automated. The manual creation of knowledge is typically suitable for annotation of content (e.g. text, images) on a small scale. For this task, we adopt form-based editors that are generated based on domain-specific patterns. Similarly, content can be annotated semi-automatically by applying a supervised NLP pipeline, particularly for Named Entity Recognition and Relation Extraction where the domain-specific patterns provide the template to be filled. Finally, content can be also automatically annotated in an unsupervised manner. The research on this direction mostly deals with the open-domain information extraction (e.g. OpenIE [6]). Most of the useful applications however require some sort of human intervention at the end (e.g., NELL [7]) to improve the precision.

Alongside unstructured content, the majority of the data comes in a semi-structured format (e.g. JSON, XML, CSV) typically through third-party APIs. The semantic annotation of such data can be done programmatically, however, this approach does not scale as the number of different data sources increase. We benefit from declarative mapping approaches. There are many approaches; like SPARQL-Generate which extends SPARQL language as a mapping language, and R2RML[5]-based languages such as RDF Mapping Language (RML) [8] and its derivations. Our knowledge creation approach makes use of RML due to its rather widespread adoption and continuous development. We developed a mapping engine for RML called RocketRML. It is implemented in NodeJS and shows particularly high performance with data in JSON format (see [9] for performance details for populating a large knowledge graph). Being an extension to R2RML, RML also works with multiple sources via JOIN operations that are done on a set of fields on each source that serve as primary and foreign keys.[6] RocketRML

[5] https://www.w3.org/TR/r2rml/.

[6] RML now also has extension like RML Fields in works to support nested objects without JOIN fields [10].

implements some optimization via memoization of JOIN operations and can handle nested objects without any fields to JOIN. RocketRML is open-source and being maintained actively by the community.[7] The details of the implementation of our knowledge creation approach can be found in [9].

In our running example, the knowledge creation process is conducted in a distributed manner. The domain-specific patterns for hotels and restaurants are created and published by tourism domain experts and the technology providers create mappings from their metadata to the domain-specific patterns. The mappings and data are then processed by RocketRML and instances in the RDF data model are produced.

Knowledge Hosting

The previous section presented different methodologies for generating knowledge for knowledge graphs. Hosting those knowledge graphs raises various challenges (Section "Challenges"). Those challenges significantly affect the decision for a database paradigm to host knowledge graphs. In Section "Database Paradigms for Hosting Knowledge Graphs", we will discuss different ways of hosting knowledge graphs using different kinds of databases and the ones we adopt in our use cases.

Challenges

Hosting knowledge graphs brings several challenges with it. We identified the following challenges that need to be considered:

Size

A major characteristic of a knowledge graph that distinguishes it from traditional knowledge bases is its massive size which can reach billions of facts. The database must be able to support operations of such a large size.

Data model

Knowledge graphs have conceptually a graph structure. From a storage perspective, the TBox and ABox are not strictly separated, they are all just nodes and edges of a graph. The database paradigm must support the graph data structure properly.

Heterogeneity of sources

Typically, knowledge graphs are built from heterogeneous sources where each source uses a different format for representing its knowledge. Related to the challenge above, the database paradigm must support the flexible expansion of the knowledge graph when a new type, property, or assertion is added.

Points of view

Knowledge graphs are typically not built for a particular use-case but are used for several use-cases coming from various applications. The different use-cases will likely have other requirements concerning the usage of a knowledge graph. Those requirements can also be conflicting, e.g., one use-case requires that local businesses have a single address only, whereas another use-case allows multiple addresses.

Database Paradigms for Hosting Knowledge Graphs

Knowledge graphs can be hosted using various database paradigms, such as relational databases, document stores, and graph databases.

We examine three major paradigms that are widely adopted in practice to see how well they are suitable for the knowledge hosting task. Relational databases can host knowledge graphs via different approaches such as the statement table approach where triples are stored in a single table, the class-table approach where each type corresponds to a table and the property-table approach where each property corresponds to a table. Each of these approaches has various drawbacks such as requiring expensive self-joins or recompilation of the schema anytime a type or property is introduced to the knowledge graph. Alternatively, ontology-based access to relational databases (i.e., Virtual RDF Graphs) can be used, however, they also need complex mappings and middle layers to support querying and reasoning.

Document stores are more flexible in terms of schema compared to relational databases. However, they suffer from the mismatch between the graph nature of knowledge graphs and the document-based paradigm as they require either joins between different collections or nested storage of instances which make it cumbersome to delete and update the data. Moreover, like relational databases, they lack native reasoning support.

Given the limitations above, an obvious choice for hosting knowledge graphs is graph databases. Many enterprise graph database solutions can manage the large size of knowledge graphs. A graph in a graph database is represented by *nodes* and *edges*. Nodes represent entities and edge the relationships between those entities. This provides the flexibility needed for accommodating expanding nature of knowledge graphs. Two data models are currently dominating the knowledge graph scene: *Property Graphs* and native *RDF Graphs*.

Nodes in Property Graphs represent entities in the graph and hold any number of property assertions. A label can be used to describe the node's role within the graph. Relationships represent a connection between two nodes, specifying

[7] https://github.com/semantifyit/RocketRML.

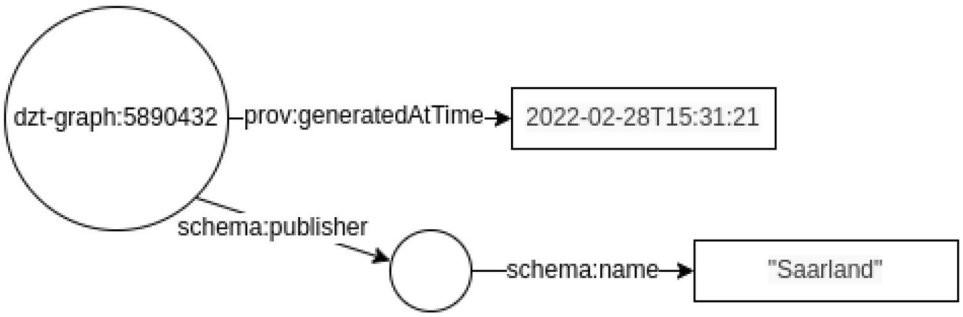

Fig. 4 Provenance information attached to a named graph

the relationship between those. A relationship always contains a direction given by a start and end node. Besides, relationships in Property Graphs can have property value assertions assigned.

Native RDF Graph databases or Triplestores support the RDF data model. RDF is triple-based and by default supports only binary relationships. This normally gives an advantage to Property Graphs as they can natively store statements about statements by attaching property-value pairs to the relationships between two entities. This capability allows seamless attachment of provenance information to the statements in a knowledge graph. There are, however, several methods used to reify RDF statements to mitigate this shortcoming of the RDF model. Two of them are Named Graphs [11] and a recent extension called RDF-Star [12]. Named Graphs extend the standard RDF triple model with a fourth element called "context" and make it a quad model. This context element can be the subject of other triples, which allows making statements about statements. The statements that share the same context URI are grouped into the same conceptual subgraph. RDF-Star provides a mechanism to create nested triples. A triple can be embedded into another triple as a subject which allows again making statements about statements.

In the end, from the modeling point of view, there is not much difference between RDF triplestores and property-graph databases. Triplestores are built on solid recommendations from W3C which leads them to have a standard query language and reasoning support via RDFS and OWL2. Property graphs are still quite vendor-specifically implemented. Therefore, we make use of RDF Graph Databases to host our knowledge graphs.

As a reification technique, we use Named Graphs as it is built into the SPARQL standard and supported by many industrial Triplestores out-of-the-box. We use named graphs to group statements into subgraphs based on their provenance and attach license information, tempo-spatial scope, and more. In our running example, we have two named graphs. These named graphs contain knowledge created based on the data coming from different providers. Additional provenance information is attached to these named graphs via using the named graph URIs as subjects and describing them with properties from ontologies like PROV-O[8] and Schema.org. In Fig. 4 we demonstrate provenance information attached to dzt-graph:5890432.

Although graph databases address many challenges of hosting knowledge graphs, they are still not the end of the story. Named graphs are still part of a knowledge graph physically. This makes it quite challenging to support different points of view that may involve possibly conflicting constraints and inference rules. Moreover, the curation operations explained in the following section are still done over a large knowledge graph which may not be as efficient as applications need. These challenges and our solution are discussed further in Section "Knowledge Deployment".

Knowledge Curation

Due to its large size and heterogeneity, a knowledge graph may have quality issues to be solved to become a useful resource for applications. Therefore, knowledge graphs must go under a series of operations that together comprise the knowledge curation process. In the following we will introduce these three major processes for assessing an improving the quality of knowledge graphs.

Knowledge Assessment

Knowledge assessment aims to evaluate the quality of knowledge graphs. The literature on quality assessment can be traced back to data quality research. Data quality is often defined as fitness for use and evaluated from various dimensions. In the 2000s these definitions have been adapted and updated across the Linked Open Data community, particularly with dimensions specific to the nature of linked data on the web (e.g., how well the data is interlinked, are the sources verifiable, is the dataset logically consistent) [13–15].

For assessing knowledge quality, there are already several frameworks with a specific degree of mechanization

[8] https://www.w3.org/TR/prov-o/.

and dimension focus. For instance, some frameworks can assess the quality of knowledge graphs manually (e.g., Sieve [16]), by crowdsourcing (e.g., TripleCheckMate [17]), via external knowledge (e.g., RDFUnit [18]) and semi-automatically (e.g., Luzzu [19]). Most of the frameworks have their source code available, however, they are no longer maintained, except for Luzzu and RDFUnit, which are distributed with GPL-3.0 and Apache-2.0 licence, respectively. More recently approaches that target the assessment of declerative mappings instead of the entire knowledge graph appeared [20, 21]. These approaches bring significant performance advantages; however, they can only be used to evaluate certain dimensions and metrics.

In the remainder of the section, we will first introduce the dimensions, metrics and their calculation functions, which are cornerstones of defining different aspects of quality and operationalizing the quality assessment process. Then, we dive deeper into two selected dimensions and their metrics, namely correctness and completeness as they are the aspects of knowledge graphs we are mainly improving with the curation process. Finally, we will present how we calculate an aggregated quality score for a knowledge graph.

Quality Dimensions, Metrics and their Calculation Functions for Knowledge Graphs

A **quality dimension** speficies a criterion for which the knowledge graph quality can be assessed. Based on the data quality literature, we can identify 23 dimensions that can be applied on knowledge graphs [3]. These dimensions are Accessibility, Accuracy (Correctness), Appropriate amount, Believability, Completeness, Concise representation, Consistent representation, Cost-effectiveness, Ease of manipulation, Ease of operation, Ease of understanding, Flexibility, Free-of-error, Interoperability, Objectivity, Relevancy, Reputation, Security, Timeliness, Tracability, Understandability, Value-addedness and Variety. For example, *Accessbility* dimension specifies the criterion that "data or part of it must be available and retrievable."

Each dimension has a certain number of **metrics** that amount to 42 metrics in total. The metrics of a dimension formalize the criterion and enable the assessment of the extent a knowledge graph fulfills the criterion of that dimension. For example, Accessibility dimension has a metric called "Availability of a knowledge graph" which evaluates the ability of derefencing URIs in a knowledge graph.

Each metric has a measurement function that allows to calculate a value for the metric. A measurement function returns a value between 0 and 1. For example, "Availability of a knowledge graph" metric ($m_{\text{availability}}$) can be calculated with the following function, given a knowledge graph k

$$m_{\text{availability}}(k) = r_{\text{succ}}/r_{\text{all}}, \quad (1)$$

where r_{succ} is the number of successful requests sent to a sample of URIs in the knowledge graph in a time frame and r_{all} is the number of all requests sent.

The impact of the each metric for the assessment of a dimension, and each dimension to the assessment of the overall quality of a knowledge graph may be different based on the domain an application. For instance, in case of a knowledge graph for financial and medical applications correctness and reliability may be more important than completeness. Therefore, we propose a weighted approach that allows defining weight coefficients for individual dimensions and metrics per domain. This allows stakeholders to determine what dimension and metric are important for a given domain (e.g. events, POIs, accommodation).

Selected Dimensions: Correctness and Completeness

Among the 23 dimensions, we select correctness and completeness to explain in more detail, as they are the core dimensions targeted by the knowledge curation process. We will introduce both dimensions including the metrics they consist of the calculation functions for those metrics.

Correctness

The correctness dimension (may also be known as Accuracy) assesses the degree of which the knowledge graph is syntactically and semantically correct. The dimension consists of four metrics:

Metric: Syntactic correctness with regards to RDF evaluates whether the RDF dumps of a knowledge graph have valid RDF syntax. *The calculation function* of this metric is returns the ratio of valid RDF dump documents exported from a knowledge graph to all RDF dump documents exported from a knowledge graph. This function can be implemented rather straightforwardly via an RDF syntax verifier.

Metric: Syntactic correctness of literal values evaluates whether the literal property values in a knowledge graph obey certain syntactical rules. *The calculation function* of this metric returns the ratio of triples whose literal-valued objects obey to certain syntactical rules among all triples with literal-valued objects. The calculation function can be implemented via regular expressions, via checking against the lexical space of their datatypes and known reference sources via string similarity measurements.

*Metric: Syntactic correctness of resource identifiers*evaluates whether the resource identifiers used in a knowledge graph are syntactically correct. *The calculation function* of this metric returns the ratio of syntactically valid resource identifiers to the all resource identifiers used in the knowledge graph. The metric can be implemented by checking

URIs against vocabularies used in the knowledge graphs via string similarity measures.

*Metric: Semantic correctness of statements*evaluates whether the statements (assertions in the ABox) in a knowledge graph are correct in terms of a formal specification or the domain they describe. *The calculation function* of this metric returns the ratio of semantically correct statements to the all statements in a knowledge graph. The implementation of this metric can be done in various ways such as statistical methods like outlier detection for seeing if an instance assertion is correct based on property distributions (e.g. [22]), checking against formal constraints (e.g., domain-specifications in the form of SHACL constraints) or crowdsourcing (e.g. collecting feedback from people whether the phone number of a business is correct).

Completeness

The completeness dimension asseses the degree of which a knowledge graph contains the sufficient and necessary statements for a domain and application in hand. This definition already makes assessing the completeness of a knowledge graph subjective and to automate the measurement of metrics a formal description of what *complete* means must be available. In the data quality literature (including LOD) completeness is typically assessed in terms of an ontology or a gold standard dataset [14]. This approach is problematic because it ignores the use case and application context aspect of the dimension. For example, not all properties of a type defined in an ontology may be relevant for an application or finding a golden standard dataset may be challenging for domain-specific knowledge graphs. We propose the following metrics:

Metric: Coverage of a domain by a knowledge graph evaluates whether a knowledge graph is complete in terms of the requirements of a given domain. *The calculation function* returns the ratio of average number of properties used on the instances of a type to all properties of that type defined by a domain expert. This metric can be implemented with the help of the domain specific patterns introduced in the Knowledge Creation process. These patterns generated by domain experts specify an extend subset of schema.org which can be used to determine the properties of a type that is required to consider an instance complete.

Metric: Completeness of a knowledge graph for an application evaluates whether a knowledge graph is complete in terms of the requirements of an application. *The calculation function* returns the ratio of successful queries to a knowledge graph to all queries by an application. The metric can be implemented with the help of examining the query logs of the knowledge graph.

Weighted Aggregate Quality Score Calculation

After the scores for all metrics for all dimensions are calculated via their functions, a weighted aggregate quality score is obtained. Given a knowledge graph k the formula below is used to calculate the weighted aggregate quality score of the ith dimension for k, $d_i(k) \in [0,1]$, where p_i is the number of metrics in the ith dimension, $m_{i,j}(k) \in [0,1]$ is the score of the jth metric of the ith dimension for k, and $\alpha_{i,j}$ is the weight of the jth metric of the ith dimension and $\sum_{j=1}^{p_i} \alpha_{i,j} = 1$.

$$d_i(k) = \sum_{j=1}^{p_i} m_{i,j}(k).\alpha_{i,j} \tag{2}$$

Once the scores for all dimensions are calculated, the weighted aggregated quality score for a knowledge graph $T(k)$ is calculated with the following formula, where n is the total number of dimensions, and β_i is the weight of the ith dimension and $\sum_{i=1}^{n} \beta_i = 1$.

$$T(k) = \sum_{i=1}^{n} d_i(k).\beta_i \tag{3}$$

Assume we have both instances in our running example in the same sample. We want to calculate the syntactic correctness of literal values. We realize that one of the geocoordinates (4926178) does not fit the syntactic structure of WGS84 format[9] as it is missing a decimal point. If these two instances are only instances in the sample, then there are 9 literal values in total. The ratio of correct literal values to all literal values would amount to 0.89.

An important thing to keep in mind about knowledge assessment is that its main purpose is to indicate the overall quality of a knowledge graph. For example, assessing the correctness of a knowledge graph is not about identifying individual errors, but assessing a representative sample whose evaluation gives an idea whether the correctness is at the desired level. In the next two chapters, we will particularly focus on what to do, when the quality is low in completeness and correctness dimensions.

Knowledge Cleaning

Knowledge Cleaning is a part of Knowledge Curation that aims to improve the correctness of knowledge graphs. Knowledge cleaning consists of two subtasks, namely error detection, and error correction. In the following, we define the types and sources of errors in a knowledge graph, the

[9] https://en.wikipedia.org/wiki/World_Geodetic_System.

approaches, and tools that tackle the knowledge cleaning task from the literature, and our approach to the task.

Like any other knowledge curation task, the cleaning task is targeting the assertions in the ABox. Errors may therefore stem from wrong instance assertions, wrong equality assertions, and wrong property-value assertions. The knowledge cleaning task deals with two types of errors namely syntactic and semantic errors. Aligned with the definitions made for the correctness dimension in the knowledge assessment task, syntactic errors occur due to identifiers or literal values not following certain structure and syntax rules. Semantic errors occur either due to violation of formal specifications or the domain the knowledge graph is describing.

There are many approaches and their implementations in the literature targeting the knowledge cleaning task. For **error detection**, most popular approach is to use integrity constraints (e.g., RDFUnit [18]) for detection of semantic errors. Traditionally these constraints were represented by SPARQL queries, but in the last 5 years, SHACL is the most popular one as it is a W3C recommendation and supported by many triple store implementations. Additional approaches for detection include statistical analysis of the distribution of types and properties to identify semantically wrong instance assertions and domain range violations (e.g., [23]) and using logic to detect semantically wrong equality assertions (e.g., [24]). Once errors are detected, they can be corrected by either removing wrong assertions or adding new assertions to make the wrong statements fit the formal specifications. LOD Laundromat [25] and more recently RDFDoc is a tool that fixes syntactic errors in RDF data by using various heuristics. Other approaches like HoloClean [26] and KATARA [27] combine external knowledge sources, crowdsourcing, and statistical methods. In the recent years, rule-based approaches (e.g., [28]) and machine learning-based approaches that benefit from knowledge graph embeddings (e.g., [29]) also emerged.

Our approach to **knowledge cleaning** focuses on the detection task. We developed VeriGraph, a tool to verify instances in knowledge graphs based on domain-specific patterns encoded as SHACL shapes. From a functionality point of view, VeriGraph adopts an integrity constraint checking approach. It can work directly on SPARQL endpoints. It contains certain optimizations that allow scalable operation on a knowledge graph. The major improvement is that the knowledge graph is not verified by checking each constraint in the shape one by one but on instances that are cached locally. This reduced the number of SPARQL queries run on the triple store the knowledge graph is stored, therefore decreasing the overhead significantly. The details about the performance can be found in [30].

We can demonstrate how error detection and correction work with our running example as follows: The error detection tool finds out that the value of schema:latitude property attached to dzt-entity:166417787 does not fit the structure of the WGS84 format based on a regular expression defined on a SHACL shape, as it is missing a decimal point. Once the error is detected, the correction process tries to place a decimal point by following a predefined heuristic for geocoordinates. Given that the latitude value can be between – 90 and 90, there are only two places the decimal point can go. A decimal point after the first digit would lead to 4.926178 and after the second digit to 49.26178. Given the heuristic that the knowledge graph contains hotels within Europe, the first option would be eliminated as combined with the longitude it refers to a point around Nigeria.

It is not a surprise that most of the approaches for knowledge cleaning focus on the detection task rather than the correction task. This is related to various challenges of correction. It is typically very challenging to correct semantic errors as there are not many ways to automate finding out the real phone number of a business and correct the wrong one, without actually calling the business. Therefore, many approaches have to work with external sources that are assumed to be reliable. This is not particularly easy, especially for domain-specific knowledge graphs. Another option is to go with a crowdsourcing approach where the users of the knowledge graph have the opportunity to detect errors in assertions and suggest fixes. Nevertheless, the knowledge cleaning task, particularly automated correction remains an interesting research topic.

Knowledge Enrichment

Knowledge Enrichment (KE) is the task of adding missing knowledge to KGs to improve their completeness. Completeness can be increased by adding missing instance assertions, identity assertions (so-called sameAs links), or other property value assertions that connect arbitrary instances to instances and literals. Increasing the completeness of a knowledge graph mainly implies the integration of external sources into a knowledge graph. Once the sources to integrate are identified, the enrichment process continues with three major steps [31]: (1) Merging of different schemata. This implies the mapping of the schema of an external source to the schema of the knowledge graph. (2) Identifying and resolving the missing assertions (e.g. identifying a hotel is also a restaurant, connecting two Points of Interest that have a containment relationship). (3) Resolving any conflicts that were introduced after new assertions (e.g. violation of disjointness between types after instance assertions, having multiple distinct values for a property with singular cardinality after identity assertions between two instances).

In our work, we currently focus on identity assertions. The creation of identity assertions is commonly referred to as the Duplicate Detection (DD) task. Identity assertions

declare that two instances of a knowledge graph describe the same object of discourse.

Knowledge enrichment, especially duplicate detection is a challenge that is at least as old as computer science itself, therefore many approaches have been developed. In the context of knowledge graphs and traditionally Linked Open Data there have been approaches like Duke [32], SILK [33], LIMES [34] that use similarity functions supported by user-created heuristics such as identification of important properties, like EAGER [35] that benefit from pre-trained vector models of knowledge graphs and approaches that benefit from knowledge graph embeddings (e.g., [36, 37]).

Our approach currently focuses on the first two steps of a typical enrichment process and follows a simple workflow that is frequently used in research for this area. The workflow is roughly divided into five steps:

1. Mapping schemata.
2. Indexing instances.
3. Applying pre-filtering.
4. Comparing instances.
5. Applying decision model.

In the following we will briefly explain our methodology for duplicate detection. The details about the implementation can be found in [38].

Mapping schemata The schemata mapping process corresponds to the ontology alignment task which is already heavily researched and it is another beast by itself. Since we assume Schema.org as the reference ontology for knowledge graphs, such an alignment is only done when an external knowledge source is being integrated, and even then only partially for the relevant part of the ontology.

Indexing instances

After the schemata are mapped, the knowledge graphs whose instances are going to be linked must be indexed. For our knowledge graphs, we use Elasticsearch as an indexing technology. Triplestores like GraphDB provide connectors to indexing services like Elasticsearch which automatically index instances for a given set of types and specified set of properties and/or property paths. This enables our DD process to quickly search for similar instances among the graph data.

Applying pre-filtering

Pre-filtering approaches are used to remove the most obvious non-duplicate instances for a given sample instance from the list of candidate duplicates as fast as possible. This is done to counteract the quadratic complexity of DD tasks. For this purpose, we make use of the `more_like_this` query function that is provided by Elasticsearch. Other indexing solutions also provide similar functionalities. It allows for vastly different grades of strictness when creating candidate lists for an instance.

Comparing instances

After the candidate list for an instance was created, we dedicate more resources to the comparison of each candidate to the instance. We compare instances, property by property where we have to differentiate between three types of property value comparisons:

1. *Literal to literal* we defined multiple comparisons and standardization functions for a subset of data types in the XML Schema Language. Comparison functions assign a value in the range of 0–1 to a pair of literals. This value could stem from the ratio of two numbers (e.g., literals of datatype `xsd:unsignedInt`) or the Levenshtein distance between to words. Optional standardization functions aim to make literal values more comparable (e.g., date standardizer which translates months in their alphanumerical form to the respective numerical form).
2. *Instance to literal* Property values that can be represented by literals and instances usually provide the same information in both forms. For the instance form, the information is likely distributed across multiple properties. To make these comparable, we apply a recursive serialization function that constructs a literal value representation for an instance to make them comparable with the functions for literal to literal comparison.
3. *Instance to instance* We have multiple options for this type of comparison: (a) We could compare the URIs that are used to identify the instances. This would make this type of comparison a literal to literal comparison. (b) We can apply the recursive serialization function, we already described in comparison type 2 and apply it to both instances. (c) Their properties are compared individually, and their similarities are aggregated. This is the most precise way of comparing instances but also requires the most effort in run-time, configuring, and understanding.

This differentiation is necessary since properties can have multiple disjunctive types for their range as it is the case with schema.org.

Applying decision model

After the instance comparison is complete, we apply a decision model that decides whether two compared instances are duplicates. A weighted aggregation of the properties' similarities is used to compute an overall similarity score. A weight describes the ability of a property to contribute to identifying an identity link between two instances. We simply use a similarity threshold to

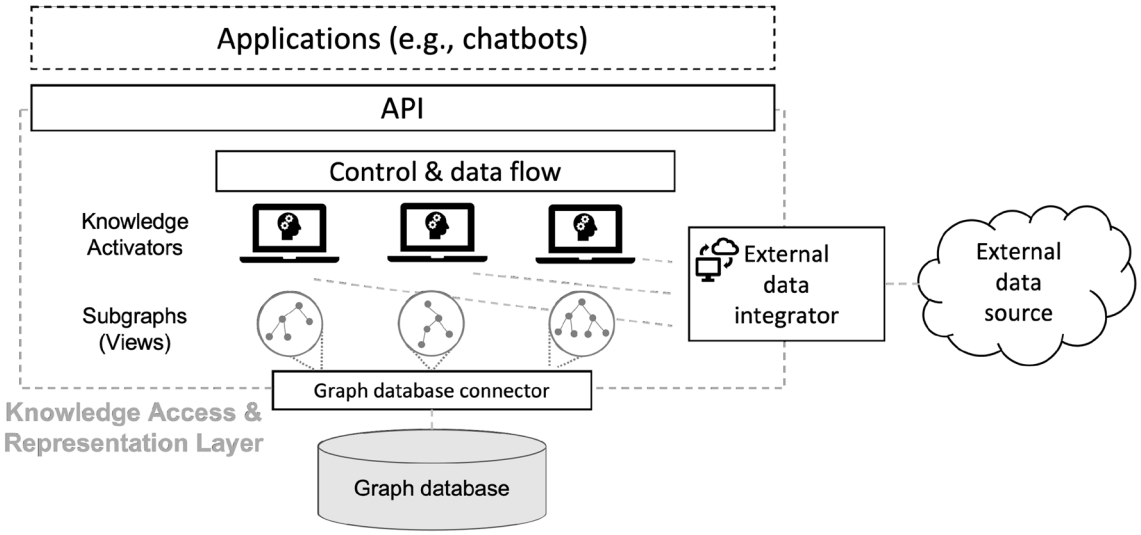

Fig. 5 Knowledge Access & Representation Layer Overview

determine whether a pair of instances should be considered for identity link creation.

In our running example, there are two instances of schema:Hotel and schema:Restaurant respectively. Given that schema:Hotel and schema:Restaurant are both subtypes of schema:LocalBusiness, the duplicate detection approach selects them as candidates after pre-filtering. Assuming the configuration for local businesses consider the properties schema:description, schema:geo, schema:name and schema:telephone. These properties have different weights for influencing duplicate detection (e.g. geo coordinates have a higher effect while description has a lower one). For schema:geo, a similarity metric based on Haversine distance, and for other properties string similarity metrics can be adopted. Due to a high similarity (i.e., same name, same geo-coordinates), these two instances are identified as duplicates and an identity link between dzt-entity:166417787 and dzt-entity:40878907 is created. This link improves completeness from different perspectives. For example, now we know that both are instances of schema:Hotel and schema:Restaurant, and the phone number of the hotel is completed with the property value from the restaurant. The next step would then be finding a strategy for resolving conflicting properties. Here there will be multiple descriptions for both instances. The enrichment process can select one of them or keep both, depending on the constraints of the domain and task. An important consideration here is how knowledge cleaning and enrichment interplay. On one hand, the enrichment process may not perform very well, if the geocoordinate is not corrected beforehand; on the other hand, the conflicting property values may be caught by the cleaning process.

Knowledge Deployment

Knowledge graphs can power a variety of applications like web search, Intelligent Personal Assistants, and autonomous agents. When deploying a knowledge graph there can be challenges due to its nature. These challenges are mainly related to the **size** of knowledge graphs and **various points of view** of applications. The curation tasks may take too long on large knowledge graphs, and the applications that give real-time responses may not afford to wait until the curation tasks are complete. Even with a simple formalism like RDFS, tasks like reasoning with domain and ranges and query answering run into problems operating on a knowledge graph with around 130 million triples.[10] In addition, different use cases might have different (contradictory) rules and constraints for the underlying data. Representing those in a single knowledge graph is impossible.

The state of the art approaches typically focus on the optimization of query performance when it comes to knowledge deployment. There are approaches that aim to balance SPARQL query processing between clients and servers (e.g., WiseKG [39]) and approaches that benefit from optimization techniques of relational databases for graph querying [40].

We propose introducing an intermediate layer between the knowledge graph and applications called Knowledge Access and Representation Layer (see Fig. 5) to tackle the challenges above. The knowledge graph as the basis for the layer is treated as a data lake allowing it to be erroneous and incomplete. With Knowledge Activators at its core, the layer on top operates on use-case-specific subgraphs (called

[10] See the benchmark results at https://github.com/kev-ang/mskr.

views) of the knowledge graph. Each view is attached to a knowledge activator, which contains a specific TBox that serves the needs of specific application contexts. The Knowledge Access and Representation Layer addresses both size and point of view issues: the former is addressed via the small size of views that allow efficient knowledge curation, and the latter is addressed via the Knowledge Activators that can support different constraints and inference rules for the same data. At the same time, the newly introduced layer allows the integration of external data on the fly. Working on materialized subgraphs of knowledge graphs has been investigated [41], however not significantly implemented in paractice. We developed a GraphQL based interface that can be used for view extraction [42]. In the next steps, this interface will be extended with RML mapping capabilities to enable mapping between the TBox of the knowledge graph and the use case specific TBox in the knowledge activators.

In our running example, different applications may have different requirements in terms of the format of geocoordinates. One application may work with coordinates in WGS84 format, another one British Nation Grid (BNG)[11] system. Accommodating the needs of both applications is only possible if we can define custom syntactic and semantic constraints for different applications via Knowledge Activators.

A control and data flow component based on Abstract State Machines handle the data flow between Knowledge Activators. Finally, an API allows various applications to access the Knowledge Access and Representation Layer. The implementation of the layer is currently still in progress.

Building the German Tourism Knowledge Graph

The German Tourism Knowledge Graph (GTKG) is at the core of the Open Data Germany[12] initiative led by the German National Tourism Board. The main goal of the project is to integrate and curate data from all regional tourism marketing organizations in Germany to provide a high-quality knowledge source for intelligent applications and improve the positioning of German tourism in Europe and beyond.

The knowledge graph is built around several main concepts that are particularly important for German tourism such as POIs, Tours, Events, Accommodation, Gastronomy, and Holiday Offers. It integrates data from more than 16 heterogeneous sources.

The initial deployment of GTKG contains more than *20K Event*, more than *20K POI* and more than *5K Tour* instances. These instances amount a total of 7.5M statements. The schema used to describe these instances are aligned with the schemas developed by the Open Data Tourism Alliance (ODTA). ODTA is an initiative that brings tourism domain experts from the German-speaking area of Europe (DACH region), namely Austria, Germany, Italy (South Tyrol), and Switzerland. These domain experts together with semantic technology experts create domain-specific patterns for the DACH region as a de facto standardized data model. ODTA also provides tutorials and supporting material for IT solution providers for the usage of domain-specific patterns for knowledge creation.

Different aspects of building knowledge graphs presented in this paper are being implemented in a toolkit by Onlim, a company that deals with knowledge graphs and applications powered by knowledge graphs. This toolkit is called Knowledge Graph Management Platform (KGMP) and is currently primarily used for building the GTKG. The KGMP provides tooling for knowledge creation based on domain-specific patterns created by ODTA, hosting with provenance tracking via named graphs powered by GraphDB, curation that focuses on correctness and completeness dimensions for assessment, and error detection for cleaning and assertion of identity links for enrichment. For deployment, the platform provides query interfaces and APIs for various applications and a web application for browsing the knowledge graph. The platform is under ongoing development and tooling for the remaining tasks mentioned in the paper is being implemented.

The knowledge graph is currently in the pre-production phase (Fig. 6) and will be publicly available with an open license in May 2022. The current number of 7.5M statements is rapidly increasing as more regional marketing organizations join the knowledge creation process. The progress can be tracked on the Open Data Germany website.[13]

Related Work

The knowledge graph building process can be compared with the traditional knowledge engineering in the core of which ontology engineering lies. There are many ontology engineering methods with various nuances between them. For example, some focus on collaboration and distributed development while others focus on iterative development and modularization. A recent survey covers many of these approaches in detail [43]. Despite their nuances, the traditional knowledge engineering methodologies typically follow a create-develop-evaluate-update-deploy approach. This is comparable with our process model for building

[11] https://en.wikipedia.org/wiki/Ordnance_Survey_National_Grid.

[12] https://open-data-germany.org/.

[13] https://open-data-germany.org/datenbestand/.

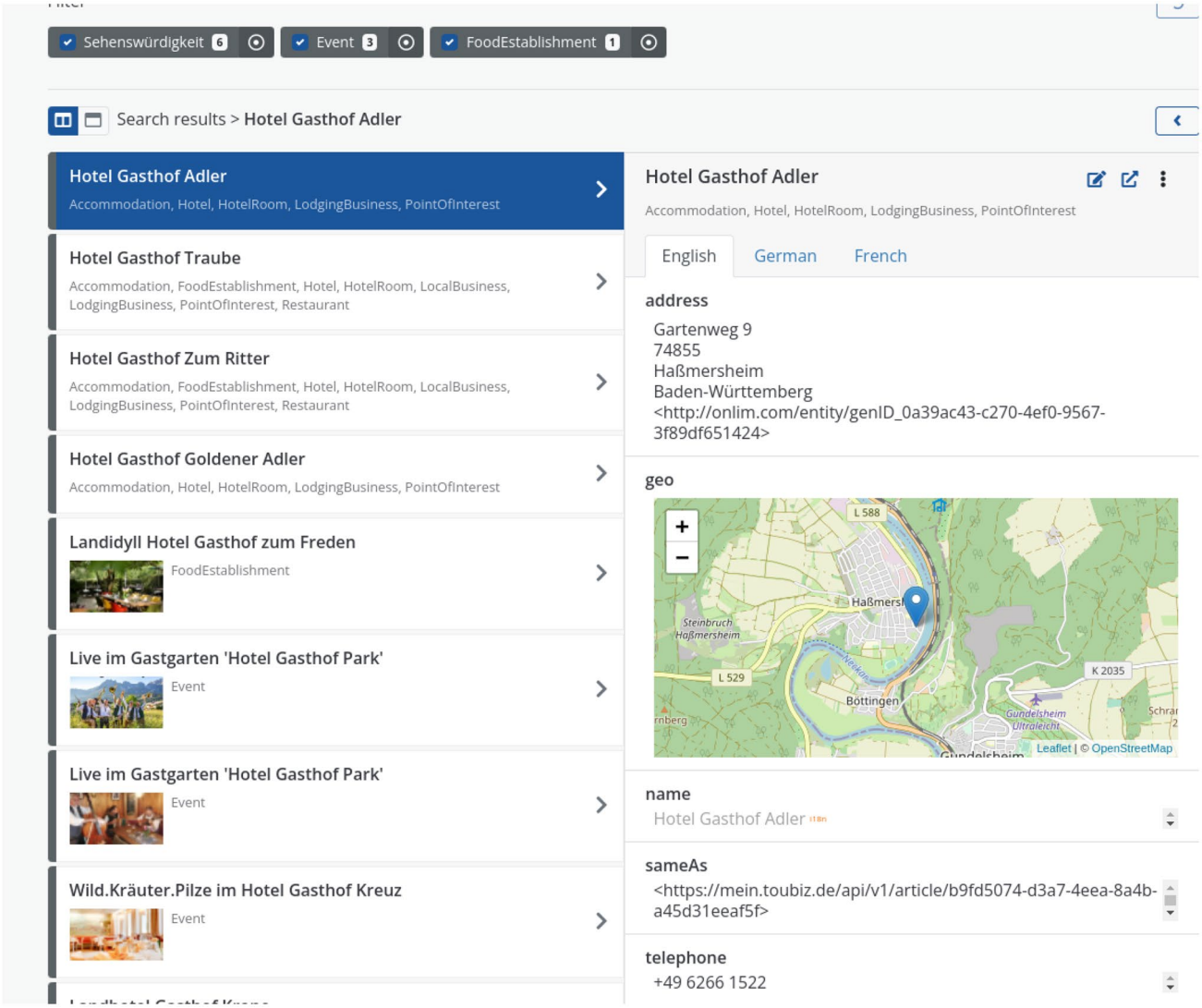

Fig. 6 A screenshot from the pre-production deployment of the knowledge graph

knowledge graphs as it also consists of processes like creation, assessment, cleaning, enrichment, and deployment. The major difference however is that traditional knowledge engineering focuses on large and complex TBoxes while building knowledge graphs would need efficient and effective methods for ABox heavy knowledge engineering due to the reasons discussed in the paper.

The individual aspects of building knowledge graphs and more traditionally linked open datasets have been the focus of both researchers and practitioners for many years. We already covered the state of the art on each aspect to some extent in the Sections "Knowledge Creation", "Knowledge Hosting", "Knowledge Curation" and "Knowledge Deployment". There are also various recent surveys studying different aspects of knowledge graph building. For example, the survey from Paulheim [44] covers various approaches for knowledge cleaning and enrichment. Another survey covers entity alignment approaches for knowledge enrichment [45]. A quick look at the literature shows that a significant amount of effort is put into the creative aspect of building knowledge graphs, which is also evident from the plethora of declarative mapping languages and tools, as well as the activities of the W3C Knowledge Graph Construction Community Group.[14] Although the individual aspects are important, knowledge graphs have a lifecycle that goes beyond one-shot knowledge creation activities. As also pointed out in a recent survey [46], the maintanance step is as crucial as the creation step. In this paper, we present a holistic approach

[14] https://github.com/kg-construct/.

to building knowledge graphs that cover the process from creation to deployment.

A relevant methodology was proposed by Sequeda et al. [47]. The methodology follows the principle of applying small iterations of simple TBox modeling and a population of relatively larger ABoxes based on this TBox. The produced knowledge graph is then evaluated and iteratively expanded until the requirements of the use case are satisfied. The methodology is however mainly focused on knowledge creation and also comparable with our knowledge creation approach and presented bottom-up and top-down models. However, we see the knowledge graph building process as more than just creating as its quality and maintenance are also fundamental for applications to work properly with it.

Conclusion and Future Work

Acquisition and representation of domain and task knowledge have been a crucial goal for AI, especially since Feigenbaum's knowledge principle. This goal led to various works such as knowledge-based systems. The increasing complexity of knowledge-based systems led to the development of knowledge engineering methodologies for the systematic development and maintenance of such systems. Knowledge graphs are the latest answer to this search. Although on the surface knowledge graphs may look like "yet another knowledge base in graph form", there are fundamental differences. Unlike traditional knowledge bases, knowledge graphs are ABox heavy, meaning that knowledge is more explicitly represented than hidden in complex TBoxes. There are also inherently larger and their flexible data model allows rather seamless integration of heterogeneous data sources.

The fundamental differences in the size and heterogeneity of knowledge graphs call for a different knowledge engineering paradigm that is more focused on the creation, hosting, and curation of ABox than TBox. We presented different aspects of building a knowledge graph and explained how it is being implemented in a significant project in the tourism domain.

In future work, we will continue with the refinement of the aspects such as assessment to make them more suitable for knowledge graphs and work on the missing parts of our approach such as semi-automated knowledge correction and link detection beyond identity links.

Funding Open access funding provided by University of Innsbruck and Medical University of Innsbruck.

Declarations

Conflict of interest Author U.S. declares that he has no conflict of interest. Author E.K. declares that he has no conflict of interest. Author K.A. declares that he has no conflict of interest. Author E.H. declares that he has no conflict of interest. Author J.O. declares that she has no conflict of interest. Author D.S. declares that he has no conflict of interest. Author J.U. declares that he has no conflict of interest. Author D.F. declares that he has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Newell A, Shaw JC, Simon HA. Report on a general problem solving program. In: IFIP Congress, 1959; vol. 256, p. 64. Pittsburgh, PA.
2. Feigenbaum EA. How the "what" becomes the "how". Commun ACM. 1996;39(5):97–104.
3. Fensel D, Simsek U, Angele K, Huaman E, Kärle E, Panasiuk O, Toma I, Umbrich J, Wahler A. Knowledge graphs—methodology, tools and selected use cases. Cham, Switzerland: Springer; 2020.
4. Şimşek U, Angele K, Kärle E, Opdenplatz J, Sommer D, Umbrich J, Fensel D. Knowledge graph lifecycle: Building and maintaining knowledge graphs. In: Proceedings of the 2nd International Workshop on Knowledge Graph Construction Co-located with 18th Extended Semantic Web Conference (ESWC 2021), 2021; vol. 2873. CEUR Workshop Proceedings. http://ceur-ws.org/Vol-2873/paper12.pdf. Accessed 30 Mar 2022.
5. Şimşek U, Angele K, Kärle E, Panasiuk O, Fensel D. Domain-specific customization of schema.org based on SHACL. In: The Proceedings of the 19th International Semantic Web Conference. LNCS, vol 12507. Springer, Athens, Greece, pp 585–600 (2020)
6. Mausam M. Open information extraction systems and downstream applications. In: Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence, 2016; p. 4074–77. https://www.ijcai.org/proceedings/2016
7. Mitchell T, Cohen W, Hruschka E, Talukdar P, Yang B, Betteridge J, Carlson A, Dalvi B, Gardner M, Kisiel B, et al. Never-ending learning. Commun ACM. 2018;61(5):103–15.
8. Dimou A, Vander Sande M, Colpaert P, Verborgh R, Mannens E, Van de Walle R. RML: a generic language for integrated RDF mappings of heterogeneous data. In: Proceedings of the Workshop on linked data on the web (LDOW2014) co-located with the 23rd International World Wide Web Conference (WWW2014), April 8. CEUR Workshop Proceedings, 2014; Vol-1184, Seoul, South Korea. http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf. Accessed 30 Mar 2022.
9. Şimşek U, Umbrich J, Fensel D. Towards a knowledge graph lifecycle: a pipeline for the population of a commercial knowledge graph. In: Proceedings of Conference on Digital Curation Technologies (Qurator 2020). CEUR-WS, Berlin, Germany 2020.

http://ceur-ws.org/Vol-2535/paper_10.pdf. Accessed 30 Mar 2022.
10. Delva T, Van Assche D, Heyvaert P, De Meester B, Dimou A. Integrating nested data into knowledge graphs with RML fields. In: KGWC2021, the Knowledge Graph Construction, 2021; vol. 2873, pp. 1–16.
11. Carroll JJ, Bizer C, Hayes P, Stickler P. Named graphs. Web Semant. 2005;3(4):247–67. https://doi.org/10.1016/j.websem.2005.09.001.
12. Hartig O, Champin P-A. Metadata for rdf statements: the rdf-star approach. In: Lotico Talk 2021. https://w3c.github.io/rdf-star/presentations/RDF-star_Lotico.pdf
13. Bizer C. Quality-driven information filtering in the context of web-based information systems. PhD thesis, Free University of Berlin 2007.
14. Färber M, Bartscherer F, Menne C, Rettinger A. Linked data quality of dbpedia, freebase, opencyc, wikidata, and YAGO. Semant Web. 2018;9(1):77–129. https://doi.org/10.3233/SW-170275.
15. Zaveri A, Rula A, Maurino A, Pietrobon R, Lehmann J, Auer S. Quality assessment for linked data: A survey. Semant Web. 2016;7(1):63–93. https://doi.org/10.3233/SW-150175.
16. Mendes PN, Mühleisen H, Bizer C. Sieve: linked data quality assessment and fusion. In: Srivastava D, Ari I. editors. Proceedings of the 2012 Joint EDBT/ICDT Workshops, Berlin, Germany, March 30, 2012; pp. 116–123. ACM, 2012. https://doi.org/10.1145/2320765.2320803.
17. Kontokostas D, Zaveri A, Auer S, Lehmann J. Triplecheckmate: a tool for crowdsourcing the quality assessment of linked data. In: Klinov P, Mouromtsev D. editors. Knowledge Engineering and the Semantic Web—4th International Conference, KESW 2013, St. Petersburg, Russia, October 7–9, 2013. Proceedings. Communications in Computer and Information Science, 2013; vol. 394, pp. 265–272. Springer. https://doi.org/10.1007/978-3-642-41360-5_22.
18. Kontokostas D, Westphal P, Auer S, Hellmann S, Lehmann J, Cornelissen R, Zaveri A. Test-driven evaluation of linked data quality. In: Chung C, Broder AZ, Shim K, Suel T. editors. 23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7–11, 2014; pp. 747–758. ACM, 2014. https://doi.org/10.1145/2566486.2568002.
19. Debattista J, Auer S, Lange C. Luzzu—a methodology and framework for linked data quality assessment. ACM J Data Inf Qual. 2016;8(1):4–1432. https://doi.org/10.1145/2992786.
20. Dimou A, Kontokostas D, Freudenberg M, Verborgh R, Lehmann J, Mannens E, Hellmann S, Van de Walle R. Test-driven assessment of [R2]RML mappings to improve dataset quality. In: Proceedings of the 14th International Semantic Web Conference: Posters and Demos. CEUR Workshop Proceedings, 2015; vol. 1486. http://ceur-ws.org/Vol-1486/paper_108.pdf. Accessed 30 Mar 2022.
21. Randles A, O'Sullivan D. Evaluating Quality Improvement techniques within the Linked Data Generation Process. In: Proceedings of the 18th International Conference on Semantic Systems. Vienna, Austria, 2022. CEUR-WS proceedings, Vol 1162.
22. Paulheim H, Bizer C. Type inference on noisy rdf data. In: International Semantic Web Conference, LNCS. 2013; vol. 8218, pp. 510–525. Springer.
23. Paulheim H. Identifying wrong links between datasets by multi-dimensional outlier detection. In: WoDOOM, CEUR-WS proceedings, 2014; Vol 1162, pp. 27–38.
24. Papaleo L, Pernelle N, Saïs F, Dumont C. Logical detection of invalid Sameas statements in rdf data. In: International Conference on knowledge engineering and knowledge management, LNAI, 2014; vol. 8876, pp. 373–84. Springer.
25. Beek W, Rietveld L, Bazoobandi H.R, Wielemaker J, Schlobach S. Lod laundromat: a uniform way of publishing other people's dirty data. In: International Semantic Web Conference, LNCS, 2014; vol. 8796, pp. 213–28. Springer.
26. Rekatsinas T, Chu X, Ilyas IF, Ré C. Holoclean: Holistic data repairs with probabilistic inference. 2017. arXiv preprint arXiv:1702.00820.
27. Chu X, Morcos J, Ilyas IF, Ouzzani M, Papotti P, Tang N, Ye Y. Katara: reliable data cleaning with knowledge bases and crowdsourcing. Proc VLDB Endow. 2015;8(12):1952–5.
28. De Meester B, Heyvaert P, Arndt D, Dimou A, Verborgh R. Rdf graph validation using rule-based reasoning. Semant Web. 2021;12(1):117–42.
29. Ge C, Gao Y, Weng H, Zhang C, Miao X, Zheng B. Kgclean: an embedding powered knowledge graph cleaning framework. 2020. arXiv preprint arXiv:2004.14478.
30. Fensel D, Şimşek U, Angele K, Huaman E, Kärle E, Panasiuk O, Omar H. Verigraph: a verification framework for knowledge integrity. Report, MindLab; 2020.
31. Bleiholder J, Naumann F. Data fusion. ACM Comput Surv. 2009. https://doi.org/10.1145/1456650.1456651.
32. Garshol LM, Borge A. Hafslund Sesam—an archive on semantics. In: Proceedings of the 10th Extending Semantic Web Conference (ESWC2013): semantics and big data, Montpellier, France, May 26–30, 2013. Lecture Notes in Computer Science, 2013; vol. 7882, pp. 578–92. Springer. https://doi.org/10.1007/978-3-642-38288-8_39.
33. Volz J, Bizer C, Gaedke M, Kobilarov G. Silk-a link discovery framework for the web of data. In: Proceedings of theWWW2009 Workshop on linked data on the Web, LDOW 2009, Madrid, Spain, 2009. CEUR WorkshopProceedings vol. 538, CEUR-WS.org 2009.
34. Ngomo AN, Auer S. LIMES—a time-efficient approach for large-scale link discovery on the web of data. In: Proceedings of the 22nd international joint conference on artificial intelligence (IJCAI2011), Barcelona, Spain, July 16–22, 2011; pp. 2312–317. AAAI Press, 2011. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-385.
35. Obraczka D, Schuchart J, Rahm E. Embedding-assisted entity resolution for knowledge graphs. In: Proceedingsof the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic WebConference (ESWC 2021), Online, June 6, 2021. CEUR Workshop Proceedings 2873, CEUR-WS.org 2021
36. Lu G, Zhang L, Jin M, Li P, Huang X. Entity alignment via knowledge embedding and type matching constraints for knowledge graph inference. J Ambient Intell Humaniz Comput, 2021; pp. 1–11. https://doi.org/10.1007/s12652-020-02821-2.
37. Rossi A, Barbosa D, Firmani D, Matinata A, Merialdo P. Knowledge graph embedding for link prediction: a comparative analysis. ACM Trans Knowl Discov Data (TKDD). 2021;15(2):1–49.
38. Opdenplatz J, Şimşek U, Fensel D. Duplicate detection as a service. 2022. https://doi.org/10.48550/ARXIV.2207.09672, arXiv:arxiv.org/2207.09672
39. Azzam A, Aebeloe C, Montoya G, Keles I, Polleres A, Hose K. Wisekg: balanced access to web knowledge graphs. In: Proceedings of the Web Conference 2021, 2021; pp. 1422–34. https://doi.org/10.1145/3442381
40. Zouaghi I, Mesmoudi A, Galicia J, Bellatreche L, Aguili T. Query optimization for large scale clustered rdf data. In: DOLAP, CEUR-WS Proceedings, 2020; vol 2572, pp. 56–65.
41. Troullinou G, Kondylakis H, Lissandrini M, Mottin D. Sofos: demonstrating the challenges of materialized view selection on knowledge graphs. In: Proceedings of the 2021 International Conference on management of data, 2021; pp. 2789–93. https://doi.org/10.1145/3448016.
42. Angele K, Meitinger M, Bußjäger M, Föhl S, Fensel A. Graphsparql: A graphql interface for linked data. In: Proceedings of the

37th ACM/SIGAPP Symposium on applied computing. SAC '22, pp. 778–85. Association for Computing Machinery, New York, NY, USA, 2022. https://doi.org/10.1145/3477314.3507655.
43. Kotis KI, Vouros GA, Spiliotopoulos D. Ontology engineering methodologies for the evolution of living and reused ontologies: status, trends, findings and recommendations. Knowl Eng Rev. 2020;35:4. https://doi.org/10.1017/S0269888920000065.
44. Paulheim H. Knowledge graph refinement: a survey of approaches and evaluation methods. Semant web. 2017;8(3):489–508.
45. Zeng K, Li C, Hou L, Li J, Feng L. A comprehensive survey of entity alignment for knowledge graphs. AI Open. 2021;2:1–13.
46. Tamašauskaitundefined G, Groth P. Defining a knowledge graph development process through a systematic review. ACM Trans Softw Eng Methodol. 2022. https://doi.org/10.1145/3522586 (**Just Accepted**).
47. Sequeda JF, Briggs WJ, Miranker DP, Heideman WP. A pay-as-you-go methodology to design and build enterprise knowledge graphs from relational databases. In: The Semantic WebISWC 2019. LNCS, vol. 11779. Springer, 2019. https://doi.org/10.1007/978-3-030-30796-7_32. Collection-title: Lecture Notes in Computer Science.