Knowledge Graph Lifecycle: Building and Maintaining Knowledge Graphs

Umutcan Şimşek[1], Kevin Angele[1,2] Elias Kärle[1,2], Juliette Opdenplatz[1], Dennis Sommer[1], Jürgen Umbrich[2], and Dieter Fensel[1]

[1] University of Innsbruck, Technikerstrasse 21a 6020 Innsbruck, Austria
{umutcan.simsek, kevin.angele, juliette.opdenplatz, dennis.sommer, dieter.fensel}@sti2.at
[2] Onlim GmbH elias.kaerle, juergen.umbrich@onlim.com

Abstract. Knowledge graphs are only useful if they satisfy the requirements of those applications in terms of quality. In this in-use experience paper, we present our approach and tools for supporting the knowledge graph Lifecycle that starts with creation and hosting and continues with the curation and deployment. The curation process enables the maintenance of a knowledge graph, especially in terms of correctness and completeness. We provide process models and evaluation of developed tools with Knowledge Graphs in the tourism domain. We discuss the lessons learned from implementing such an approach in an open and commercial setting in several use cases.

Keywords: knowledge graphs · knowledge graph lifecycle · knowledge curation · knowledge creation

1 Introduction

The lifecycle of a knowledge graph comes with two main challenges (1) how to integrate heterogeneous sources in a knowledge graph in a scalable manner (2) how to make them a high-quality resource (e.g., semantically and syntactically correct, no duplicate instances) given the applications in hand.

In this in-use experience paper, we present various tasks of the knowledge graph lifecycle and the tools we developed or adopted to support the knowledge graph lifecycle. The Knowledge Graphs built with this approach are deployed in an open as well as a commercial setting in the tourism domain to support conversational agents. We learned various lessons while implementing our approach about;

- orchestration of different tasks in the knowledge graph lifecycle
- technical and conceptual challenges of dealing with heterogeneous data and distributed actors for knowledge creation

– different perspectives on quality and knowledge integrity for error detection
– challenges of duplication detection configuration during knowledge enrichment

In the remainder of the paper, we first present the tasks of the knowledge graph lifecycle (Section 2) and tools developed or adopted to tackle them, including the use cases for which the knowledge graphs are deployed. Then, we present the lessons learned from developing and implementing our approach (Section 3). Finally, we provide concluding remarks and indicators for future work (Section 4). Note that we do not have a dedicated related work section. Naturally, we benefited from a plethora of research work while developing our approach. Due to space restrictions, we do not have a dedicated related work section; however, we discuss the ones that are directly related to the lifecycle and individual processes. A comprehensive review can be found in [5].

2 Knowledge Graph Lifecycle

There are already proposed methodologies for iterative construction of Knowledge Graphs from various sources (a recent one is described in [11]), but construction is only one side of the coin. On the one hand, it must be built from various heterogeneous sources, on the other hand, it must be turned into a high-quality resource that satisfies the requirements of the use case and applications in hand [14, 2]. Figure 1 shows these processes and the tools developed or adopted to support them.

The lifecycle starts with the creation process that deals with the generation of semantically annotated data from heterogeneous sources. For this task, we developed the Importer Tool, which is an ETL tool utilizing RML mappings and a mapping engine we developed called RocketRML [16][3], as part of the semantify.it platform[7][4]. The created knowledge is then hosted in an RDF triplestore like GraphDB[5].

The curation process aims to improve the correctness and completeness of a knowledge graph. The first step is to assess the quality of the knowledge graph. Knowledge quality assessment may involve various dimensions including correctness and completeness[6].

Based on the quality score of the correctness and completeness dimensions, the cleaning and enrichment processes can be triggered. The cleaning task involves error detection and correction. We developed the VeriGraph tool that uses integrity constraints implemented with SHACL to detect errors in a knowledge graph. The enrichment process involves detecting duplicate instances in a knowledge graph (or across knowledge graphs) and enrich them with "same as" links.

[3] https://github.com/semantifyit/RocketRML
[4] https://semantify.it - registration and login required.
[5] https://graphdb.ontotext.com
[6] The knowledge assessment tool is still at an early stage of development and not presented in this paper. An early demo can be seen at https://qat.semantify.it

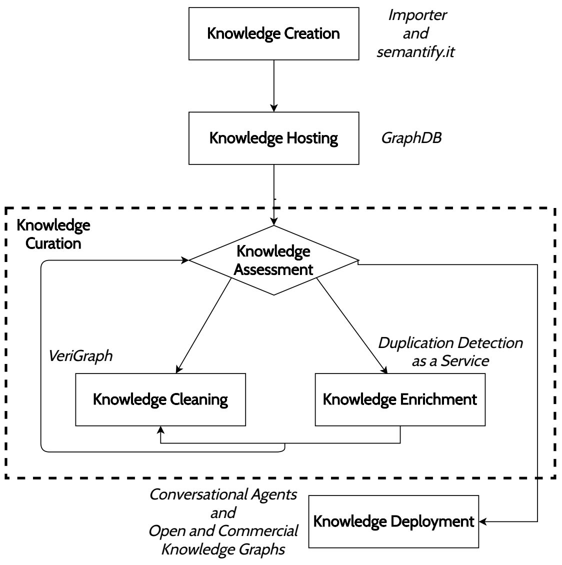

Fig. 1: The knowledge graph lifecycle (adapted from [5]). The italic labels represent the tools developed/used for each process

The property values of the linked instances may need to be fused afterward, which may require another cleaning process to identify violations of integrity constraints (e.g., a property with one maximum cardinality may have multiple values after the fusion.). The curated knowledge graphs are finally deployed to be consumed by various applications. These processes continue iteratively to constitute the lifecycle. In the remainder of the section, we focus on the creation, cleaning, and enrichment tasks and present the tools we employed to tackle these tasks including some design decisions. Finally, knowledge deployment will be presented in the form of use cases.

2.1 Knowledge Creation

Knowledge creation, as defined in [5], describes "extracting information from different sources, structuring it, and creating useful knowledge". For the creation process, we use schema.org vocabulary as schema, because it covers many domains and is a de facto industrial standard for semantic annotations on the web.

In principle knowledge, creation can be done manually, for example via a GUI, through mappings from (semi-)structured sources or semi-automatically from unstructured sources. In our use cases, the majority of the knowledge is created via declarative mappings from hierarchical data sources. In the context of our work, the data to be integrated into the knowledge graph was obtained from different service providers in different formats, typically JSON or XML. The mapping was then defined from those sources to schema.org.

The process model for knowledge creation via mappings is the following: We first collect raw data from different service providers via web services. Each object retrieved is mapped to schema.org to create an instance of a schema.org type with its property values assertions. Alternatively, data providers can provide RDF data directly. The generated or acquired RDF data is then enriched with provenance information based on PROV-O[7].

Besides the technical challenge of creating mapping files, there is also a conceptual one that involves the selection of types and properties for the mapping. It is even more challenging when the actors creating the mappings and selecting the types and properties are not the same. We developed an approach for creating domain-specific patterns of schema.org and its extensions[15]. These patterns describe the relevant types, properties, and constraints for a domain and facilitate the communication between domain experts, knowledge engineers, and mapping rule creators. The patterns are implemented via SHACL shapes to make them more machine-processable and can be used to verify created knowledge (see also Section 2.2). For the declarative mapping rules, we adopted RDF Mapping Language (RML) [4] with YARRML[8] syntax mostly to enable the software developers to create mappings easily, as they are familiar with YAML-based syntaxes.

We implemented the creation process in the Importer tool. The tool allows registration of new sources including all the access information and their RML mapping files. The timing and frequency of the mappings can be specified with cron strings. The Importer has Apache NiFi[9] in its core to manage the entire data flow from accessing raw data to storing it in a triplestore. Apache NiFi is a dataflow management tool that offers load balancing, buffering and guaranteed delivery. The actual mapping is executed via an external RocketRML instance, a scalable RML mapper implemented with NodeJS.

RocketRML currently supports JSON, XML, and CSV formats and adopts optimization techniques like JOIN path memoization for high-performance. It supports various JSON-Path and XPath implementations allowing users to access extra features like backward traversal in a JSON file with JSON-Path Plus[10]. It also supports function mappings which are frequently used for transforming property values and distinguishing between different subtypes of schema.org type during the mapping (e.g., different types of events can be dynamically mapped with a single mapping). Listing 1 shows an excerpt from a mapping supported by RocketRML[11]. The mapping file creates schema:LocalBusiness instances from various touristic regions' data in Tyrol. Note that the getType function returns the suitable subtype of schema:LocalBusiness (e.g., schema:Store, schema:Library) for a given instance. The relationship between a local business and its opening hours is given by the nested structure of XML tags and not

[7] https://www.w3.org/TR/prov-o
[8] https://rml.io/yarrrml/
[9] https://nifi.apache.org
[10] https://www.npmjs.com/package/jsonpath-plus
[11] Full mapping can be found online: https://tinyurl.com/96xcfs3k

```
#prefixes
sources:
...
mappings:
  acc:
    sources:
      - acc
    s: ml:$(@Id)
    po:
      - [a, {function: myfunc:getType, parameters:
      ↪  ["$(Details/Topics/Topic/@Id)"]}]
      - [schema:name, "$(Details/Names/Translation[@Language='de'])",
      ↪  de~lang]
      - [schema:name, "$(Details/Names/Translation[@Language='en'])",
      ↪  en~lang]
      ...
      - [schema:openingHoursSpecification, {mapping: hours, join: [@Id,
      ↪  ../../../../@Id]}]
   ...
```

Listing 1: An excerpt from an RML mapping with YARRML syntax for local businesses in touristic regions

by primary-foreign key relationships. Therefore, the mapping uses a backward traversal from opening hours object to the local businesses' Id field to join them properly.

The RML mapper used in the Importer, RocketRML, can map 25K triples per second on average. However, the overhead caused by sending queries to the GraphDB instance over HTTP harms the overall import process. A more detailed explanation of knowledge creation via mappings and a detailed evaluation of the importer tool can be found in [17].

2.2 Knowledge Curation

Knowledge Curation is a process for assessing and improving a knowledge graph in various dimensions, especially correctness and completeness (see also "knowledge refinement" [10], with a narrower set of tasks). In this section, we explain the processes comprising Knowledge Curation and the tasks on which we focused in the scope of our work.

Knowledge Cleaning Knowledge Cleaning is a process that aims to improve the correctness of a knowledge graph. It consists of (a) error detection, the task for identifying the erroneous type and property value assertions, and (b) error correction, fixing the identified statements. We focused on the former, particularly the verification task where the knowledge graph is checked against a specification such as integrity constraints.

Our approach is based on verifying the instances in a knowledge graph against the domain-specific patterns of schema.org. These patterns are expressed with a subset of SHACL [12]. For such an approach we considered various SHACL verifiers but none of them covered our needs properly. RDFUnit was the closest tool to satisfy our requirements as it is triplestore-independent and can work directly on SPARQL endpoints without loading a data dump to the memory. However, we had issues with large Knowledge Graphs in some of our use cases (particularly Tyrolean Tourism Knowledge Graph) as the verification never ended after 10M triples (see below for evaluation).

For detecting errors in a knowledge graph, we conceptualized and developed a verifier that checks whether a particular subset of a knowledge graph fits the domain-specific pattern. Figure 2 shows the process model.

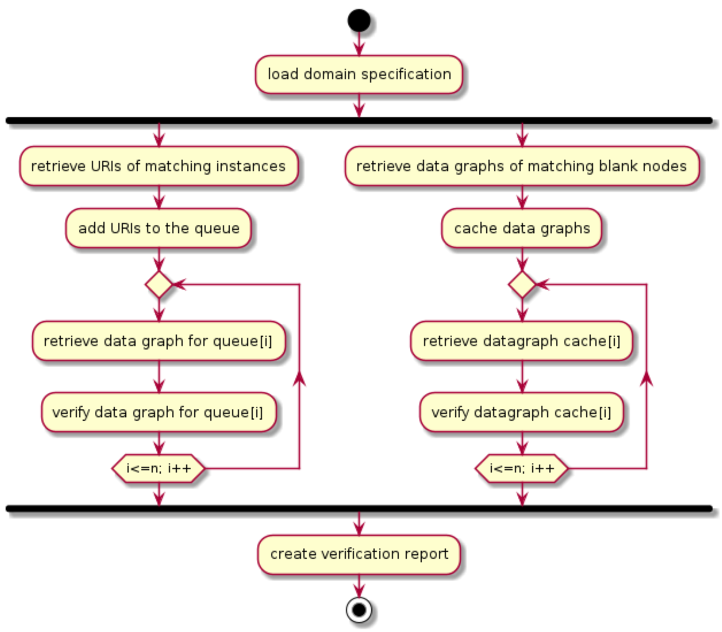
load domain specification
retrieve URIs of matching instances
add URIs to the queue
retrieve data graph for queue[i]
verify data graph for queue[i]
i<=n; i++
retrieve data graphs of matching blank nodes
cache data graphs
retrieve datagraph cache[i]
verify datagraph cache[i]
i<=n; i++
create verification report

Fig. 2: UML Activity Diagram for error detection process model

The first step loads a domain-specific pattern that comprises the shapes graph for verification. Then the verification process is split into two lines: The first line of verification retrieves the URIs of the instances that match the target specification and adds them to a verification queue. Then, for each URI in the

[12] Details of the subset can be found in [15]

queue, a data graph is retrieved and verified against the domain-specific pattern. In parallel, the verification process retrieves all blank nodes that match the target description and their data graphs and stores them in a cache. Then, each data graph in the cache is verified against the loaded domain-specific pattern. The results of both lines of verification are then compiled in a verification report. Note that the data graph in both lines corresponds to the subgraph built by following all the outgoing edges of a focus node recursively until no new node can be added to the data graph (e.g., all nodes to be expanded are literals). Due to the limitations of SPARQL, such a recursive traversal is tricky. There is a way to do this in a single query by using `?s (:|!:)* ?o` graph pattern, but admittedly it is a bit *hacky*. One can also rely on DESCRIBE queries if the triplestore implements them appropriately.

We implemented our error detection approach in the VeriGraph tool[13]. The tool has been implemented in Javascript and available with an open license. It can be configured to run on any knowledge graph that provides a SPARQL endpoint. In our experience with many SHACL verifiers, we realized there are generally two main issues in practice: (1) operating in-memory, which causes insufficient memory problems with large data graphs (2) SPARQL endpoints are not always reliable for frequent queries that return high-volume results. The first issue we address with a caching mechanism. The data graphs are cached on the disk and only loaded to the memory when they are needed for verification. The second issue is addressed by both indexing and the caching mechanism. Indexing the URIs and querying their data graphs one-by-one reduces the size of the data graph returned by a single query. Each constraint component defined by the property shapes is checked by graph-traversal in the memory. This reduces the number of SPARQL queries running against an endpoint for verification. Additional to the typical SHACL verification report, the VeriGraph tool provides metadata about the verification process (e.g., duration, number of violations found).

We evaluated the VeriGraph on several subgraphs of Tyrolean Tourism Knowledge Graph with an increasing number of triples, starting from 100K up to 1B[14]. Each knowledge graph contains instances of types like Event, Hotel, HotelRoom, Person, and Product. The evaluation has been conducted on a server with an Intel Core i9-9900K Octa-Core 3.60GHz processor, 64GB RAM, and 2TB SSD. We compared our implementation with the SHACL verifier of AllegroGraph (via agTool)[15], RDFUnit, built-in SHACL verifier of Stardog, and TopBraid SHACL API[16]. The instances were verified against a set of constraints with different target specifications[17], except for one constraint for Stardog, due to a non-supported constraint type (a property-pair constraint). Figure 3 shows the verification time in relation to knowledge graph size. While each tool detects about the same num-

[13] https://github.com/semantifyit/VeriGraph
[14] http://dataset.sti2.at/datasets/
[15] https://franz.com/agraph/support/documentation/6.6.0/shacl.html
[16] https://github.com/TopQuadrant/shacl
[17] https://github.com/semantifyit/VeriGraph/blob/master/constraints/constraints.ttl

ber of violations for the same size, in terms of time spent, VeriGraph stands out as the size grows. It is the only one that can finish verification on a knowledge graph with 1B triples. In smaller Knowledge Graphs, VeriGraph is behind the tools either working completely in-memory (TopBraid) or on their triplestores natively (AllegroGraph, Stardog) for smaller datasets. For RDFUnit and VeriGraph connecting to generic SPARQL endpoints of triplestores create an overhead. Our initial investigations showed that RDFUnit's performance is affected primarily not by the size but the number of violations found. This could be because of generating one SPARQL query for each constraint component and overhead caused by processing the results of these queries to create verification reports. Nevertheless, VeriGraph is not a complete SHACL verifier as it only recognizes a subset of SHACL (e.g. only class-targets are allowed). However, it appears to be a feasible choice for our use cases.

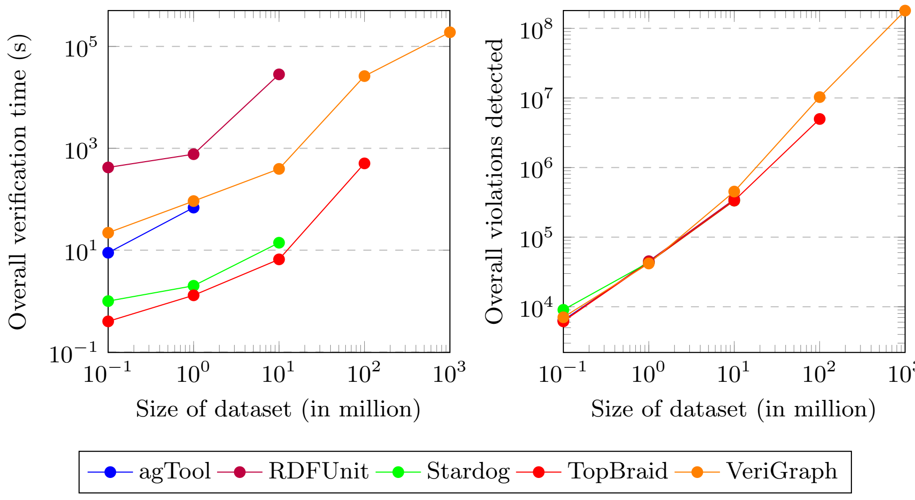

Fig. 3: Comparison of different tools in terms of total verification time and number of violations detected for different dataset sizes [1].

Knowledge Enrichment Knowledge Enrichment is a process that aims to improve the completeness of a knowledge graph [5]. The completeness is enhanced by identifying and adding missing instance, property value, and equality assertions. In our work, we focused on the duplicate detection task, to find the equality assertions between instances within or across Knowledge Graphs and add the missing instance equality assertions. We take schema.org as the golden standard and do not focus on the alignment of TBox. Heterogenous schemas from different Knowledge Graphs are mapped to schema.org via declarative mappings.

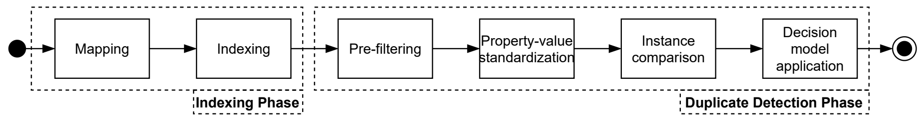
Mapping
Indexing
Pre-filtering
Property-value standardization
Instance comparison
Decision model application
Indexing Phase
Duplicate Detection Phase

Fig. 4: A short depiction of the duplicate detection process.

We developed a highly configurable service-oriented approach to the duplicate detection problem that allows linking duplicate instances in a knowledge graph or from external Knowledge Graphs [9]. The process model is shown in Figure 4. The whole process is divided into two main phases: (1) the indexing phase where the knowledge sources are mapped to a common format if needed and indexed into an internal Elasticsearch[18] instance, and (2) the duplicate detection phase which is divided into four steps: The first step is the pre-filtering step that determines candidate duplicates of two previously indexed knowledge sources. This step selects a set of candidate instances to which the duplicate detection process will pay attention[19]. In the second step properties are normalized such that two instances are easier to compare (e.g., via regular expressions over string values, mathematical operations to normalize units of certain numerical values). The third step executes the actual detailed comparison between the candidate duplicates which results in a similarity score for a candidate duplicate. Here several different similarity metrics are used for different types of property values (e.g., Jaccard, Levenshtein for string similarity; Euclidean distance for geocoordinates). Finally, the fourth step applies a decision model in which the similarities from the third step are utilized to classify the suspected duplicates as either duplicates or non-duplicates. The output of the duplicate detection phase can then simply be translated into *schema:sameAs* statements.

The duplication detection problem is almost as old as computer science itself and there have been a plethora of approaches to tackle it. We also examined various tools such as Duke[6], LIMES[8], Silk[13] for linking instances in knowledge graphs. They all have different advantages and disadvantages; however, one common aspect is that they all need some form of a configuration file for properties used for similarity measurement, their weights, and thresholds for determining duplicate instances. The configuration effort can become quite high especially when the schemas get complicated. Many tools offer machine learning algorithms to find the best values for different configuration parameters to achieve the best F-score. We combined the insights we obtained from these tools and provided a supervised configuration learning approach with larger flexibility and granularity that allows learning parameters not only for choosing similarity metrics and

[18] https://www.elastic.co/elasticsearch/

[19] The pre-filtering works based on the more_like_this queries of elasticsearch. The ”likeness” is calculated with TF-IDF. Putting a high threshold for the number of matching terms may harm the recall of the overall approach. See https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-mlt-query.html .

thresholds but also indexing and pre-filtering steps. We provide the possibility to use various algorithms such as time-constrained brute force, hill climbing, localized brute force, and genetic algorithms with random mutations for each parameter. The configuration learning can be adjusted towards optimizing for not only F-score but also recall or precision.

We implemented our approach in a tool called Duplicate Detection as a Service (DDaaS). The tool consists of multiple services for different tasks. These services are orchestrated via REST. The decision in the favor of a service-oriented architecture is to facilitate easy replacement of individual components and their independent development.

To evaluate the approach, we compared it with three other tools, Duke, LIMES and Silk, all of which also influenced the development of DDaaS. We compared the tools over two datasets (Restaurants[20] and SPIMBENCH[21]). The results are displayed in Table 1.

Restaurants			
Tool	F1-Score	Precision	Recall
DDaaS	0.76	1.00	0.61
Duke	0.77	1.00	0.62
LIMES	0.80	0.86	0.74
Silk	0.40	0.79	0.27

SPIMBENCH			
Tool	F1-Score	Precision	Recall
DDaaS	0.85	0.98	0.76
Duke	0.09	0.05	0.75
LIMES	0.72	0.88	0.61
Silk	0.62	0.77	0.53

Table 1: Duplicate detection comparison of DDaaS, Duke, LIMES, and Silk

We ran every tool the most automated way possible and results indicate that DDaaS is at least on par with the other tools. The SPIMBENCH results are particularly interesting here, while the results for the Restaurants dataset are more balanced. The main differences between these datasets lie in the number of properties and their completeness. While the Restaurant dataset is a perfect *toy* dataset, the instances in the SPIMBENCH dataset are very sparse with regards to property values on many instances. Duke's purely genetic approach to learning a configuration for this particular dataset seems to be flawed as it will keep retrying to use properties that are not even available for most instances. DDaaS' approach to learning a configuration is a composition of different approaches which includes a genetic approach but also less randomized approaches. We call this composition the learning strategy. Since every single element of this strategy (i.e., configurable pieces) can be configured to aim to optimize one of the three measures (precision/recall/F1), it achieved a good performance even on such an incomplete dataset. Further evaluation on Tyrolean Tourism Knowledge Graph is ongoing.

[20] https://www.cs.utexas.edu/users/ml/riddle/
[21] https://project-hobbit.eu/challenges/om2020/

2.3 Knowledge Deployment

"The proof of the pudding is in the eating." A knowledge graph is only as valuable as the applications it enables. Therefore, the knowledge deployment task deals with the applications that are powered by a knowledge graph. In this section, we present three use cases where this lifecycle-based approach is being implemented to build knowledge graphs to power various applications.

German Tourism Knowledge Graph The German Tourism Knowledge Graph is the reference project to implement the presented lifecycle-based approach. A "knowledge management tool" is currently being implemented to manage creation, hosting, curation, and deployment. The knowledge graph will integrate tourism-related data from the tourism marketing organizations of 16 federal states and many other external sources. The knowledge graph management tool will offer features like provenance tracking, machine-understandable licenses, and various types of visualizations as well as APIs for programmatically accessing to the knowledge graph. The knowledge graph will follow domain-specific patterns of schema.org and its extension developed by the Open Data Travel Alliance which is an organization that brings tourism experts from Austria, Germany, Italy, and Switzerland to create schemas for tourism knowledge graphs. These patterns are published in human-readable form and as SHACL shapes to help the data providers from the touristic marketing organizations to create RDF data via declarative mappings. Same patterns will be used to verify the incoming data.

The project started in December 2020 and will run for 2 years, with an initial prototype in May 2021[22]. It will be open to application developers from different domains (e.g., mobility service providers, online travel agencies) and aim to foster the development of intelligent applications that improve e-tourism processes.

Tyrolean Tourism Knowledge Graph A notable example is the Tyrolean Tourism Knowledge Graph[23], which contains more than 12B statements. It is populated with data from 11 different sources (mainly Destination Management Organizations from different regions in Tyrol) and updated daily. The presented knowledge creation process is applied fully in this use case and knowledge cleaning processes are running on demand. The knowledge coming from different sources is organized in named graphs. The named graphs imported from the same source on different time points are linked with each other via the provenance information, which allows applications like time series analysis on frequently changing data (e.g., accommodation prices, weather measurements).

Onlim Conversational Agents Onlim GmbH deploys knowledge graphs developed with the presented approach commercially. Onlim uses their knowledge

[22] See the project timeline online. - https://open-data-germany.org/projektstand/

[23] http://tirol.kg

graph to power their conversational agents in different domains such as tourism, education, energy, and finance. Most notably in the tourism domain, they provide about 20 conversational agents. These agents are typically goal-oriented dialog systems (GDS) that help users to achieve their goals via conversations. In the case of Onlim, a knowledge graph powers a GDS in two different ways:

– Providing entities for annotating user utterances to train Natural Language Understanding (NLU) models.
– Serving as a knowledge source to provide the knowledge needed for a task at hand

Onlim uses state-of-the-art GDS development frameworks such as DialogFlow[24] and RASA[25] to streamline the conversational aspects of a dialog system such as NLU, dialog management and NLG. Such frameworks work with intents, structures that represent the user goals a GDS supports. The frameworks use supervised machine learning to classify utterances to intents. They use Knowledge Graphs to create annotated utterances for each intent to help the machine learning models classify incoming utterances to the correct intents. An intent is then mapped to a SPARQL query and the user's question is answered based on the data provided by the knowledge graph (e.g., accommodation, events, infrastructure). The lifecycle explained throughout the paper ensures that the answers returned a high quality (e.g., resolved duplicate instances, correct property values). An example of such a GDS can be found online[26].

3 Discussion and Lessons Learned

In this section, we discuss our lessons learned from implementing the presented lifecyle, from the perspective of the overall approach, and creation and curation processes with references to the use cases above.

3.1 Lessons learned from the implementation of the lifecycle

Orchestration of different tasks in the lifecycle The knowledge graph lifecycle involves many tasks and each task has been addressed in the literature with a vast number of methods and tools. We realized however there is an important gap while implementing the lifecycle which is a tool that can provide periodic and on-demand actuation of various processes and their orchestration. An open and decoupled architecture can facilitate relatively painless integration of different steps in the lifecycle. The interoperability of different tools targeting different tasks remains an interesting research and engineering challenge. Here we can take a page from the book of the question-answering systems community as they have been proposing an open architecture to increase the reusability and interoperability of the tools targeting different steps of question-answering system development [12].

[24] https://dialogflow.com
[25] https://rasa.ai
[26] https://www.oberoesterreich.at/ - Flo-Bot virtual assistant.

Community effort needed to maintain existing research products There is a plethora of research that resulted in various tools for creation and curation processes. Unfortunately, many of them, especially relatively older ones were abandoned in their GitHub repositories, and not maintained further. Naturally, we gained valuable insights even by only studying the publications. However, it is hard to assess their suitability for different use cases without being able to run them properly and this may lead to reinventing the wheel. Community groups such as Knowledge Graph Construction Community Group (KGC CG) may be the solution to this "research prototype graveyard" situation. Such groups consisting of research and industrial partners can take a selection of approaches and tools and further maintain them as an open-source community effort, possibly under the umbrella of organizations like Apache Foundation.

3.2 Lessons learned from knowledge creation

Real data is not perfect, knowledge creation is not trivial Constructing Knowledge Graphs from heterogeneous sources scale well with declarative mappings. However, the data received from real-world IT solution providers are not always ideal. For instance, in the Tyrolean Tourism Knowledge Graph use case, we frequently encountered data sources that do not provide any fields to join two logical sources (e.g., events and their organizers) but the relationship is specified by nested structures. We worked around this by extending the existing JSONPath and XPath[27] implementations with a *~PATH* term which represents the absolute path of a value in the JSON or XML tree, which is suitable for joining nested structures. Moreover, various source-specific cleaning steps are involved in many cases, which hinders scalable development. Here again community efforts like KGC CG can be beneficial for identifying common challenges in declarative mappings and addressing them within the existing tools and approaches.

Conceptual and social challenges stand The advantages of using declarative mappings are clear in terms of flexibility and reusability, however, the conceptual and social challenges still stand. The domain experts must define the domain by identifying relevant types, properties, and constraints and communicate them to the developers and mapping creators. This is particularly challenging when these actors are distributed across different organizations, as it is in the German Tourism Knowledge Graph, where semantically annotated data comes from at least 16 different organizations to be integrated into a single knowledge graph. We experienced that simple human- and machine-understandable patterns of schema.org and its extensions published by domain experts improve the knowledge creation process significantly.

3.3 Lessons learned from knowledge curation

There can be different perspectives on knowledge integrity One experience we had with Tyrolean and German Tourism Knowledge Graph use cases is

[27] Newer XPath implementations already have similar functions.

that the different instances of the same type may have different expected shapes. For instance, a generic Organization shape may require schema:vatID property however for a schema:Organization instance that is the value of organizer property of an event only the name property may be interesting. A SHACL shape that targets the Organization type would verify both Organization instances, which is not the intended behavior. To address this, we see domain-specific patterns as types with local properties and ranges. This means the relevant shape of each instance has to be asserted on that instance. Then the verification turns into instance checking under Closed-World Assumption.

Distinguishing between different kind of constraints may help optimizing cleaning process For commercial applications of knowledge graphs such as the Onlim Conversational Agents, maintaining constant knowledge integrity is crucial, especially in domains like energy and finance. No matter how efficient the constraint checking process is conducted, the processing time may go out of acceptable limits as the knowledge graph size and constraint complexity increase. Distinguishing between constraints involving only metadata and constraints involving property values can help the scalability of constraint checking as the former can be done over the mapping files which has proven to be efficient[3].

Configuration of duplicate detection task and necessary trade-offs Correct configuration of the duplication detection task has technical and conceptual challenges in itself, as it is necessary to identify the right properties and right parameter values for similarity calculation and filtering is important. Fortunately, many tools offer configuration learning to help this process. For applications like conversational agents where time is of the essence, the ability to learn configurations for not only the duplication detection phase but also the preprocessing steps like filtering and indexing may be beneficial. This way the process can be optimized towards precision or recall, depending on the strictness of preprocessing.

4 Conclusion and Future Work

In this paper, we presented our experience with implementing a knowledge graph lifecycle including creation, curation, and deployment. We presented the tasks in the knowledge graph lifecycle and employed a set of tools for many of those tasks. The current implementation is lacking a proper tool to orchestrate the lifecycle but it is currently being developed in use cases like the German Tourism Knowledge Graph. There are still some tasks such as validating knowledge graphs against the real world, fusing linked instances, and automating error correction that requires further research. Moreover, the maturity of our tools is at a different stage, however, they are actively being developed by industrial adopters such as Onlim. We provided an evaluation of different tools supporting the lifecycle individually. Their real evaluation will be in the next couple of years as the developed approaches and tools are continuously being tested in the knowledge graphs and applications of Onlim.

As a knowledge graph gets bigger and supports more applications, it may come to a point that the curation process may be infeasible, both due to the size of the knowledge graph and changing contexts (e.g., different applications and customers may have a different set of constraints and rules). Therefore, our further research will focus also on building a layer on top of knowledge graphs that enables applications to work on small subsets of knowledge graphs with different configurations for curation which will allow the customization of the knowledge graph for different application contexts.

Acknowledgement

This work has been partially funded by the industrial research project MindLab[28].

References

1. Angele, K., Holzknecht, O., Huaman, E., Panasiuk, O., Simsek, U.: D312y2: VeriGraph: A verification framework for Knowledge Integrity. Tech. rep., MindLab Project, Innsbruck, Austria (2020), https://drive.google.com/file/d/1RudX-yt9JxomMb6OBCi4UD10vLtqWZBv/view
2. Athanasiou, S., Giannopoulos, G., Graux, D., Karagiannakis, N., Lehmann, J., Ngomo, A.C.N., Patroumpas, K., Sherif, M.A., Skoutas, D.: Big poi data integration with linked data technologies. In: EDBT. pp. 477–488 (2019)
3. Dimou, A., Kontokostas, D., Freudenberg, M., Verborgh, R., Lehmann, J., Mannens, E., Hellmann, S., Van de Walle, R.: Assessing and refining mappingsto rdf to improve dataset quality. In: Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (eds.) The Semantic Web - ISWC 2015. pp. 133–149. Springer International Publishing, Cham (2015)
4. 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, Vol-1184 (2014), http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf
5. Fensel, D., Şimşek, U., Angele, K., Huaman, E., Kärle, E., Panasiuk, O., Toma, I., Umbrich, J., Wahler, A.: Knowledge Graphs. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-37439-6
6. Garshol, L.M., 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, vol. 7882, pp. 578–592. Springer (2013), https://doi.org/10.1007/978-3-642-38288-8_39

[28] https://mindlab.ai/

7. Kärle, E., Şimşek, U., Fensel, D.: semantify. it, a platform for creation, publication and distribution of semantic annotations. arXiv preprint arXiv:1706.10067 (2017)
8. Ngomo, A.N., 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. 2312–2317. AAAI Press (2011), https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-385
9. Opdenplatz, J.: Duplicate detection as a service (2020), master's Thesis
10. Paulheim, H.: Knowledge graph refinement: A survey of approaches and evaluation methods. Semantic Web Journal **8**(3), 489–508 (2017). https://doi.org/10.3233/SW-160218, https://doi.org/10.3233/SW-160218
11. Sequeda, J.F., Briggs, W.J., Miranker, D.P., Heideman, W.P.: A pay-as-you-go methodology to design and build enterprise knowledge graphs from relational databases. In: Ghidini, C., Hartig, O., Maleshkova, M., Svátek, V., Cruz, I., Hogan, A., Song, J., Lefrançois, M., Gandon, F. (eds.) The Semantic Web – ISWC 2019. pp. 526–545. Springer International Publishing, Cham (2019)
12. Singh, K., Radhakrishna, A.S., Both, A., Shekarpour, S., Lytra, I., Usbeck, R., Vyas, A., Khikmatullaev, A., Punjani, D., Lange, C., et al.: Why reinvent the wheel: Let's build question answering systems together. In: Proceedings of the 2018 World Wide Web Conference. pp. 1247–1256 (2018)
13. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and maintaining links on the web of data. In: Proceedings of the 8th International Semantic Web Conference (ISWC 2009), Chantilly, USA, October 25-29, 2009. Lecture Notes in Computer Science, vol. 5823, pp. 650–665. Springer (2009), https://doi.org/10.1007/978-3-642-04930-9_41
14. Weikum, G., Dong, L., Razniewski, S., Suchanek, F.M.: Machine knowledge: Creation and curation of comprehensive knowledge bases. ArXiv **abs/2009.11564** (2020)
15. 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. Springer (2020)
16. Şimşek, U., Kärle, E., Fensel, D.: Rocketrml - A nodejs implementation of a use-case specific RML mapper. In: Proceedings of 1st Knowledge Graph Building Workshop co-located with 16th Extended Semantic Web Conference (ESWC), to appear. CEUR Workshop Proceedings (2019), http://arxiv.org/abs/1903.04969
17. 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 (jan 2020), http://ceur-ws.org/Vol-2535/paper_10.pdf