Informatics

Extraktion von SHACL Shapes für Evolving Knowledge Graphs

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Eva Pürmayr, BSc
Matrikelnummer 11807199

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr.-Ing. Katja Hose

Wien, 1. Jänner 2024

Eva Pürmayr

Katja Hose

TU WIEN Informatics

SHACL Shape Extraction for Evolving Knowledge Graphs using QSE

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Business Informatics

by

Eva Pürmayr, BSc
Registration Number 11807199

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr.-Ing. Katja Hose

Vienna, January 1, 2024

Eva Pürmayr Katja Hose

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Eva Pürmayr, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Jänner 2024

Eva Pürmayr

Danksagung

Ihr Text hier.

Acknowledgements

Enter your text here.

Kurzfassung

Ihr Text hier.

Abstract

Überarbeiten am Schluss, kopiert von Proposal

Graph databases serve as a foundation for knowledge graphs, which have a broad range of applications. Ensuring data quality is vital and SHACL can be used as a validation language for this purpose. Previously, an approach called QSE (quality shapes extraction) which automatically extracts SHACL shapes from large datasets, has been released. An extension to this program is called Shactor, a web-based tool that visualizes the extraction process and provides statistics. Since knowledge graphs are not static, there exist different versions of a graph, maybe with minimal changes. There is a lack of tools to compare shapes between these graph versions. To address this gap, the proposed solution involves creating a web-based tool and algorithm adaptions in this niche.

Contents

CHAPTER 1

Introduction

Storing complex and interconnected data is quite a challenge for data engineers. Knowledge graphs offer a versatile solution and have a broad range of applications, both in industry and in academia. For instance, knowledge graphs can serve as the foundation for machine learning algorithms. A prominent example of a knowledge graph is DBpedia [19], which contains the data behind Wikipedia. This example highlights the potential size of knowledge graphs. Beyond DBpedia, there are many other examples of knowledge graphs across various domains.

Given the wide range of applications and the massive amounts of data involved, maintaining data quality is crucial. Changes to knowledge graphs can occur frequently and may originate from different stakeholders, sometimes introducing errors. For instance, the data in DBpedia changes at least daily, with numerous users from around the globe contributing to these updates. The potential errors in such large knowledge graphs pose a specific problem for applications that rely on the trustworthiness of data within a knowledge graph. Ensuring data quality is therefore vital.

Knowledge graphs can be expressed in the Resource Description Framework (RDF) [12], and data quality can be ensured using the validation language SHACL [18]. With SHACL, a schema, in the form of so-called shapes, can be created, which can be used to validate a graph. During this validation, erroneous data can be filtered out, enhancing overall data quality. Creating such schemas manually is impractical - therefore, algorithms have been developed that automatically extract SHACL shapes from large knowledge graphs. One such algorithm is QSE (Quality Shapes Extraction) [74], which uses parameters to improve the quality of the generated schema even further. This is crucial because an automatically created schema itself can contain errors if it is based on incorrect data. Once a schema is generated, it can be used to validate existing data and filter out incorrect data.

Another important aspect in this area is the evolution of knowledge graphs. Since knowledge graphs are not static, there may exist different versions of a graph, maybe with minimal changes. This evolution is particularly relevant to the topic of data quality.

1.1 Problem Statement

The evolution of knowledge graphs increases the complexity of schema creation to ensure data quality. QSE was written with a focus on the extraction of SHACL shapes for a specific version of a graph, but it does not provide a way to compare these shapes between different graph versions. It would be beneficial for users to identify shapes that remain unchanged across multiple versions of a graph or to detect shapes that have changed. Currently, the state of the art is to manually compare the extracted shapes between versions, using a text comparison tool. Using manual methods is the only option, but this is tedious and lacks practicality.

Another problem in this area is the inefficiency of running QSE independently on each version of a graph, even when there are only minimal changes in the data and schema.

In summary, this work aims at shapes extraction and comparison, specifically in the context of evolving knowledge graphs. As there is currently no tool available for this use case, the characteristics of usability and execution time remain unmeasurable.

1.2 Goals and Expected Outcome

To overcome these issues, a user interface will be created, which allows users to conveniently compare SHACL shapes between different versions of a graph. For reusability, graph versions and extracted shapes will be stored in a local database. The tool will allow users to identify which shapes remained the same, have changed, been added, or been deleted. Additionally, it will provide information on why shapes were deleted. These features will apply not only to two graph versions but to multiple versions.

The second main goal of this thesis is to explore various methods for making the SHACL shape extraction more efficient for evolving knowledge graphs. One approach is to validate existing SHACL shapes generated by QSE on a second version of a graph using the query language SPARQL. This reduces the execution time for the second version, although it has the drawback of not generating shapes for newly added data. Another approach is to use the changeset between two graph versions and the results from the first QSE run to build the shapes for the second version. This method also reduces the execution time but requires the changeset between graph versions. Lastly, an attempt is to generally reuse the shapes generated by QSE in the first run during a subsequent run on another version to minimize the execution time.

Summing up, after this thesis is finished, users should be able to conveniently and efficiently compare SHACL shapes across various versions of a knowledge graph.

Reorder Research questions?

1.3 Research Questions

To specify these goals, the following research questions have been elaborated. There are two versions of a graph, G_1 and G_2, and the corresponding SHACL shapes generated by QSE, S_1 and S_2.

RQ1: What is an appropriate way to compare S_1 and S_2 in a user-friendly way and explain the differences (addition, removal, change)?

RQ2: Given S_1 and the changeset between G_1 and G_2, how can we use the changeset and S_1 to derive S_2?

RQ3: What is an appropriate extension of the QSE algorithm so that - after executing standard QSE on G_1, we can parse G_2 to obtain the changed shapes without having to run the complete QSE algorithm again on G_2?

RQ4: Given SPARQL endpoints hosting the graphs, how can we use SPARQL queries to derive which shapes of S_1 remain unchanged and which were removed for G_2?

Appropriate for RQ1 means user-friendly, useful, and correct. It can be measured during the evaluation of semi-structured interviews with experts. Appropriate for RQ2, RQ3, and RQ4 means correct and faster in comparison to the method described in the evaluation section.

CHAPTER 2

Related Work

This chapter delivers a comprehensive overview of knowledge graphs, RDF, SPARQL, and SHACL. Building on this foundation, a dedicated section explores evolving knowledge graphs, which is particularly important to this subject. Additionally, the chapter presents related work retrieved from a systematic literature review, along with a description of the QSE algorithm which forms the basis of this thesis.

2.1 Preliminaries

The following pages will cover fundamental aspects of semantic systems. Topics include an overview of knowledge graphs, RDF (Resource Description Framework), the query language SPARQL, and the validation language SHACL.

2.1.1 Knowledge Graphs

The topic of this thesis is located in the area of semantic systems and knowledge graphs. Semantic systems are systems that make use of explicitly represented knowledge, through conceptual structures such as ontologies, taxonomies, or knowledge graphs [52]. Many formal definitions exist for knowledge graphs, such as the one articulated by professors at TU Wien: "A knowledge graph contains semantically related information as nodes and edges" or "Knowledge Graphs are graph-structured representations intended to capture the semantics about how entities relate to each other" [72]. Knowledge graphs find application across diverse domains, including commercial and academic sectors. Moreover, they serve as the foundation for Data Science, Graph Databases, Machine Learning, Reasoners, and Data Integration/Wrangling. Noteworthy among these knowledge graphs is DBpedia [19], which contains cleaned data from Wikipedia in all languages. In 2023, DBpedia's Databus, the platform hosting its data, provided 4,100 GByte of data [4]. Additionally, other prominent knowledge graphs exist, such as Wikidata [17], compromising approximately

110 million editable records. Wikidata serves as the central repository for Wikimedia projects like Wikipedia, Wikivoyage, Wiktionary, Wikisource, and others. It is a free and open knowledge base. Besides general information, as it is provided by DBpedia, knowledge graphs have found compelling applications in fields such as chemistry and biology. For instance, Linked Life Data [9] offers access to 25 public biomedical databases, facilitating questions like "find all human genes located in Y-chromosome with the known molecular interactions". Around 10 billion RDF statements are provided by Linked Life Data. Finally, PubChem [73], an open chemistry database, is another notable example of a large knowledge graph.

2.1.2 RDF

Knowledge graphs can be expressed as property graphs or by the Resource Description Framework (RDF) [52]. The primary distinction lies in the fact that property graphs allow edges to have attributes. However, this thesis focuses on RDF, which was developed in the 1990s and is a W3C recommendation [12]. The primary strength of RDF lies in its ability to associate metadata with resources. An RDF graph contains numerous statements, known as triples, each consisting of a subject, predicate, and object. An example is illustrated in Figure 2.1.

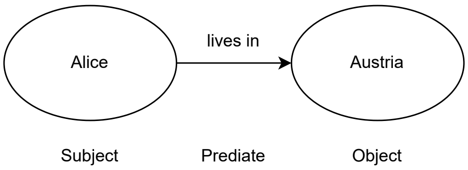


Figure 2.1: Example of RDF

An important term in knowledge graphs is URI, which stands for Uniform Resource Identifier. In RDF, anything with a URI is considered a resource. URIs are strings that identify objects and resources typically found on the web. However, it is important to note that resources do not have to be on the web. A further important property of URIs is their uniqueness. A URI can be a Uniform Resource Name (URN), a Uniform Resource Locator (URL), or a combination of both. An Internationalized Resource Identifier (IRI) is a URI. An example of defining a resource using a URI could be "http://semantics.id/ns/example#Alice". To build RDF triples, the RDF data model was defined which includes four sets [80]:

- The set of resources $\mathcal{R}$ contains all entities for which RDF statements can be made, and these are identified by URIs.
- The set of properties $\mathcal{P}$ lists all features with values that can be attached to resources. Properties, which are also RDF resources identified by URIs, define the

relationships between subjects and objects in an RDF triple and are referred to as predicates. These properties form a subset of all resources.

- The set of literals $\mathcal{L}$ includes other values such as character sequences, integers, decimals, dates, or booleans.
- Lastly, the set of statements $E : \langle s, p, o \rangle$ contains all triples in the graph. As previously mentioned, a triple contains a subject, a predicate, and an object. The subject of an RDF statement is an RDF resource or a blank resource, the predicate is an RDF property. The object can be either an RDF resource, a literal, or a blank resource. For instance, an adapted example from the one above could have the predicate "age" and the object "25", where the object is a literal, unlike the original example where the object is a resource.

As blank resources or blank nodes were already mentioned, they exist besides RDF resources and literals. They are used to improve the information structure and are not globally unique. For example, a list cannot be modeled as an RDF triple without using blank nodes. The definition of RDF graphs can be formally articulated as:

Definition 1 *(RDF graph [74]). Given the sets of resources $\mathcal{R}$, literals $\mathcal{L}$ and blank nodes $\mathcal{B}$, an RDF graph $\mathcal{G} : \langle N, E \rangle$ is a graph with nodes $N \subset (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})$ and edges $E \subset \{\langle s, p, o \rangle \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{R} \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})\}$*

A small example of a full graph can be seen in Listing 2.1. In this example, two people are defined, who both have a name (Bob and Alice) and mutually know each other. Prefixes are defined, which can be used in RDF to shorten URIs. For instance, "foaf:name" abbreviates "<http://xmlns.com/foaf/0.1/name>". A dot always indicates the end of an RDF statement.

```
 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://example.org/alice> a foaf:Person .
<http://example.org/bob> a foaf:Person .
<http://example.org/alice> foaf:name "Alice" .
<http://example.org/bob> foaf:name "Bob" .
<http://example.org/alice> foaf:knows
    <http://example.org/bob> .
<http://example.org/bob> foaf:knows
    <http://example.org/alice> .
```

Listing 2.1: RDF graph in Turtle syntax

Furthermore, the letter "a" indicates, that the resource "Alice" is a Person. This predicate a, which is an element of $\mathcal{P}$, abbreviates the property rdf:type [12]. It links all instances

of a class to the node representing the class. Formally, all classes, which form a subset of $\mathcal{R}$, can be defined as $C : \{c \in \mathcal{R} \mid \exists s \in \mathcal{R} \text{ so that } \langle s, a, c\rangle \in \mathcal{E}\}$.
RDF graphs can be represented in various formats, such as RDF/XML, N3/Turtle, or in N-Triples format, which is shortened as "nt" or generally as a labeled directed graph. An example of an RDF graph in N-Triples format based on Listing 2.1 in a shortened version is shown in Listing 2.2.

```
<http://example.org/alice>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person> .
<http://example.org/alice>
<http://xmlns.com/foaf/0.1/name> "Alice" .
<http://example.org/alice>
<http://xmlns.com/foaf/0.1/knows> <http://example.org/bob> .
```

Listing 2.2: RDF graph in N-Triples Format

Knowledge graphs can be stored by different graph stores, for instance, GraphDB [7]. GraphDB is provided by Ontotext, available as a free or commercial version. It offers a visual interface for the interaction with graphs but also supports integration with various programming languages. An alternative is OpenLink Virtuoso [10] which can also be used in Microsoft Azure or Amazon AWS. To work with knowledge graphs, there exist different frameworks, such as Apache Jena [2] or Eclipse RDF4J [35] which are both solely compatible with Java.

2.1.3 SPARQL

Knowledge graphs can be queried by the query language SPARQL [15]. SPARQL shares similarities with SQL, as it incorporates familiar keywords like "select", "where", "group by", and "order by". An example query, as shown in Listing 2.3, retrieves the names of people along with the number of their friends.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(?friend) AS ?count)
WHERE {
    ?person foaf:name ?name .
    ?person foaf:knows ?friend .
} GROUP BY ?person ?name
```

Listing 2.3: SPARQL example

The WHERE clause specifies a pattern to be matched in the graph. The variable "?person" represents an RDF resource in the graph and is further constrained by the statements, that the person must have a name and a friend. SPARQL offers various other functionalities, including insert, update, and delete statements, filtering, ordering, and subqueries.

2.1.4 SHACL

Since data quality is an important topic in all areas, SHACL [13], which stands for Shapes Constraint Language, has evolved as a language to validate RDF graphs against a certain set of conditions. SHACL can be expressed as an RDF graph itself, it can also serve as the description of the data graph. This representation is termed a "shapes graph" S, whereas the actual dataset is called the "data graph" G. Shapes graphs contain different shapes, which describe certain classes and their properties.
A key concept within SHACL is the node shape, which is often used to describe a class in the RDF graph and may contain multiple property shapes. A property shape is usually used to describe the objects of a property of a certain class. Mathematically, this can be defined as:

Definition 2 *(Shape graph [74]). A shape graph S contains node shapes N, with $\langle s, \tau_n, \Phi_n \rangle \in N$ where s is the subject IRI of the node shape (or the name), $\tau_n \in C$ is the target class, and Φ_n are the property shapes, in the form $\phi_n : \langle \tau_p, T_p, C_p \rangle$. τ_p is the target property, defined with "sh:path" and $T_p \subset R$ contains, in case of the node kind of τ_p "sh:Literal", an IRI describing the literal type e.g. "xsd:string". In case of the node kind of τ_p "sh:IRI", a set of IRIs is provided. C_p is a pair $(n, m) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ with $n \leq m$, which describes the min and max cardinality constraints.*

An illustrative example is presented in Listing 2.4, where the validation of the class "ex:Person" is specified. Every person must have at least one name.

```
ex:PersonShape a sh:NodeShape ;
sh:targetClass ex:Person ;
sh:property [
    a sh:PropertyShape;
    sh:path ex:name;
    sh:minCount 1
] .
```

Listing 2.4: SHACL example

SHACL shapes can have different targets. In this example, the target is the class ex:Person, specified using the predicate "sh:targetClass". The prefix "sh" abbreviates SHACL i.e. the URI "http://www.w3.org/ns/shacl#". However, targets of shapes can be defined in different ways, not necessarily limited to classes. For instance, a shape might exclusively target several nodes, denoted by "sh:targetNode". Alternatively, it is possible to define a predicate, e.g. "ex:knows" and target subjects or objects of this predicate, where the IRIs "sh:targetSubjectsOf" and "sh:targetObjectsOf" would be used. However, in this thesis, targets of node shapes will be only identified with "sh:targetClass". The IRI of the node shape shown in Listing 2.4 is "ex:PersonShape", which can also be considered as the name of the node shape. Property shapes can also have IRIs that identify them; in this example, it could be "ex:namePersonShapeProperty".

Besides various SHACL keywords used for property shapes, some important IRIs include "sh:NodeKind". This IRI defines whether the object of the specified predicate is "sh:Literal" or "sh:IRI", among other possible values. Another constraint is "sh:datatype" which states the data type of the target objects, such as "xsd:integer" or "rdf:langString". The "sh:class" predicate is used to ensure that the target object is an instance of the specified class. Each property shape mandatorily requires a predicate "sh:path", whose object specifies an IRI declaring the focus nodes of property shapes reachable from the node shape via this path. Property shapes can also include cardinality constraints like "sh:minCount" or "sh:maxCount". Additionally, SHACL supports lists with "sh:in" and logical constraints with "sh:or" to express more complex shapes. SHACL provides many more options than those listed here, enabling a detailed description of a graph's schema.
When a data graph is validated against a shapes graph, it is verified that all target entities fulfill all property shapes in the given node shape. This can be expressed in the following way:

Definition 3 *(Validating Shapes Semantics [74]). All entities e, which are instances of τ_n, in a graph G are validated by a node shape n, if the following conditions are fulfilled for all property shapes $\phi_n : \langle \tau_p, T_p, C_p \rangle$ in n:*

- *If the node kind of ϕ_n is "sh:Literal", then for every triple $(e, \tau_p, l) \in G$, l is a literal of type T_p*
- *If the node kind of ϕ_n is "sh:IRI", then for every triple $(e, \tau_p, o) \in G$, o is an instance of type C_p, or of one of its subclasses*
- *$n \leq |\{(s, p, o) \in G : s = e \wedge p = \tau_p\}| \geq m$, where $C_p = (n, m)$*

There also exist other logic-based languages, which can describe the schema of a graph, namely the Web Ontology Language (OWL) [11], or Shape Expressions (ShEx) [14].

2.2 State of the art

After understanding the fundamentals of semantic systems, specifically RDF and SHACL, the subsequent section dives into the central themes of this thesis: evolving knowledge graphs and the QSE algorithm. This section also covers the findings from the literature review, including topics such as data quality in knowledge graphs and other algorithms for extracting shapes from existing graphs.

2.2.1 Evolving Knowledge Graphs

As one might expect, knowledge graphs are not static; they evolve over time just like the reality they present. DBpedia [19], for instance, is updated regularly as new knowledge is generated continuously.
Time can be represented in evolving knowledge graphs in different ways [72]. It can be

saved as data, for example, the construction year of a building can be recorded in the knowledge graph. Contrastingly, time can also be saved as metadata. This includes the creation time of an entry, the time an entry was deleted, and the existence of different versions of an entry over time. Consequently, when time is saved as metadata, a knowledge graph can be dynamic – providing access to all observable atomic changes over time – or versioned – offering static snapshots of the knowledge graph at specific points in time. In this thesis, only versioned knowledge graphs will be considered.

When discussing evolution, several new terms must be considered. First, structural evolution can be measured, where different descriptive statistics, such as centrality or connectedness are measured over time within a graph. Dynamics in an evolving knowledge graph can be assessed by examining growth and change frequencies, which can be interesting to compare across different areas of the graph. Timeliness refers to the freshness of the data, indicating if data is out-of-date or out-of-sync. Monotonicity describes whether data is only added to the graph or if there are also updates and deletions. Semantic drift happens when there is a change in the meaning of a concept over time.

It is also important to consider who makes changes in a knowledge graph. These changes can be made by anonymous users, registered users, authoritative users, or bots. Understanding who is making the changes is crucial, as it can impact data quality in various ways.

Often, when changes in knowledge graphs occur, they primarily happen at the level of object literals [72]. Changes can be atomic, focusing on operations at the resource level, or they can be local or global. Changes can also be monitored at the schema level. In a study of the DyLOD dataset (Dynamic Linked Data Observatory) over one year, between 20% and 90% of schema structures changed between two versions. This indicates that while some nodes retained the same schema structure, new schema structures were added, and some were no longer used. This observation is particularly important for this thesis.

However, the dimension of time opens new challenges in knowledge graphs. For instance, there is a need for new data models and storage methods tailored to evolving knowledge graphs. Additionally, there will be novel approaches to query processing, reasoning, and learning that incorporate the temporal aspect in evolving knowledge graphs.

Based on this knowledge, various projects have been developed, and extensive research has been conducted in this area. For instance, EvolveKG [62] is a framework designed to reveal cross-time knowledge interactions. Another approach [81] involves creating a summary graph for different versions of objects, which refers to the dynamic option of evolving knowledge graphs, as discussed earlier. Additionally, KGdiff [59] has been developed to track changes in both the schema and the individual data points.

2.2.2 Data Quality in Knowledge Graphs

Numerous approaches have also been developed in the area of data quality which is a vital topic for knowledge graphs due to the huge amount of data involved and the schema flexibility. GraphGuard [37] introduces a framework aimed at improving data quality in knowledge graph pipelines for both humans and machines. Additionally, there has been

a systematic review [86] of quality management in knowledge graphs, as well as a general survey [56] on knowledge graphs, including the temporal aspects of them. Another paper [53] aims to enhance existing quality assessment frameworks by incorporating additional quality dimensions and metrics.
Also in the area of quality, a method [66] has been developed to judge, whether an incoming graph change is correct or not with classifiers based on topographical features of a graph. PAC (property assertion constraints) [36] has the goal of checking data before it gets added to the knowledge graph. With this approach, errors can be prevented and the quality of knowledge graphs can be enhanced. PAC works by restricting the range of properties using SPARQL. Additionally, a dissertation [33] has focused on incorporating data transformations in knowledge graphs to clean the data and complete the graphs by calculating derived data.

2.2.3 Quality Shapes Extraction

QSE (Quality Shapes Extraction) [74] is an algorithm designed to extract SHACL shapes from knowledge graphs to ensure data quality. It is particularly designed for very large knowledge graphs and it is implemented in Java. A main advantage of QSE is that it eliminates the need for manual steps during shape generation as it would be unmanageable for huge datasets such as Wikidata, where there are approximately 2 million property shapes to identify. Furthermore, a key objective is to eliminate spurious shapes that can be generated during automatic shapes extraction. For instance, in DBpedia, some entities which represent musical bands are incorrectly assigned to the class "dbo:City". Without filtering, these errors would be replicated in the extracted shapes.
To address this issue, QSE calculates two parameters: support and confidence. The support parameter determines the number of entities associated with a certain class for a node shape while for a property shape, it defines the cardinality of entities conforming to that class. The confidence parameter measures the ratio between the number of entities that conform to a property shape and the total number of entities that are instances of the target class. Formally, this can be described in the following way:

Definition 4 *(Support [74]). Given a node shape $\langle s, \tau_n, \Phi_n \rangle \in N$ and a property shape $\phi_n : \langle \tau_p, T_p, C_p \rangle \in \Phi_n$, the support is defined as the number of entities e which satisfy ϕ_n*

$$supp(\phi_n) = |\{e \in R, \text{ which satisfy } \phi_n\}|$$

Definition 5 *(Confidence [74]). Given a node shape $\langle s, \tau_n, \Phi_n \rangle \in N$ and a property shape $\phi_n : \langle \tau_p, T_p, C_p \rangle \in \Phi_n$, the confidence is defined as the proportion of entities which fulfill ϕ_n among the entities that are instances of the target class τ_n of s*

$$conf(\phi_n) = \frac{supp(\phi_n)}{|\{e|\langle e, a, \tau_n \rangle \in \mathcal{E}\}|}$$

When these parameters are configured, QSE generates shapes only if they exceed the specified thresholds for support and confidence values. If these thresholds are not set,

QSE generates all shapes, which are referred to as the default shapes.
QSE is available as a command-line tool, primarily designed to extract SHACL shapes from graphs provided as a file, which must be formatted as "N-Triples". Moreover, QSE offers a query-based option for graphs that are stored on graph stores such as GraphDB. To generate shapes from a graph, it is necessary to analyze all nodes and their types in a graph, as nodes can be used as subjects or objects. Furthermore, it is necessary to count how often a property connects nodes of two given types. This is done in four steps, where the QSE algorithm involves two iterations through all triples. During the initial phase (entity extraction), all instances based on their types are counted, hence only triples containing a type declaration (e.g. "rdf:type") are considered. The algorithm stores the entity types associated with each entity and calculates the total count for each class. Subsequently, in the second run (entity constraints extraction), the algorithm gathers metadata for property shapes. Here, all triples without a type declaration are analyzed. The subject and the object types for each predicate are saved. The results, just as the previous data, are stored in maps. Following this, in the third step, support and confidence metrics are computed from the previously saved data. Based on these metrics, spurious shapes can be filtered out. Finally, in the last step, the algorithm generates SHACL shapes. The names of the shapes are declared and the information gathered so far is brought to the node- and the corresponding property shapes. To output the shapes, they are stored locally in a triple store and exported as a turtle file. Additionally, Java objects representing the node and property shapes exist, which can be used in subsequent Java programs, but are not visible to the end user.
Due to the potential size of knowledge graphs, QSE is offered in two versions: the exact version, where the steps are described in detail above, and the approximate version. The exact version keeps type and property information in memory which can lead to a high memory consumption. QSE-Approximate was developed to enable shape extraction on commodity machines with limited memory. This algorithm is based on a multi-tiered dynamic reservoir sampling algorithm which replaces the first phase of QSE-Exact where all triples are read.
Another configuration possibility for QSE is to list the names of the classes C on which QSE should be executed. This feature allows for the extraction of shapes for specific classes only, rather than running QSE on the entire graph.

Example of graphs and QSE output

To better grasp the theory behind QSE, a simple example with Alice and Bob is provided. A snippet from this knowledge graph is presented in Listing 2.2, however, the complete RDF graphs are available in Appendix B, in Listing B.1, Listing B.2, and Listing B.3. A third person, Jenny, is added to the graph. After executing QSE-Exact without parameters for support and confidence, the full output is shown in Listing B.4. Listing 2.5 shows the node shape generated for the class "Person". This can be seen in the last triple of the snippet, by the property "targetClass". The node shape has a support of three because there are three people in the knowledge graph. Furthermore, there are three property shapes in the node shape, the first one describing the "rdf:type" property for the instances of the "Person" class, and the second one is about the "knows" property, where each person knows another person. Lastly, the property shape described in Listing 2.6, is about the "name" property. It shows that the property is set for Alice, Bob, and Jenny and therefore the confidence is 100%, which is written as "1E0". The object of these triples must be a literal of the type "xsd:string" and because the property is set for all instances, "minCount=1" is added to the shapes, marking it as obligatory. The "path" describes the property, namely "<http://xmlns.com/foaf/0.1/name>".

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://shaclshapes.org/PersonShape> rdf:type
    <http://www.w3.org/ns/shacl#NodeShape> ;
  <http://shaclshapes.org/support> "3"^^xsd:int ;
  <http://www.w3.org/ns/shacl#property>
    <http://shaclshapes.org/instanceTypePersonShapeProperty> ;
  <http://www.w3.org/ns/shacl#property>
    <http://shaclshapes.org/knowsPersonShapeProperty> ;
  <http://www.w3.org/ns/shacl#property>
    <http://shaclshapes.org/namePersonShapeProperty> ;
  <http://www.w3.org/ns/shacl#targetClass>
    <http://xmlns.com/foaf/0.1/Person> .
```

Listing 2.5: Result of QSE-Exact on the people knowledge graph, node shape for the class "Person"

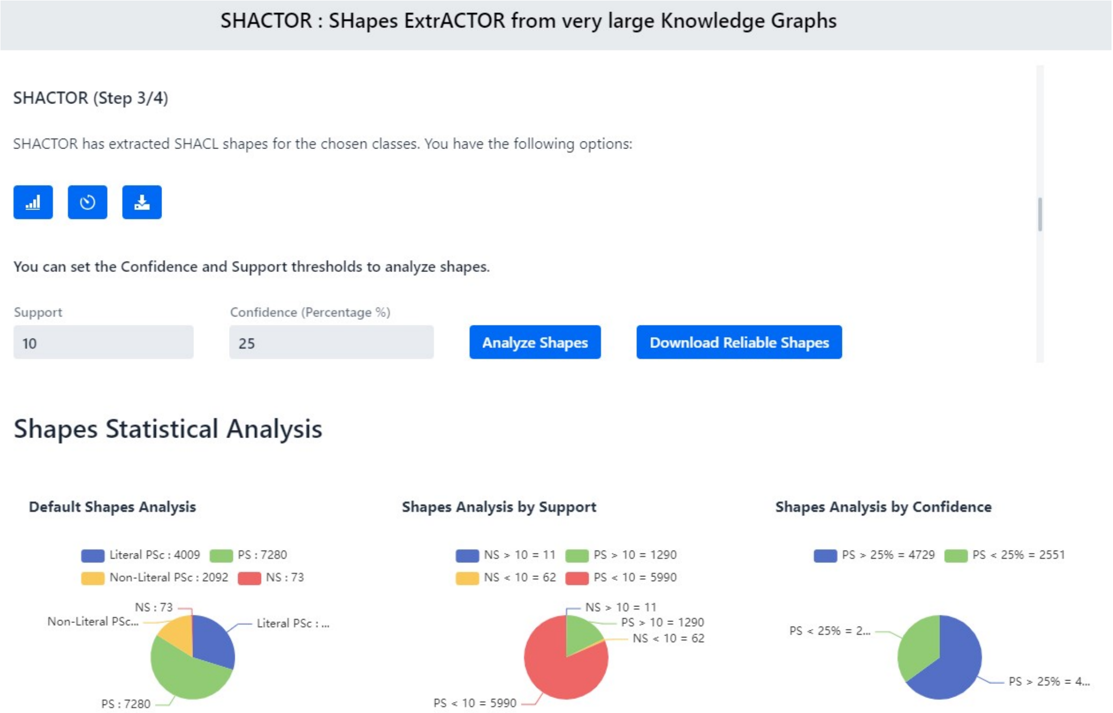
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://shaclshapes.org/namePersonShapeProperty> rdf:type
    <http://www.w3.org/ns/shacl#PropertyShape> ;
  <http://shaclshapes.org/confidence> 1E0 ;
  <http://shaclshapes.org/support> "3"^^xsd:int ;
  <http://www.w3.org/ns/shacl#NodeKind>
    <http://www.w3.org/ns/shacl#Literal> ;
  <http://www.w3.org/ns/shacl#datatype> xsd:string ;
  <http://www.w3.org/ns/shacl#minCount> 1 ;
  <http://www.w3.org/ns/shacl#path>
    <http://xmlns.com/foaf/0.1/name> .
```

Listing 2.6: Result of QSE-Exact on the people knowledge graph, property shape for the property "knows" from the class "Person"

Shactor

The visual extension to QSE is called Shactor [75]. It is a web-based tool that visualizes the extraction process and provides valuable statistics. Shactor was developed using



SHACTOR : SHapes ExtrACTOR from very large Knowledge Graphs

SHACTOR (Step 3/4)

SHACTOR has extracted SHACL shapes for the chosen classes. You have the following options:

You can set the Confidence and Support thresholds to analyze shapes.

Support: 10

Confidence (Percentage %): 25

Figure 2.2: Statistics provided by Shactor after analyzing version 1 of the Bear-B dataset

Vaadin [16] and supports only the file-based version of QSE. An example screenshot after the extraction process is shown in Figure 2.2

Limitations in the QSE algorithm

Not finished yet, only notes... Is this the correct section for limitations?

- Problems, when same but different IRIs are used: http://dbpedia.org/property/almaMater or http://dbpedia.org/ontology/almaMater -> This was not implemented on purpose, it should be two shapes. Also leads to confusing shapes, e.g. two or-items in shape with same classes (from different node shapes originally) e.g. BearB-V1 religion_1PersonShapeProperty, two support and confidence entries, property shapes are added up for both classes in the same shape (testen)

- QueryBased: sometimes unclear empty shapes, because of incorrect data

- Shactor: Example Film: If all classes are selected and Support is set to 3 (confidence 1), dateOfBirth (ScriptWriter, support = 1) is also printed. If only ScriptWriter class is selected, dateOfBirth property is not printed.

- When a comment is added at the beginning of the file, everything crashes. E.g. Bear-B first line: #V 2015-08-01T21:33:00Z

- InstanceShapes get filtered in QueryBased, but not in FileBased
- Shactor: Encoding issues for special characters: e.g.
  <http://dbpedia.org/resource/2015_US_Open_(tennis)>
  <http://dbpedia.org/ontology/budget> "4.22534E7"^^ <http://dbpedia.org/datatype/usDollar>
- Pruning works with >, not with >=, which can be very confusing
- Objects in Shactor only support only one targetClass, not multiple

Other shapes extraction algorithms

Many algorithms follow the idea of extracting SHACL shapes or other schema information from existing data sources to improve data quality and eliminate the need for manual schema creation. Examples include SHACLGEN [57] or ShapeDesigner [26], which follow a similar approach as QSE-Exact but lack the ability to process large knowledge graphs [74]. Additional examples include Shape Induction [65] and the approach described by Spahiu et al. [79] which both extract SHACL shapes from existing data. SCOOP [38] similarly extracts SHACL shapes from existing knowledge graphs but uses ontologies and data schemas primarily rather than individual entities. IOP [8, 67] is a predicate logic formalism that identifies specific shapes over connected entities in the knowledge graph, with its corresponding learning method known as SHACLearner. SheXer [42], using a Python library, produces SHACL shapes similar to QSE. A logic called ABSTAT [79] which automatically extracts SHACL shapes from knowledge graphs has already been considered in QSE. Another approach, the Ontology Design Pattern [68], uses ontologies to automatically generate SHACL shapes. Astrea [31] employs Astrea-KG, which provides mappings between ontology constraint patterns and SHACL constraint patterns. While not generating SHACL shapes itself, Trav-SHACL [44] plans the execution and traversal of a shapes graph to detect invalid entries early.
Also not directly related to shapes extraction, visualizing SHACL shapes has also been explored. A master's thesis [22] was written that generates a 3D model of interconnected SHACL shapes.

CHAPTER 3

Methods

This chapter describes the scientific methods that are used in this work. First, all methods are described individually, and then the process of answering the research questions using the mentioned methods is explained.

3.1 Research Methods

This section explores the scientific methods used in this thesis, namely Design Science Research, Systematic Literature Review, prototyping, semi-structured expert interviews, qualitative content analysis, algorithmic design, and technical experiments.

Delete Research Methods again and keep only Methodological steps?

3.1.1 Design Science Research

Design Science [51], rooted in engineering, focuses on problem-solving through the creation of useful artifacts. In Design Science Research design is both a process and a product and the evaluation of both is crucial. This leads to a continuous cycle where artifacts undergo regular evaluation and subsequent refinement, often referred to as the Design Cycle [50].
Two important terms in design science are relevance and rigor. The artifact's relevance to its environment (people, organizations, technology) is crucial. This relationship forms a cycle: the artifact's development is shaped by the environmental needs and, in turn, influences that environment through its application, termed the Relevance Cycle. Simultaneously, ensuring rigorous development involves addressing existing knowledge. Here, the knowledge base influences the design cycle, while the finished product contributes back to the knowledge base in what's known as the Rigor Cycle.
The heart of Design Science is the iterative Design Cycle, with Relevance and Rigor Cycle as its inputs. These cycles depend on each other, although during the actual research execution, the Design Cycle remains relatively independent.
Hevner proposed seven guidelines for the design science framework:

1. Design as artifact: an artifact (construct, model, method, instantiation) must be produced
2. Problem relevance: ensuring the solution is relevant to business problems
3. Design evaluation: evaluation methods must be executed to ensure the utility, quality, and efficacy
4. Research contributions: contributions in the area of design artifact, design foundations, and/or design methodologies must be provided
5. Research rigor: employing rigorous methods during construction and evaluation
6. Design as a Search Process: searching for an artifact requires available means to reach desired ends while satisfying laws
7. Communication of Research: the result must be presented to a technology-oriented and management-oriented audience

An especially important topic within the design cycle revolves around evaluation [70]. This involves establishing specific criteria, such as functionality, performance, or usability as benchmarks for assessment. The study has evaluated several artifact types (e.g. algorithm, instantiation, construct) and evaluation method types (e.g. expert evaluation, technical experiment, prototype). The choice of evaluation method depends on the type of artifact; technical experiments were predominantly used for algorithms, instantiations, methods, and models.

3.1.2 Systematic Literature Review

Both rigor and relevance cycle require an understanding of the current state of the art. Particularly, the relevance cycle relies on identifying existing research gaps, a task that will be addressed through a partial systematic literature review (SLR) [60]. Typically, a systematic literature review includes the following steps:

1. Planning: This phase begins by identifying the need for an SLR, (optionally) commissioning the SLR, and defining research questions. After that, the review protocol must be developed and evaluated, which includes search terms, resources where to search, study selection criteria, and a quality checklist. A pilot phase helps refine these elements.
2. Conducting: Firstly, the research must be identified and primary studies should be selected. After that, the study quality must be assessed, followed by data extraction, monitoring, and synthesis.
   It is important for the search terms to also consider synonyms, abbreviations, and combinations. Documentation throughout ensures transparency and replicability. After all primary papers have been obtained, the papers must be filtered. Instead

of reading all the papers, some papers can be dropped based on the titles and then on the abstracts (study selection criteria). Then the remaining papers can be read and further reduced. In the end, data needs to be extracted with a pre-defined form, if possible, by multiple reviewers. During data synthesis, data is summarized. This can be done descriptively, qualitatively, or quantitatively.

3. Reporting: The last step involves specifying the dissemination mechanism (how and where the results should be published, e.g. in a magazine, journal articles, or on a website), formatting, and evaluating the report. During the evaluation, peer reviewing can be used e.g. for journal articles. Quality checklists and other prepared documents from the planning phase can be utilized here.

3.1.3 Prototyping

Within the design cycle of Design Science Research (DSR), an artifact is developed. Prototyping [55] has parallels to software engineering, which often uses agile development cycles with frequent feedback. The primary aim is to construct the Minimum Viable Product before completing the development process, aligning with the Plan-Do-Check-Act methodology.

Camburn et al. [27] describe prototypes as a pre-production representation of some aspect of the final design. Prototypes can be used for refinement, communication, exploration of new concepts, and learning. Noteworthy guidelines include testing, timing, ideation, fixation, feedback, usability, and fidelity. Various prototyping techniques exist, such as iterative, parallel, requirement relaxation, or subsystem isolation, each offering distinct advantages for particular objectives. Despite the diverse techniques, detailed guidance on prototyping remains limited, except for the iterative aspect.

3.1.4 Semi-structured expert interviews

Semi-structured interviews (SSIs) [21] are a qualitative evaluation method within the design cycle, allowing feedback integration into the construct phase. Unlike traditional surveys, SSIs engage fewer participants, employing open-ended questions that focus on 'how' and 'why', and the structure is not fixed. The disadvantages of SSIs are that they are time-consuming, labor-intensive, and require interviewer sophistication. SSIs prove beneficial when seeking open responses and individual perspectives, suitable for groups like program recipients or interested stakeholders.

The methodological steps include selecting respondents and arranging interviews. The target group should be identified, researchers choose ordinarily a manageable random sample. A brief introduction letter detailing the interview's time frame is crucial before sending invitations. This should be piloted before sending. Next, the questions should be drafted and an interview guide should be created. There should not be too many issues on the agenda. Closed-ended questions can be an ideal start, open-ended questions can follow these questions. Attention to translations and avoiding questions that evoke pressure to give socially acceptable answers is essential. Flexibility during the interview

allows agenda adaptation, keeping easy questions at the start, challenging ones at the end, and demographic inquiries at the interview's conclusion.
During the interview, the interviewer should establish a positive first impression and ask for permission to record the interview. This allows the interviewer to participate more actively. Preparation ensures clear prioritization of questions. Interviewers should maintain a calm, listen actively, and seek clarification when necessary.
The analysis involves visualizing closed-ended responses in tables/graphs and employing qualitative content analysis to categorize open-ended ones.

3.1.5 Qualitative Content analysis

Qualitative Content analysis [64] is an approach of systematic, rule-guided qualitative text analysis that can be used in the design cycle of DSR. It is used for fitting text material into a model of communication. The aspects of text interpretation will be put into categories, that follow the research question. There are two ways for category development, namely inductive and deductive category development. Inductive category development forms the categories out of the text, whereas deductive formulates the categories more on theory.
The detailed steps of inductive category development involve, based on the research question, the determination of the category definition. Then, categories are formulated based on the material. after 10-50% of the material, the categories are revised and the process starts again. After working through all the material, everything should be checked for reliability and the results can be interpreted, which can trigger another iteration.

3.1.6 Algorithmic Design

During the design cycle, another method applicable is Algorithmic Design [78], also known as Algorithm Engineering, a scientific approach to developing efficient algorithms. The method involves iterative steps: design, analysis, implementation, and experiments. A key focus lies in crafting algorithms that are simple, implementable, and reusable, with performance being a crucial factor.
It is crucial in the analysis phase, that the design algorithm is easy to analyze, to close the gap between theory and practice. Implementing the algorithm demands attention to nuances across programming languages and hardware, as minor details can yield significant differences. The last step is experiments, which can influence the analysis again. Many experiments can be done with relatively little effort. However, they also require nontrivial planning, evaluation, archiving, and interpretation of the results. The starting point should be a falsifiable hypothesis, which can then be used for the next iteration of the cycle.
Realistic models serve as inputs during the design phase, representing abstractions of application problems. During the experimentation phase, real inputs are required. Most of these steps are closely related to applications.

3.1.7 Technical experiments

Design of Experiments [24] is a statistical tool that can be used for planning and conducting experiments. A practical methodology can be split into the planning, designing, conducting, and analyzing phases [39]. The planning phase requires a clear, objective, specific, and measurable statement of the problem. A meaningful response characteristic should be carefully chosen, there can also be multiple. Furthermore, process variables and design parameters must be selected, mostly they are set from knowledge of historical data. The next steps are classification and determining the level of process variables, the interaction between all variables must also be known.

The design phase is highly individualized, depending on many factors. Key principles to consider here include randomization, replication, and blocking [23].

In the conducting phase, the experiment is carried out and the results are evaluated. It is important to document everything during the experiment, even unusual occurrences.

The analysis phase involves interpreting results and drawing conclusions based on the collected data.

3.2 Methodological steps

To answer these research questions mentioned in Section 1.3, the framework of Design Science Research is used. As previously discussed, Design Science Research uses three cycles. In this thesis, a Systematic Literature Review serves as the methodology for the relevance and rigor cycle. Therefore, business needs can be derived from the literature, and knowledge from the rigor cycle can influence the thesis. Ultimately, this process contributes to the knowledge base and aligns with evolving business needs by the cycle's conclusion. Prototyping and algorithmic design are used in the design cycle. Methods during the evaluation are semi-structured expert interviews along with qualitative content analysis and technical experiments.

Use present- or past tense?

3.2.1 Systematic Literature Review

Given the time-consuming nature of a Systematic Literature Review, a partial review was chosen. During the planning phase, a review protocol was established and only one online library was defined, where strict selection criteria were applied, to limit the number of papers. In the conducting phase, qualitative data extraction was done, aiming to provide a broad overview of the knowledge base and business needs rather than addressing a specific research question. Reporting was done in a simplified way.

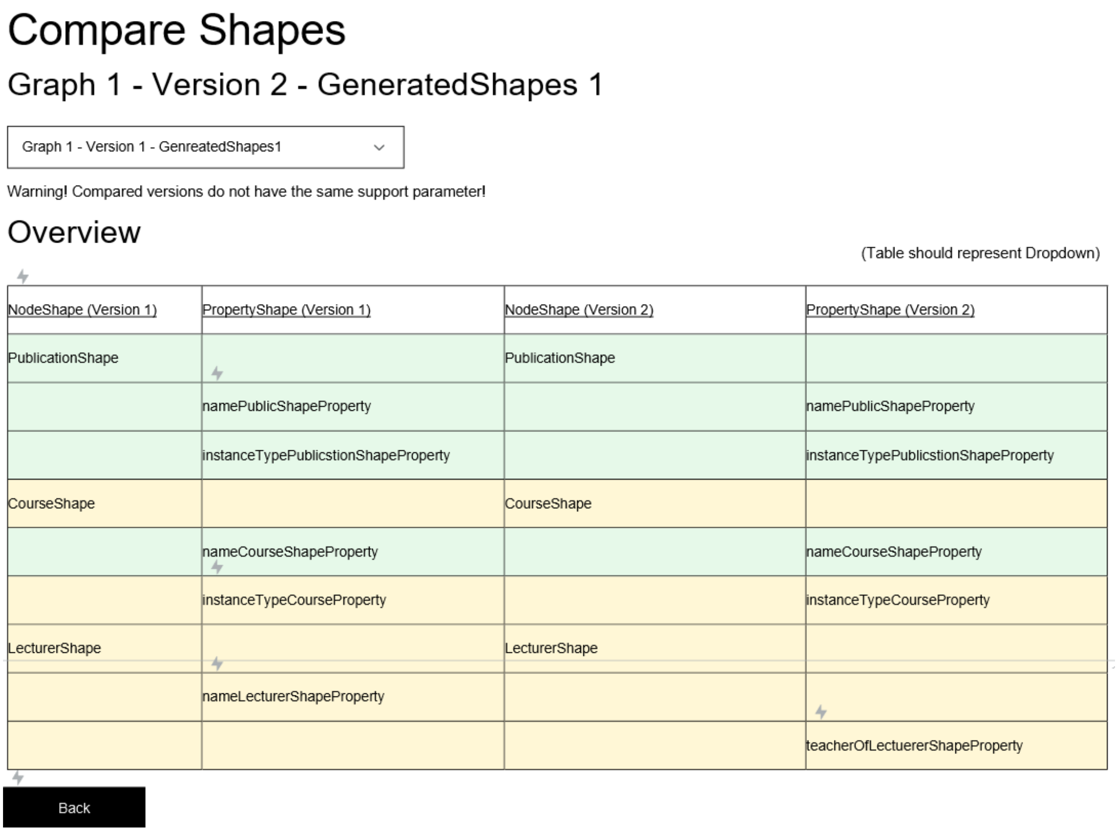
The review protocol contained the following items:

- Objectives: Establish a broad understanding of the current knowledge base in the area of evolving knowledge graphs, automatic extraction of data quality constraints from knowledge graphs, and comparing of SHACL shapes and differences between knowledge graphs

- Search Keywords: “evolving knowledge graphs”, “data quality in knowledge graphs”, “shacl extraction from knowledge graphs”, “comparing shacl shapes”, “differences between knowledge graphs versions”
- Study selection criteria: published after 2000, written in English or German, accessible with TU-Vienna Account for free
- Library: Google Scholar
- Study Exclusion Criteria: Has nothing to do with research questions or objectives
- Procedure: In the conduct phase, the first ten results for each search keyword were evaluated. The outcomes of this phase are detailed in Table A.1 in Appendix A. After reviewing the titles and applying the exclusion criteria, the abstracts of the papers were read and each paper was rated for relevance on a scale from 1 (very relevant) to 3 (not relevant). The results of this phase are presented in Table A.2 in Appendix A. Papers with a high relevance (rated 1) were further assessed by examining parts of the paper or the entire paper.
- Data Extraction: Based on their abstracts and in some cases the full paper, key sections of the content have been summarized.
- Data Synthesis: The content of the papers was descriptively synthesized.
- Data Reporting: The results of the Systematic Literature Review are presented in Section 2.2.

3.2.2 Prototyping

During the design cycle, RQ1 was answered using the prototyping method [55] within the construct phase. JustInMind [6], a design and prototyping tool for web and mobile apps, was chosen for this purpose. The free version was sufficient for this task, as the offered features were sufficient. The initial stages involved low-fidelity prototypes to determine the most suitable design that satisfies the end user’s requirements. Although only one prototype was developed, this one was already pretty similar to the final web app. Examples of this prototype are showcased in Figure 3.1 and Figure 3.2. Iteratively, the minimum viable product was developed, incorporating feedback from the evaluation phase.



Compare Shapes

Graph 1 - Version 2 - GeneratedShapes 1

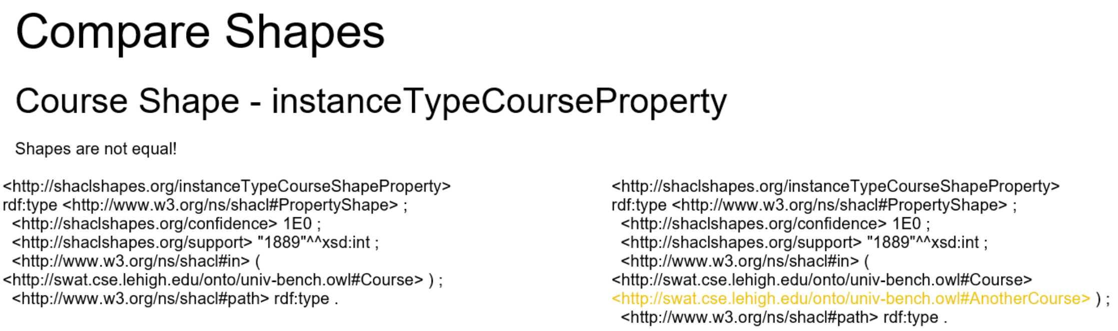
Warning! Compared versions do not have the same support parameter!

Overview

(Table should represent Dropdown)

NodeShape (Version 1)	PropertyShape (Version 1)	NodeShape (Version 2)	PropertyShape (Version 2)
PublicationShape		PublicationShape	
	namePublicShapeProperty		namePublicShapeProperty
	instanceTypePublicstionShapeProperty		instanceTypePublicstionShapeProperty
CourseShape		CourseShape	
	nameCourseShapeProperty		nameCourseShapeProperty
	instanceTypeCourseProperty		instanceTypeCourseProperty
LecturerShape		LecturerShape	
	nameLecturerShapeProperty		
			teacherOfLectuererShapeProperty

Figure 3.1: Prototype of a comparison



Compare Shapes

Course Shape - instanceTypeCourseProperty

Shapes are not equal!

```
<http://shaclshapes.org/instanceTypeCourseShapeProperty>
rdf:type <http://www.w3.org/ns/shacl#PropertyShape> ;
  <http://shaclshapes.org/confidence> 1E0 ;
  <http://shaclshapes.org/support> "1889"^^xsd:int ;
  <http://www.w3.org/ns/shacl#in> (
<http://swat.cse.lehigh.edu/onto/univ-bench.owl#Course> ) ;
  <http://www.w3.org/ns/shacl#path> rdf:type .
```

```
<http://shaclshapes.org/instanceTypeCourseShapeProperty>
rdf:type <http://www.w3.org/ns/shacl#PropertyShape> ;
  <http://shaclshapes.org/confidence> 1E0 ;
  <http://shaclshapes.org/support> "1889"^^xsd:int ;
  <http://www.w3.org/ns/shacl#in> (
<http://swat.cse.lehigh.edu/onto/univ-bench.owl#Course>
<http://swat.cse.lehigh.edu/onto/univ-bench.owl#AnotherCourse> ) ;
  <http://www.w3.org/ns/shacl#path> rdf:type .
```

Figure 3.2: Prototype of the comparing a shape in detail

3.2.3 Algorithmic Design

Contrastingly, RQ2, RQ3, and RQ4 used algorithmic design [78] in the construct phase of the design cycle. The approach entails maintaining algorithm simplicity, reusability, and utilizing libraries. Iterative development through small experiments refined the algorithms. During development, synthesized data and small knowledge graphs were utilized. As realistic input, graph snapshots from the BEAR datasets were used to test the applications. The BEAR, **BE**nchmark of RDF **AR**chives, datasets [3] are specifically designed for testing of evolving semantic web data, contain three real-world datasets and will be described in more detail in Section 7.0.2.

3.2.4 Evaluation

For the evaluation part of the design cycle for RQ1, semi-structured expert interviews [21] with students, who have already participated in Introduction To Semantic Systems were conducted. The analysis of interview content employed the inductive approach of qualitative content analysis [64]. The important part here was to measure the usability of the user interface in comparison to finding differences across the shapes without the tool. The detailed procedure and the results are provided in Chapter 7.

For the evaluation phase in the design cycle for RQ2, RQ3, and RQ4, technical experiments [24] were conducted, aligning with common practices in DSR projects for instantiations [39]. The experiments ran on a virtual machine since the data size of the test data was huge. After it was ensured, that the algorithm worked correctly, speed became the measurable metric.

The baseline here is the time it takes to generate S_1 by using G_1, plus the generation time of S_2 by using G_2. Additionally, the execution time of a script that compares S_1 and S_2 by their shape names was added to the baseline. However, more details are provided in Chapter 7.

CHAPTER 4

Requirements and Functionality

This chapter provides an overview of the technical requirements and the functionalities needed to address each research question. Research Question 1 is addressed by a tool called ShapeComparator, while Research Question 4 is implemented by a tool named SPARQL-ShapeValidator.

4.1 ShapeComparator

As the initial task was to enable a user to compare shape graphs in a user-friendly way, a tool with a visual interface was implemented. First, it allows users to upload different versions of a graph, for example, G_1 and G_2. Next, the tool facilitates the extraction of shape graphs, like S_1 and S_2, using QSE. Finally, users can compare these generated shapes with each other.
A brief overview of the overall architecture for this task is shown in Figure 4.1. The diagram illustrates the involvement of a database that stores graph versions provided by the user and the generated shapes. Additionally, the application uses the QSE algorithm to generate SHACL shapes. The application is accessible via a web interface in a browser. The user workflow is provided in Figure 4.2. While this diagram illustrates the default workflow, users have the flexibility to return to previous steps at any time. The subsequent sections describe the functionalities of these steps in more detail.

Storage of different graph versions

To make the comparison of graph versions easier, ShapeComparator offers the possibility to save graphs and their respective versions. Each graph can have multiple versions associated with it. For instance, the graph "Bear-B", which will be described in more detail in Section 7.0.2, has versions ranging from 1 to 89.
In compliance with QSE, ShapeComparator requires that only graphs or versions in

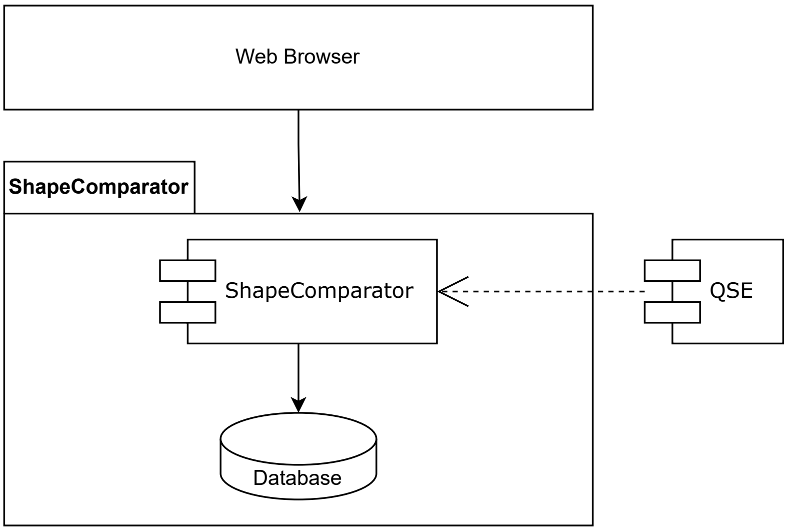


Figure 4.1: ShapeComparator - Architecture

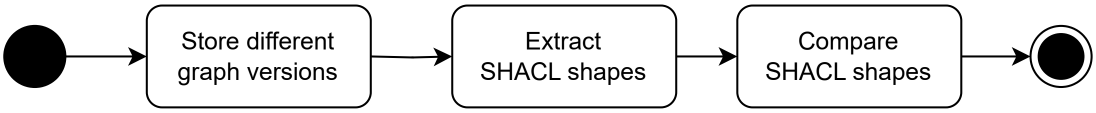


Figure 4.2: Default user workflow

N-Triples format can be uploaded. Users have the option to either upload a file or use a pre-configured graph file.
The capability to store graphs and versions improves the user experience significantly. Without this feature, users would have to upload files repeatedly for comparisons which would be very inconvenient and consume additional time and effort. Shactor [75] does not include this saving feature. This is specifically convenient as there may arise scenarios where running QSE multiple times on the same graph file is necessary, for instance, with varying support or confidence parameters.

Extract SHACL shapes

When QSE should be run, the user chooses a graph and the desired version. Similar to Shactor, all classes C mentioned in the graph G and how often they occur in G are listed. The user can choose the classes on which QSE should be run. Additionally, the user can either extract default shapes (all shapes) or set the support and confidence parameters for pruning. Notably, a node shape does not have a confidence parameter. The program also stores the extracted SHACL shapes in the database, allowing them to be selected for comparison later.

Compare SHACL shapes

Finally, the extracted shapes S are available for comparison. Users can choose one or multiple QSE runs along with their extracted shapes for comparison. In general, there is the possibility to compare extracted shapes with all combinations of different parameters such as support, confidence, or chosen classes. However, there will be warnings, if extracted shapes with different support or confidence parameters are compared or if extracted shapes with different classes are selected. The order in which the objects are compared can also be chosen. For instance, if there are two extracted shape graphs from two QSE runs (S_1 and S_2), then S_1 can be compared with S_2, or vice versa, S_2 with S_1. For an overview of the comparison, node and property shapes are organized in a tree-view format. Objects from different QSE runs are displayed side by side, with shapes mapped by their name across different QSE runs. Non-matching shapes are highlighted so that the user can easily find differences. Users can search for specific shapes per name or filter by identical node shapes, identical property shapes, or different shapes in general.

There is a detailed view for every node or property shape in the comparison overview, where the SHACL shapes of the QSE runs are displayed as text. It was decided to display SHACL shapes as turtle text snippets, as this format is both readable and familiar to users as the intended users of this tool have experience with SHACL in any way.

In this detailed view, discrepancies are also visually highlighted in red or green, indicating additions or removals in the text. If multiple shapes were selected for comparison, there is the possibility to select the desired shapes, as only two shapes can be directly compared at a time. Additionally, to the text comparison, all shapes are listed along with their corresponding support and confidence values, as these were omitted from the original SHACL shape.

When a user selects a row in the overview where a shape was deleted, a prompt will appear, explaining why this shape was excluded. This information is retrieved from the default shapes which are also preserved during shape extraction.

4.2 SPARQL-ShapeValidator

As outlined in RQ4 in Section 1.3, the primary goal of the described algorithm is to enhance the speed of QSE for evolving knowledge graphs. In this context, QSE may have already been executed on a previous version of a graph G_1, resulting in the SHACL shapes S_1, and the goal is to obtain S_2 for a subsequent version G_2 of a graph. To achieve this, the query-based option of QSE is used.

Given that changes in the schema between graph versions are often minimal, it is more efficient to validate the existing shapes S_1 rather than to reprocess all triples in the graph G_2, as QSE does. The reason for this is that the shapes graph is generally much smaller than the data graph. For example, the data graph might contain thousands of people, each with a name, while the shapes graph would have only one shape to describe this relationship. The validated shapes produced by the SPARQL-ShapeValidator are called S_{2*} in this context whereas the shapes generated if QSE would be run on G_2 are called

S_2. The main advantage of this approach is that it is faster than running QSE on G_2. However, the drawback of this solution is that the shape recognition is incomplete because new shapes that were not present in S_1 cannot be identified which means that $S_{2*} \neq S_2$. Therefore the goal of this method is not to generate S_2, it rather aims to balance runtime efficiency with the accuracy of the results. The SPARQL-ShapeValidator is therefore mainly useful for large graphs where shapes extraction performance is particularly important and where the detection of new shapes is secondary.
The software architecture for this solution is shown in Figure 4.3. The SPARQL-ShapeValidator needs the query-based version of QSE to obtain S_1 and also it needs access to G_2 which must be stored on a triplestore like GraphDB.

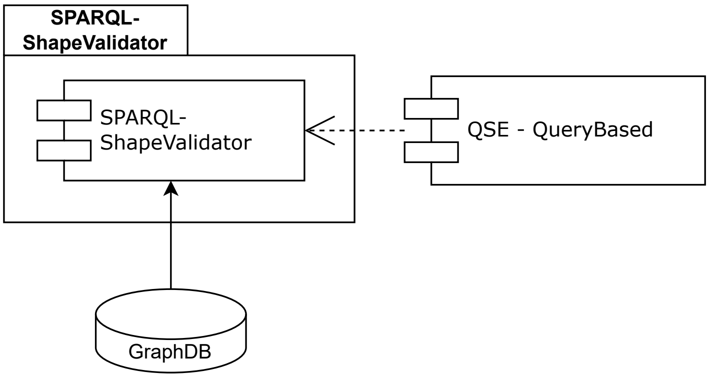


Figure 4.3: SPARQL-ShapeValidator - Architecture

The main steps in this algorithm are illustrated in Figure 4.4. The process begins with the Java objects from S_1 as input. Each shape in S_1 is checked for its relevance in G_2. Based on this verification, the algorithm modifies the turtle file containing all SHACL shapes S_1, removing any parts that are no longer valid. The resulting output is a new turtle file with the shapes S_{2*}.

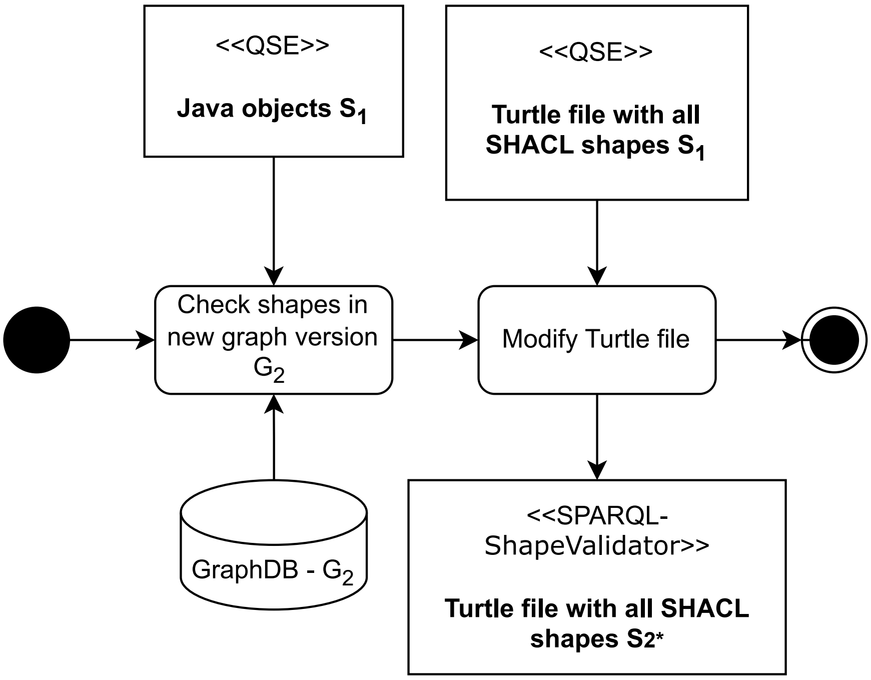

<<QSE>>
Java objects S_1
<<QSE>>
Turtle file with all SHACL shapes S_1
Check shapes in new graph version G_2
Modify Turtle file
GraphDB - G_2
<<SPARQL-ShapeValidator>>
Turtle file with all SHACL shapes S2*

Figure 4.4: SPARQL-ShapeValidator - Activity Diagram

CHAPTER 5

Research Contribution

This chapter lists key ideas that were used for resolving the research questions. A key part describes the general approach to the comparison of SHACL shapes which is used in all research questions. Furthermore, the algorithms used in RQ4 are detailed.

5.1 Comparing SHACL shapes

A key task for this thesis is to enable the comparison of generated SHACL shapes. The information sources for this comparison include the SHACL file generated by QSE and the internal Java objects after the QSE execution. Both of these information sources contain all node and property shapes. QSE structures SHACL files always in the same way using the TurtlePrettyFormatter [82]. This tool offers various configurations for structuring turtle files and the resulting SHACL shapes are mostly consistent, with a few exceptions.

However, comparing entire SHACL shape files directly is impractical, and also the Java objects are not easily comparable or interpretable by users.

A primary challenge is to present a single SHACL shape in a readable format that is user-friendly and also easily comparable by algorithms. To achieve this goal, it was decided to present the SHACL shapes as turtle text snippets with some modifications (e.g. removing support and confidence values). This format is both readable and familiar to users, and easy to compare for algorithms.

Several approaches were considered on how to use the information provided by QSE to get these SHACL shape text snippets. In the final version, a combination of the two artifacts was used, the list of objects was used internally and the SHACL file was used for text generation.

A crucial decision in this context is the assumption that the most effective method for identifying a shape is by using its name. After some experiments, it became clear that the fastest way to get a SHACL shape as text by name is to use regular expressions (regex)

on the SHACL file and obtain the text of a shape in this way. With regular expressions, texts can be efficiently searched by using keywords, wildcards, and other rules.
An alternative attempt involved reading the SHACL file into an RDF4J or Jena model. RDF4J and Jena are Java libraries that allow the creation of so-called models from a turtle file, which can then be managed as a graph in Java. After the creation, the model can be filtered using SPARQL queries or provided methods to retrieve only the relevant text for each shape. This was achieved by searching for triples where the subject equals the name of the shape. However, more complex shapes can include SHACL-Or items or SHACL-In lists which are internally stored with blank nodes. To retrieve the information stored in these blank nodes, the triples had to be loaded recursively from the whole shape graph. This approach, in general, proved to be inefficient, therefore the final solution was to use regex to extract the shape segments directly from the SHACL file. Although this approach has several disadvantages, such as potential inconsistencies, it is notably faster.
After extracting the original text for a shape, it still needs to be made comparable to other shapes. First, support and confidence values must be removed since these values mostly differ between different graph versions. This was done by converting the text segment into a Jena graph and then filtering the support and confidence statements. Additionally, issues arose with SHACL-In and SHACL-Or items, as these are not considered by the TurtlePrettyFormatter. Some adjustments to the text were needed to reorder these items alphabetically. This task was solved using regex and Java string operations. Finally, the shape is now comparable to shapes with an identical name from other graph versions.

5.2 Validating shapes with SPARQL

This section outlines the logic used in RQ4, the SPARQL-ShapeValidator.
The task is divided into two main steps. In the first step, the generated SHACL shapes in the form of Java objects from the initial QSE run are used as input. The algorithm loops through these objects and checks the constraints in the second version of the graph. The output from this step consists of the same objects as those provided by QSE in the first run but with updated support and confidence values. The pseudo-code for this algorithm is provided in Algorithm 5.1.
In the second step, which is illustrated in Algorithm 5.2, the adapted objects from the first step are iterated through. During this process, any objects (including node shapes, property shapes, and constraints in property shapes) that do not meet the thresholds for support and confidence are deleted. If these thresholds are not specified, then all shapes with a support of zero are removed. The deletion, just as the selection of shapes which was described in Section 5.1, works per name and is mostly completed with regex. The input for this step is the SHACL shapes file from the first QSE run, and the output is also saved as a turtle file, with the updated SHACL shapes.
There are specific edge cases to consider for the deletion of shapes. One such case occurs when there is a SHACL-Or list in the first version of a shape but there is only a single constraint in the second version. In this scenario, the list must be removed. For this task, the shape is converted to a Jena model, and the inner constraint is

unwrapped via SPARQL. After that, the Jena model is converted back to a string using the TurtlePrettyFormatter library. Another edge case arises when the support values of a node shape and a property shape are no longer equal. In this case, it is necessary to remove the SHACL-minCount triple if it was present in the initial version of the shape. The result of this second step is a Turtle file containing all validated SHACL shapes. This file serves as the basis for comparison during the evaluation phase.

Algorithm 5.1: Validate existing shapes with SPARQL

```
Input  : $S_{1j}$: Java objects from $S_1$, $G_2$: second version of graph
Output : $S_{2*j}$: Adapted Java objects for $G_2$
$S_{2*j} \leftarrow S_{1j}$
Run SPARQL query on $G_2$ which returns all counts for all classes mentioned in
  $S_{1j}$
Update the support values for the node shapes in $S_{2*j}$
foreach node shape $\langle s, \tau_n, \Phi_n \rangle \in S_{2*j}$ do
    foreach property shape $\phi_n : \langle \tau_p, T_p, C_p \rangle \in \Phi_n$ do
        Run SPARQL query on $G_2$ with given nodeKind and $\tau_p$ for all $T_p$ which
          returns the count for the specific constraint and update the support and
          confidence values for $\phi_n$ in $S_{2*j}$
    end
end
return $S_{2*j}$
```

5.3 Contributions to the QSE algorithm

Not finished yet, only notes

Pull Request to shactor branch

- Shapes get duplicated after every run in QueryBased: -> deleteOutputDir in runParser (line 74)
- Bug: Query for literals property does not get replaced in QbParser -> incomplete shapes
- Shactor: SAIL repository is locked when executed with all classes and then only with one class e.g. ScriptWriter -> Shactor needs to be restarted

Algorithm 5.2: Delete unvalidated SHACL shapes from file

```
Input  : $S_{2*j}$: Adapted Java objects from $S_1$ for $G_2$, $S_{1f}$: Turtle file with all
         SHACL shapes $S_1$, $\omega$: support-threshold, $\epsilon$: confidence-threshold
Output : Turtle file with updated SHACL shapes $S_{2*f}$
$S_{2*f} \leftarrow S_{1f}$
foreach node shape $\langle s, \tau_n, \Phi_n \rangle \in S_{2*j}$ do
    if support of node shape $\leq \omega$ then
        delete node shape from $S_{2*f}$ with regex based on $s$
        delete all associated property shapes $\Phi_n$ from $S_{2*f}$ with regex
    else
        foreach property shape $\phi_n : \langle \tau_p, T_p, C_p \rangle \in \Phi_n$ do
            if support of $\phi_n \leq \omega$ or confidence of $\phi_n \leq \epsilon$ then
                delete $\phi_n$ from $S_{2*f}$ with regex
                delete triple in node shape that makes $\phi_n$ part of the node shape
            else
                if $|T_p| > 1$ then
                    foreach constraint in $\phi_n$ do
                        if support of constraint $\leq \omega$ or confidence of constraint $\leq \epsilon$
                         then
                            Delete constraint from the property shape from $S_{2*f}$
                             with regex
                        end
                    end
                    if only one constraint is left after deletion then
                        Remove the SHACL-Or list from the property shape and
                         connect the constraint directly to the property shape
                    end
                end
                if support of node shape $\neq$ support of $\phi_n$ then
                    Remove the SHACL-MinCount triple from property shape from
                     $S_{2*f}$ with regex
                end
            end
        end
    end
end
return $S_{2*f}$
```

CHAPTER 6

Implementation

This chapter describes the implementation of all research questions, using the requirements defined in Chapter 4, and the algorithms and considerations in Chapter 5. An example with a small knowledge graph will be used to explain and describe the use cases in more detail. Furthermore, screenshots of the ShapeComparator will be provided and the technical details of the implementation are listed.

Put code on central dbai GitHub

6.1 Technical Details

To implement this project Java (Version 17) was used. There were two projects developed, ShapeComparator as the user interface and a command-line Java project, that contains the algorithms from RQ3 and RQ4. This differentiation was made due to the different kinds of applications, since ShapeComparator is used as a web application with an internal database and the algorithms from RQ3 and RQ4 only require a console application.

6.2 Version of QSE

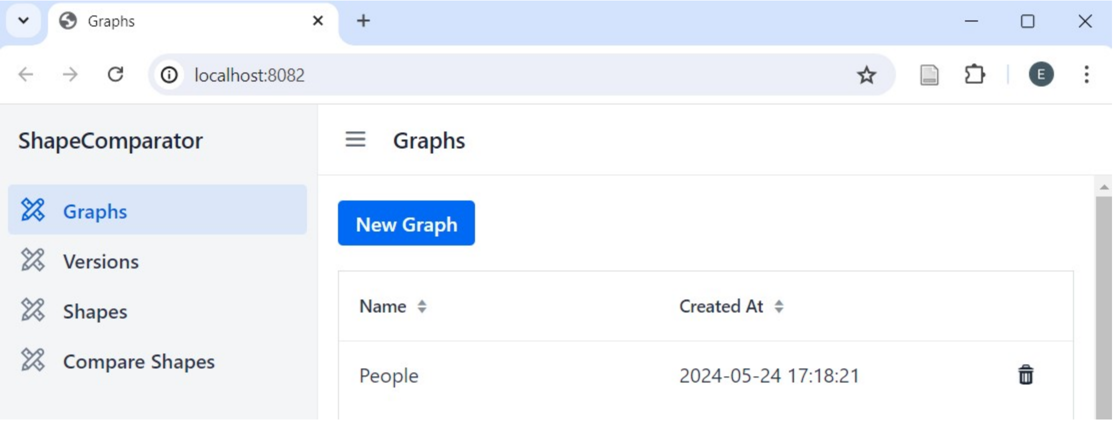
There are different versions of QSE available, namely the "main" branch, which is used for applying QSE on a single graph in the console efficiently, and the "shactor version". Shactor utilizes a variant of the QSE algorithm that differs from the main branch. This version generates Java objects for each node shape, property shape, and similar elements, in addition to producing SHACL shapes as a turtle file. These objects are needed to display the SHACL shapes in the graphical interface of Shactor. Consequently, this thesis employs the "Shactor"-version of QSE [5] as well. It is used for the graphical part, the ShapeComparator, but also for the algorithms since the performance impact for the Java shape object generation in QSE is minimal. Of course, the modifications described in Section 5.3 are included in this QSE version, which is used by all parts of this project.

6.3 ShapeComparator

To answer RQ1, a web application has been created that allows users to compare generated SHACL shapes from QSE effectively. This comparison is available across multiple versions of a graph. The goal was to maximize the usability, enabling the user to see differences as easily as possible.
In reference to Shactor [75], the application was developed with Vaadin (Version 24.2.4) [16], and Spring. Vaadin is a full-stack framework that allows front-end development with Java. For this project, the free version of Vaadin was selected as the features provided were sufficient. Another tool involved was the JavaScript library jsdiff [34] to compare the text segments of the SHACL shapes visually. This tool is used to highlight the differences automatically in the details view of ShapeComparator.

6.3.1 Illustration with Screenshots

An extended version of the example from Section 2.1.2 was chosen to showcase the functionalities of ShapeComparator. The complete RDF graphs are available in Appendix B. ShapeComparator is designed for laptop screens, a screenshot is illustrated in Figure 6.1 The initial screen of the ShapeComparator application shows a list of all graphs saved in the application. Each graph can have multiple versions associated with it. In this demonstration, the "People" knowledge graph has three versions. These versions of the knowledge graphs are fully printed in Listings B.1, B.2 and B.3.



Name	Created At
People	2024-05-24 17:18:21

Figure 6.1: Full screenshot of the ShapeComparator application

In the first version of the knowledge graph, there are three people, Alice, Bob, and Jenny, who have a name attribute and an attribute that indicates that they know each other. A snippet of this knowledge graph is shown in Listing 6.1.

```
<http://example.org/alice>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/alice>
```

```
    <http://xmlns.com/foaf/0.1/name> "Alice" .
<http://example.org/alice>
    <http://xmlns.com/foaf/0.1/knows>
    <http://example.org/bob> .
```

Listing 6.1: Snippet of People Knowledge Graph (Version 1)

In the second version of this knowledge graph three cats, each with an associated color, are added. A snippet of this knowledge graph can be seen in Listing 6.2.

```
<http://example.org/orangeCat>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Cat> .
<http://example.org/orangeCat>
    <http://example.org/color> "orange" .
```

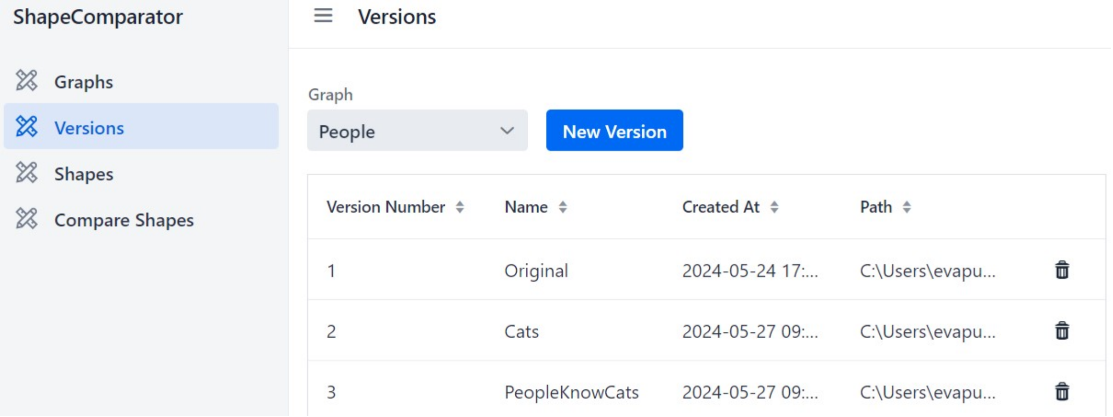
Listing 6.2: Snippet of People Knowledge Graph (Version 2)

In the third version, the color attribute is removed from two of the three cats, and instead of knowing each other, Alice, Bob, and Jenny each now know one cat, as shown in Listing 6.3.

```
<http://example.org/alice>
    <http://xmlns.com/foaf/0.1/knows>
    <http://example.org/orangeCat> .
```

Listing 6.3: Snippet of People Knowledge Graph (Version 3)

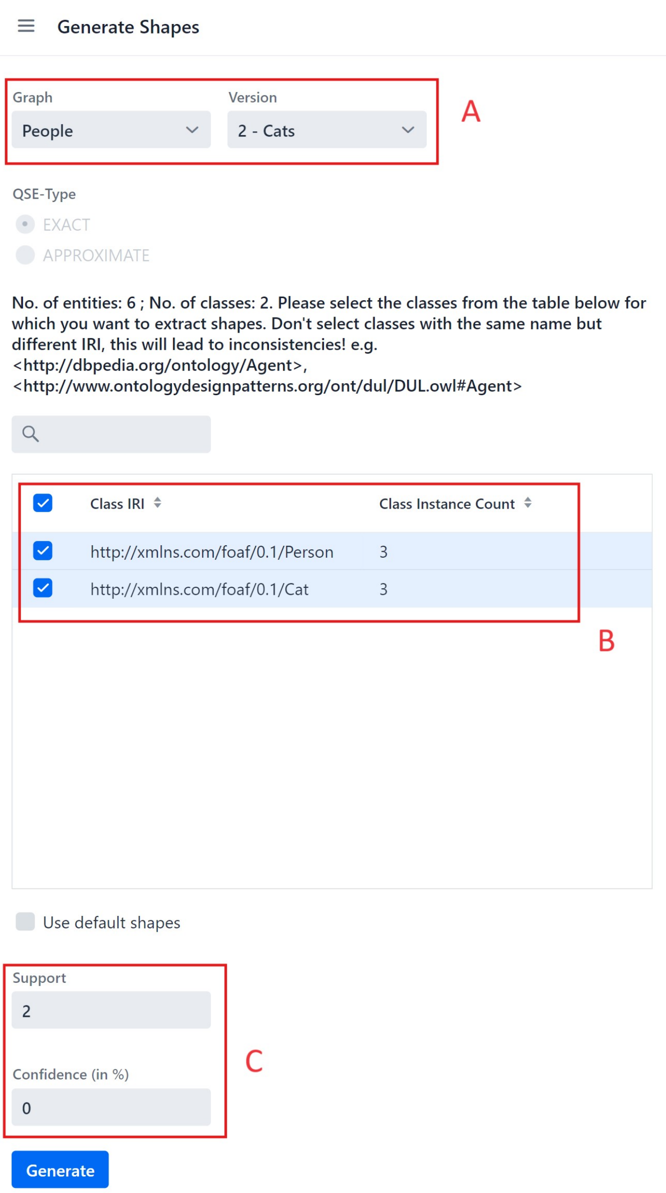
The user interface for the overview of the versions of a graph is illustrated in Figure 6.2.



Version Number	Name	Created At	Path
1	Original	2024-05-24 17:...	C:\Users\evapu...
2	Cats	2024-05-27 09:...	C:\Users\evapu...
3	PeopleKnowCats	2024-05-27 09:...	C:\Users\evapu...

Figure 6.2: Overview of all versions of a graph

After the knowledge graph versions are saved, to avoid uploading them repeatedly, the user can continue to extract SHACL shapes. Figure 6.3 illustrates this process for the second version of the graph (Box A). QSE automatically lists all classes along with their instance counts for selection (Box B). As previously described, this version includes three people and three cats. The support parameter is set to two and the confidence parameter is set to zero (Box C). This means that all triples, where two or fewer triples conform to that class are not considered for the resulting shapes. The confidence parameter is set to zero percent, ensuring no triples are dropped because of too low confidence values. The detailed definitions of the support and confidence parameters are provided in Section 2.2.3. QSE was also executed on the other two versions of the knowledge graph.



Generate Shapes

Graph: People

Version: 2 - Cats

QSE-Type

● EXACT

○ APPROXIMATE

No. of entities: 6 ; No. of classes: 2. Please select the classes from the table below for which you want to extract shapes. Don't select classes with the same name but different IRI, this will lead to inconsistencies! e.g. <http://dbpedia.org/ontology/Agent>, <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Agent>

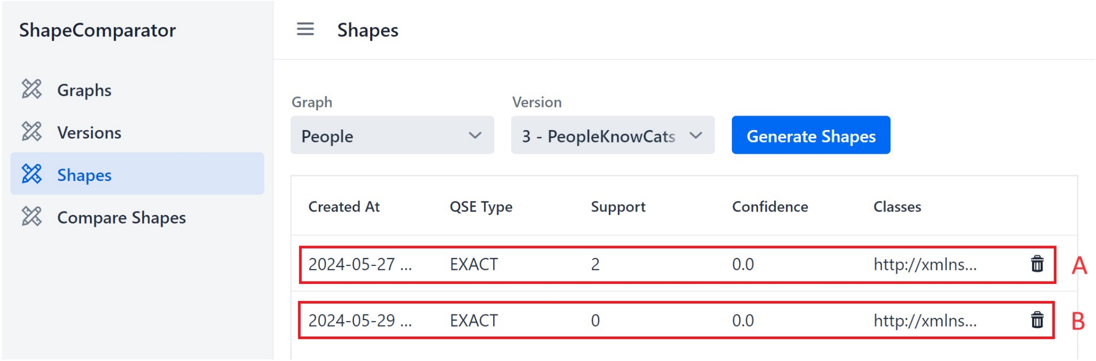
☑	Class IRI	Class Instance Count
☑	http://xmlns.com/foaf/0.1/Person	3
☑	http://xmlns.com/foaf/0.1/Cat	3

☐ Use default shapes

Support: 2

Confidence (in %): 0

Figure 6.3: SHACL shapes can be extracted for a version of a graph

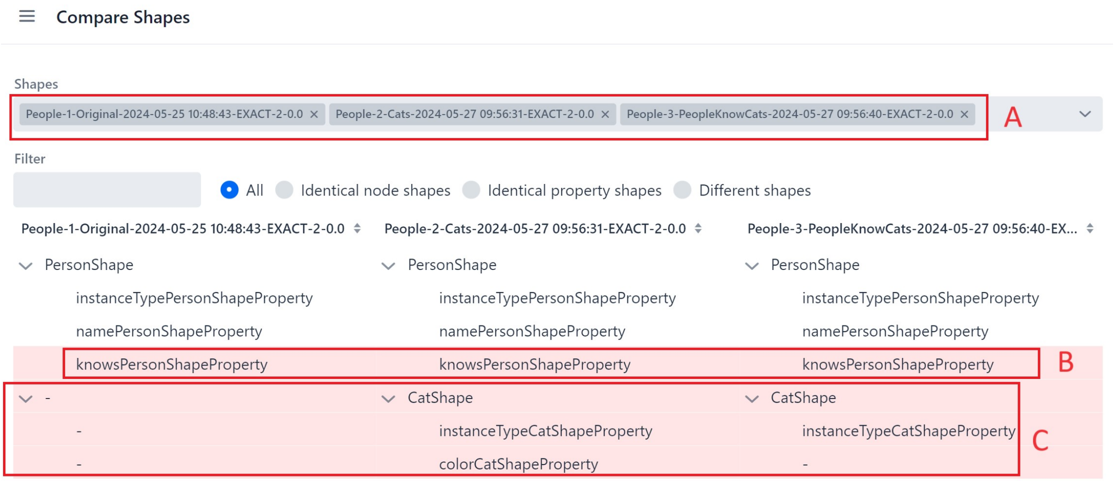


Graph: People

Version: 3 - PeopleKnowCats

Created At	QSE Type	Support	Confidence	Classes
2024-05-27 ...	EXACT	2	0.0	http://xmlns...
2024-05-29 ...	EXACT	0	0.0	http://xmlns...

Figure 6.4: Overview of all extracted shapes for a version of a graph



Shapes: People-1-Original-2024-05-25 10:48:43-EXACT-2-0.0 × People-2-Cats-2024-05-27 09:56:31-EXACT-2-0.0 × People-3-PeopleKnowCats-2024-05-27 09:56:40-EXACT-2-0.0 ×

Filter: ● All ○ Identical node shapes ○ Identical property shapes ○ Different shapes

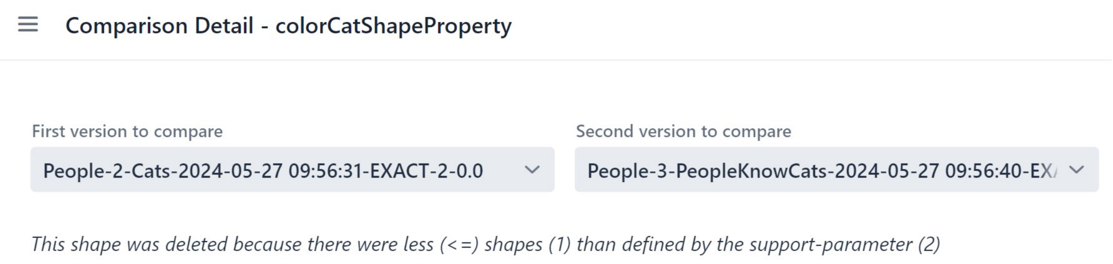
People-1-Original-2024-05-25 10:48:43-EXACT-2-0.0	People-2-Cats-2024-05-27 09:56:31-EXACT-2-0.0	People-3-PeopleKnowCats-2024-05-27 09:56:40-EX...
PersonShape	PersonShape	PersonShape
instanceTypePersonShapeProperty	instanceTypePersonShapeProperty	instanceTypePersonShapeProperty
namePersonShapeProperty	namePersonShapeProperty	namePersonShapeProperty
knowsPersonShapeProperty	knowsPersonShapeProperty	knowsPersonShapeProperty
-	CatShape	CatShape
-	instanceTypeCatShapeProperty	instanceTypeCatShapeProperty
-	colorCatShapeProperty	-

Figure 6.5: Comparison between extracted shapes

The overview of all extracted shapes for the third version is illustrated in Figure 6.4. In this example, the previously generated shapes (Box A) and the default shapes are extracted, as indicated by the support and confidence parameter values of zero (Box B). However, this extraction is not relevant to the subsequent example; it simply demonstrates that the program supports multiple shape extractions for a single version.

Finally, the extracted shapes are available for comparison in Figure 6.5. At the top of the page, the three previously generated QSE runs along with their extracted shapes are chosen (Box A). The overview indicates changes for the "knows" property in the class Person, as this property points to cats in the third version of the graph (Box B). Additionally, there are changes for the cat class (Box C).

Clicking on the row "knowsPersonShapeProperty" in the comparison table (Figure 6.7) redirects to a more detailed comparison view, where the discrepancies are visually highlighted in red and green, indicating additions or removals in the text. "Cat" is written in green, as the targeted class is now a "Cat" and not "Person" anymore, which is written



Comparison Detail - colorCatShapeProperty

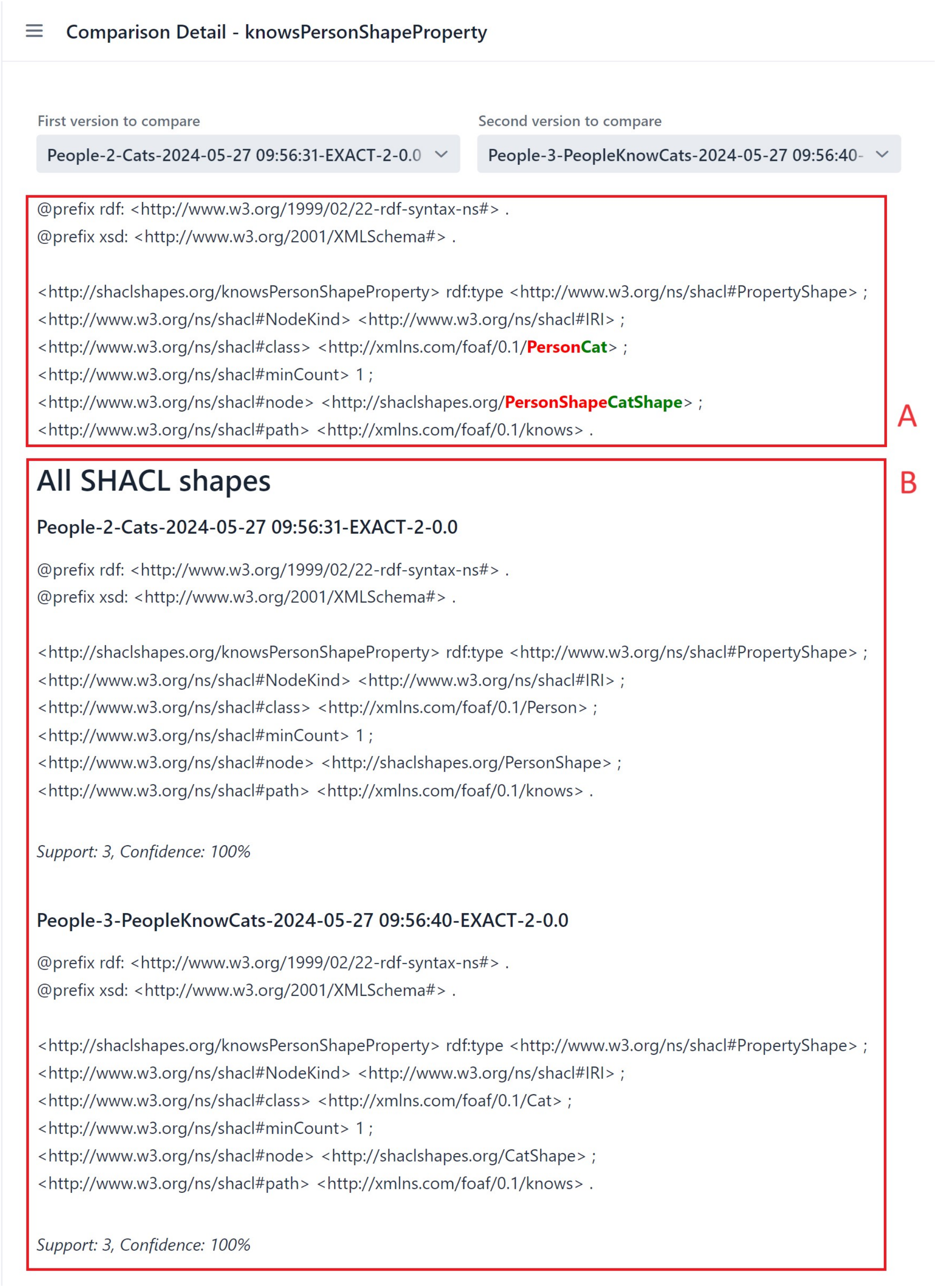
First version to compare: People-2-Cats-2024-05-27 09:56:31-EXACT-2-0.0

Second version to compare: People-3-PeopleKnowCats-2024-05-27 09:56:40-EX/

This shape was deleted because there were less (<=) shapes (1) than defined by the support-parameter (2)

Figure 6.6: Detailed comparison view in case a shape was deleted

in red (Box A). Below the comparison, all SHACL shapes are listed along with their corresponding support and confidence values (Box B). In this example, the text for the first version was removed, since nothing has changed between the first and the second version for this property. To showcase the removal of a shape due to low support or confidence values, the detailed comparison of the "colorCatShapeProperty" is chosen. In this example, the SHACL shape no longer exists in the third version of the graph because the "color" attributes of the black and grey cats were deleted. As a result, the support for this property is only one, which is below the defined threshold of two when the shapes were created. This explanation is displayed in Figure 6.6.



Comparison Detail - knowsPersonShapeProperty

First version to compare: People-2-Cats-2024-05-27 09:56:31-EXACT-2-0.0

Second version to compare: People-3-PeopleKnowCats-2024-05-27 09:56:40-

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://shaclshapes.org/knowsPersonShapeProperty> rdf:type <http://www.w3.org/ns/shacl#PropertyShape> ;
<http://www.w3.org/ns/shacl#NodeKind> <http://www.w3.org/ns/shacl#IRI> ;
<http://www.w3.org/ns/shacl#class> <http://xmlns.com/foaf/0.1/PersonCat> ;
<http://www.w3.org/ns/shacl#minCount> 1 ;
<http://www.w3.org/ns/shacl#node> <http://shaclshapes.org/PersonShapeCatShape> ;
<http://www.w3.org/ns/shacl#path> <http://xmlns.com/foaf/0.1/knows> .
```

All SHACL shapes

People-2-Cats-2024-05-27 09:56:31-EXACT-2-0.0

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://shaclshapes.org/knowsPersonShapeProperty> rdf:type <http://www.w3.org/ns/shacl#PropertyShape> ;
<http://www.w3.org/ns/shacl#NodeKind> <http://www.w3.org/ns/shacl#IRI> ;
<http://www.w3.org/ns/shacl#class> <http://xmlns.com/foaf/0.1/Person> ;
<http://www.w3.org/ns/shacl#minCount> 1 ;
<http://www.w3.org/ns/shacl#node> <http://shaclshapes.org/PersonShape> ;
<http://www.w3.org/ns/shacl#path> <http://xmlns.com/foaf/0.1/knows> .
```

Support: 3, Confidence: 100%

People-3-PeopleKnowCats-2024-05-27 09:56:40-EXACT-2-0.0

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://shaclshapes.org/knowsPersonShapeProperty> rdf:type <http://www.w3.org/ns/shacl#PropertyShape> ;
<http://www.w3.org/ns/shacl#NodeKind> <http://www.w3.org/ns/shacl#IRI> ;
<http://www.w3.org/ns/shacl#class> <http://xmlns.com/foaf/0.1/Cat> ;
<http://www.w3.org/ns/shacl#minCount> 1 ;
<http://www.w3.org/ns/shacl#node> <http://shaclshapes.org/CatShape> ;
<http://www.w3.org/ns/shacl#path> <http://xmlns.com/foaf/0.1/knows> .
```

Support: 3, Confidence: 100%

Figure 6.7: Detailed comparison view

6.3.2 Database model

Figure 6.8 shows the database tables which were auto-created by Spring. A graph can have multiple versions and those can have different extracted shapes. For each QSE run (i.e., extracted shapes), classes were defined during the extraction process (specifying which classes of the graph QSE should be run on). Every extracted-shapes object contains multiple node shapes, with default shapes also being stored here. If the user chooses to extract only the default shapes, then no data is stored under default shapes. Each node shape has multiple property shapes. These classes were originally copied from QSE, however, they were duplicated to accommodate customization and other modifications. Every property shape can have ShaclOrList items, as it was previously modeled in QSE. As a database, a local H2 instance is used as it is lightweight and easy to integrate with the Spring framework.

6.3.3 Performance enhancements

After completing the initial draft of the ShapeComparator, it was tested with very small datasets to uncover potential edge cases. However, as testing progressed with larger datasets, it became clear that certain features needed to be adjusted to enhance the speed of the web application.

Originally, it was planned to store the extracted SHACL files directly in the database as text. However, this approach quickly led to performance issues while querying data because SHACL files can get quite large. Therefore the extracted shapes and graph versions are stored in the project directory as turtle files and the file path pointing to the files is saved in the database as a reference.

A further performance improvement involved saving the generated text for the detail view for each node and property shape, which notably accelerated the comparison process. Additionally, implementing lazy fetching for node shapes, rather than fetching them eagerly from the database, contributed to further performance improvement. Given that node and property shapes can have numerous objects, resulting in large generated texts, it is impractical to keep all this information in memory. To optimize performance, it is more efficient to retrieve the required shapes from the database on demand. Also, the objects for the comparison overview were cached, to prevent reloading all objects when the user returns from the comparison details page.

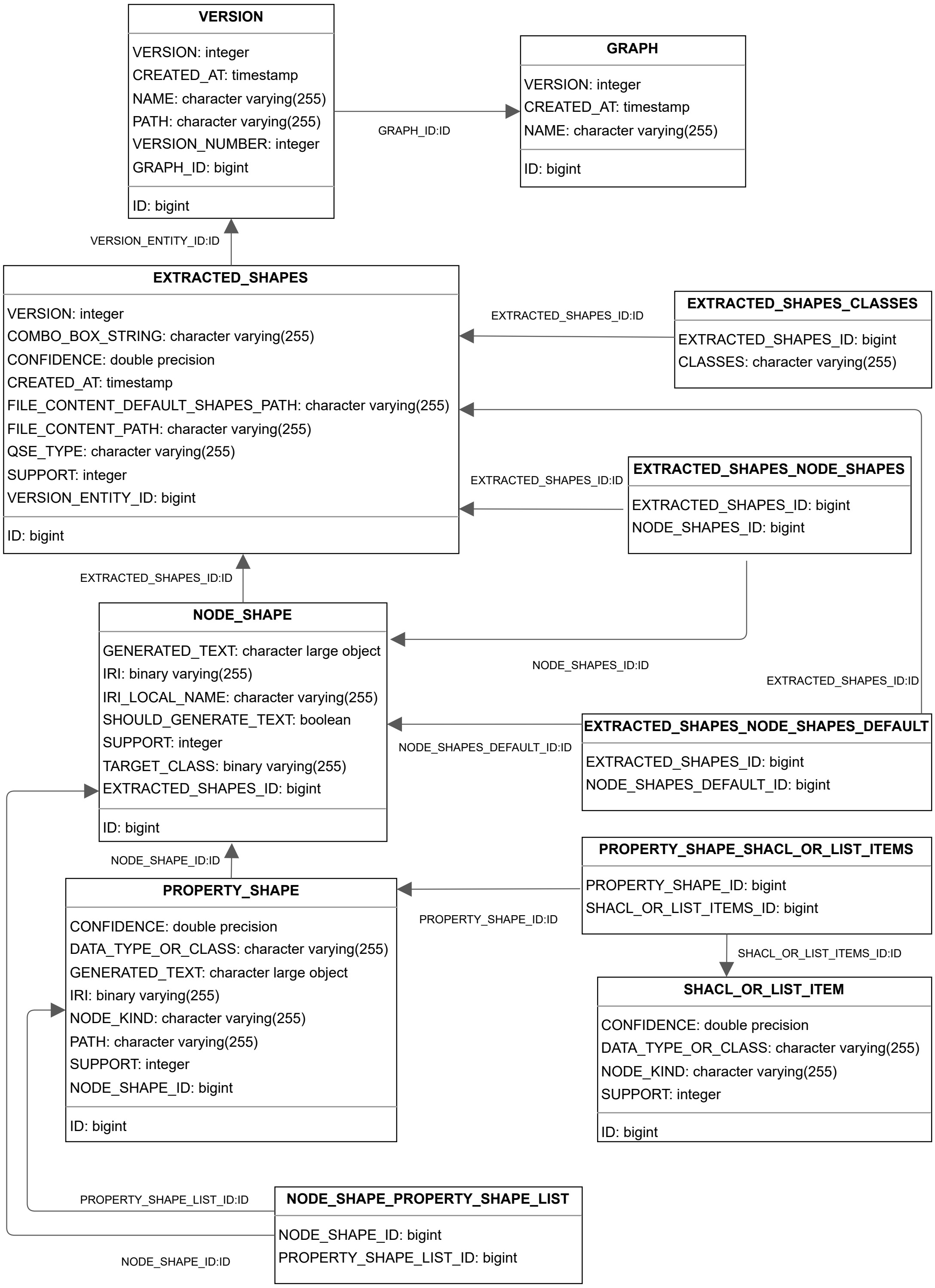

VERSION
VERSION: integer
CREATED_AT: timestamp
NAME: character varying(255)
PATH: character varying(255)
VERSION_NUMBER: integer
GRAPH_ID: bigint
ID: bigint
GRAPH_ID:ID
GRAPH
VERSION: integer
CREATED_AT: timestamp
NAME: character varying(255)
ID: bigint
VERSION_ENTITY_ID:ID
EXTRACTED_SHAPES
VERSION: integer
COMBO_BOX_STRING: character varying(255)
CONFIDENCE: double precision
CREATED_AT: timestamp
FILE_CONTENT_DEFAULT_SHAPES_PATH: character varying(255)
FILE_CONTENT_PATH: character varying(255)
QSE_TYPE: character varying(255)
SUPPORT: integer
VERSION_ENTITY_ID: bigint
ID: bigint
EXTRACTED_SHAPES_ID:ID
EXTRACTED_SHAPES_CLASSES
EXTRACTED_SHAPES_ID: bigint
CLASSES: character varying(255)
EXTRACTED_SHAPES_ID:ID
EXTRACTED_SHAPES_NODE_SHAPES
EXTRACTED_SHAPES_ID: bigint
NODE_SHAPES_ID: bigint
EXTRACTED_SHAPES_ID:ID
NODE_SHAPE
GENERATED_TEXT: character large object
IRI: binary varying(255)
IRI_LOCAL_NAME: character varying(255)
SHOULD_GENERATE_TEXT: boolean
SUPPORT: integer
TARGET_CLASS: binary varying(255)
EXTRACTED_SHAPES_ID: bigint
ID: bigint
NODE_SHAPES_ID:ID
EXTRACTED_SHAPES_ID:ID
NODE_SHAPES_DEFAULT_ID:ID
EXTRACTED_SHAPES_NODE_SHAPES_DEFAULT
EXTRACTED_SHAPES_ID: bigint
NODE_SHAPES_DEFAULT_ID: bigint
NODE_SHAPE_ID:ID
PROPERTY_SHAPE
CONFIDENCE: double precision
DATA_TYPE_OR_CLASS: character varying(255)
GENERATED_TEXT: character large object
IRI: binary varying(255)
NODE_KIND: character varying(255)
PATH: character varying(255)
SUPPORT: integer
NODE_SHAPE_ID: bigint
ID: bigint
PROPERTY_SHAPE_ID:ID
PROPERTY_SHAPE_SHACL_OR_LIST_ITEMS
PROPERTY_SHAPE_ID: bigint
SHACL_OR_LIST_ITEMS_ID: bigint
SHACL_OR_LIST_ITEMS_ID:ID
SHACL_OR_LIST_ITEM
CONFIDENCE: double precision
DATA_TYPE_OR_CLASS: character varying(255)
NODE_KIND: character varying(255)
SUPPORT: integer
ID: bigint
PROPERTY_SHAPE_LIST_ID:ID
NODE_SHAPE_PROPERTY_SHAPE_LIST
NODE_SHAPE_ID: bigint
PROPERTY_SHAPE_LIST_ID: bigint
NODE_SHAPE_ID:ID

Figure 6.8: ERD of the database used for saving graphs and shapes in ShapeComparator

6.4 SPARQL-ShapeValidator

The requirements of RQ4, outlined in Section 4.2, were implemented using the algorithms detailed in Section 5.2 in Java as a command line application. The program executes QSE on the first version of the graph and then performs the validation and deletion steps. With the example provided in Subsection 6.3.1, SPARQL-ShapeValidator can be used instead of running QSE both on the first and the second version of the graph. However, in this case, where only new entities (e.g. cats and their colors) are added to the second version of the graph, the downside of this algorithm is shown, since there are no shapes deleted. Consequently, the SHACL shapes produced by the SPARQL-ShapeValidator miss the CatShape and the colorCatShapeProperty. This limitation is a known, significant drawback of this algorithm. Despite this issue, the SPARQL-ShapeValidator demonstrates a performance advantage due to the small number of shapes, making it considerably faster than running QSE twice. The experiments to prove this observation will be discussed in the evaluation in Section 7.

However, when the SPARQL-ShapeValidator is executed on the third version of the graph, using the second version as the basis, the colorCatShapeProperty is removed successfully. This shape was added in the second graph and is therefore present in the shapes provided by QSE. A support threshold of two was applied again. In this case, the knowsPersonShapeProperty is also deleted by the SPARQL-ShapeValidator because the target objects of the "knows" property in the third version are cats instead of people. Consequently, SPARQL-ShapeValidator cannot find this property anymore. The output of the SPARQL-ShapeValidator for version three would therefore look like the shape provided in Listing 6.4. This demonstrates again that the SPARQL-ShapeValidator is not a perfect fit for a small graph like this where performance impacts are insignificant but the detection of new shapes is more important.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://shaclshapes.org/CatShape> rdf:type
    <http://www.w3.org/ns/shacl#NodeShape> ;
  <http://www.w3.org/ns/shacl#targetClass>
    <http://xmlns.com/foaf/0.1/Cat> .

<http://shaclshapes.org/PersonShape> rdf:type
    <http://www.w3.org/ns/shacl#NodeShape> ;
  <http://www.w3.org/ns/shacl#property>
    <http://shaclshapes.org/namePersonShapeProperty> ;
  <http://www.w3.org/ns/shacl#targetClass>
    <http://xmlns.com/foaf/0.1/Person> .

<http://shaclshapes.org/namePersonShapeProperty> rdf:type
     <http://www.w3.org/ns/shacl#PropertyShape> ;
  <http://shaclshapes.org/confidence> 1E0 ;
  <http://shaclshapes.org/support> "3"^^xsd:int ;
  <http://www.w3.org/ns/shacl#NodeKind>
    <http://www.w3.org/ns/shacl#Literal> ;
  <http://www.w3.org/ns/shacl#datatype> xsd:string ;
  <http://www.w3.org/ns/shacl#minCount> 1 ;
  <http://www.w3.org/ns/shacl#path>
    <http://xmlns.com/foaf/0.1/name> .
```

Listing 6.4: Result of SPARQL-ShapeValidator on the People Knowledge Graph (Version 3) with the People Knowledge Graph (Version 2) as basis

CHAPTER 7

Evaluation

This chapter gives an overview of the evaluation of the research questions. The ShapeComparator was evaluated with expert interviews, while the algorithms from RQ3 and RQ4 were evaluated with technical experiments. Furthermore the test data is described in detail as well as the virtual machine which was used for the experiments.

7.0.1 System information

The application was tested on a Linux Virtual machine with Debian (Version 11), 8 CPU cores, a speed of 2200 MHz, and 128 GB of RAM.

7.0.2 Test-data

The test data utilized is sourced from BEAR [3] which was designed for testing archiving and querying of evolving semantic web data. These datasets, collectively known as the **BE**nchmark of RDF **AR**chives (BEAR) consist of three real-world datasets. The BEAR data offers valuable statistics and provides various methods for accessing the data, such as having one N-Triples file per version.
For testing, mostly the data from BEAR-B was used. While datasets from BEAR-A (snapshots from Dynamic Linked Data Observator) were too large, with one file reaching 5GB, the data from BEAR-C (Open Data portals) was too homogeneous. For BEAR-B the daily changesets were used since there are already 89 different versions available. The dataset has been compiled from DBPedia Live Changesets over three months (August to October 2015), including the 100 most volatile resources along with their updates.

7.1 ShapeComparator

expert interview

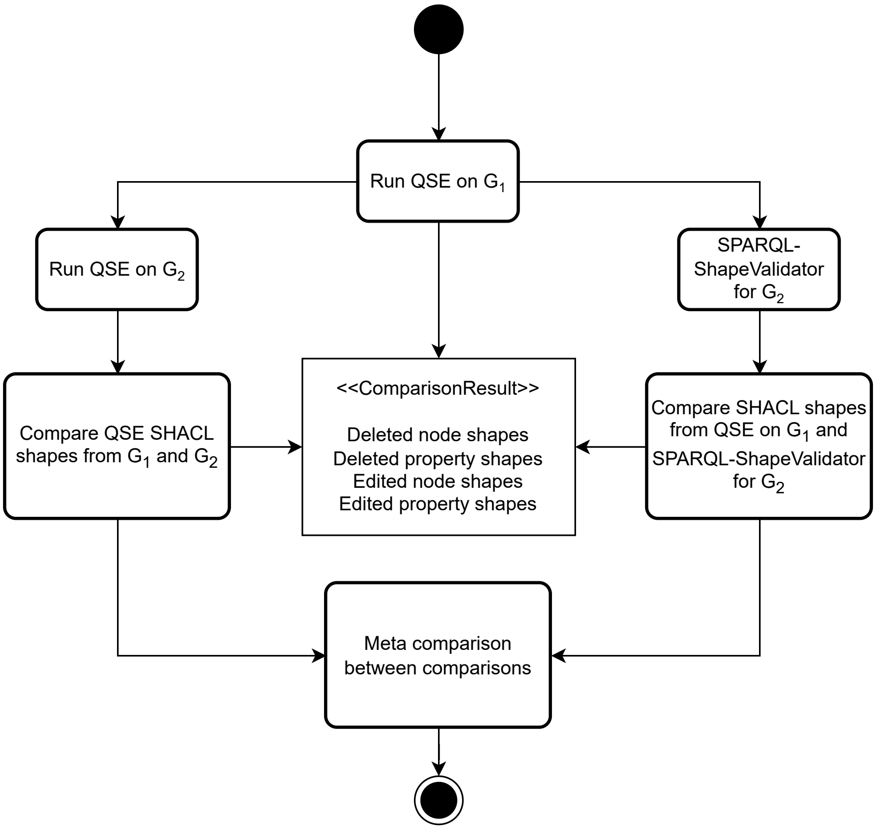

Run QSE on G_1
Run QSE on G_2
SPARQL-ShapeValidator for G_2
Compare QSE SHACL shapes from G_1 and G_2
<<ComparisonResult>>
Deleted node shapes
Deleted property shapes
Edited node shapes
Edited property shapes
Compare SHACL shapes from QSE on G_1 and SPARQL-ShapeValidator for G_2
Meta comparison between comparisons

Figure 7.1: Activity Diagram for the evaluation of SPARQL-ShapeValidator

7.2 SPARQL-ShapeValidator

The evaluation of the SPARQL-ShapeValidator was conducted by comparing its performance to running QSE for both versions of the graph which is the baseline. The measurable characteristic used for this comparison is the execution time.

The process and the composition of the execution times are visualized in Figure 7.1. The left path presents the baseline, and the right one shows the process for the SPARQL-ShapeValidator. For the baseline, the execution time includes the time taken by QSE to generate SHACL shapes for the first version of the graph, which is the same for both the baseline and the SPARQL-ShapeValidator since QSE is run once for both methods. Additionally, the execution time for the baseline includes the time required for QSE to be run on the second version of the graph. For the SPARQL-ShapeValidator, this part is substituted with the execution time of the algorithm itself. The comparison time is included in both execution times. This refers to the time required to compare all shapes from the first version with those from the second version, using the algorithm described in Section 5.1. In this use case only deleted node- and property shapes and edited node- and property shapes are relevant. Added shapes are not considered, since

the SPARQL-ShapeValidator cannot detect them.
To compare the results of the baseline with the SPARQL-ShapeValidator contentwise, a meta comparison was developed which compares the differences found by the baseline and the SPARQL-ShapeValidator with each other. The results of this meta-comparison are outputted in a text file in the project directory. For the people knowledge graph, used in Section 6.4, the comparison between the second and the third version is provided in Listing 7.1. As it was previously mentioned, the "knowsPersonShapeProperty" does not point to the same class anymore which is why the property shape is deleted in the version of the SPARQL-ShapeValidator and edited in the QSE comparison. Also, the node shape of the SPARQL-ShapeValidator is edited, since the reference for the "knowsPersonShapeProperty" is removed. The execution times for this example were measured on a local PC and can therefore be ignored.

```
===== Comparison of Compare-Methods =====
==== Deleted Node Shapes ====
==== Deleted Property Shapes ====
    === Unique in Sparql-Comparison ===
        http://shaclshapes.org/knowsPersonShapeProperty
==== Edited Node Shape Names ====
    === Unique in Sparql-Comparison ===
        http://shaclshapes.org/PersonShape
==== Edited Property Shape Names ====
    === Unique in QSE-Comparison ===
        http://shaclshapes.org/knowsPersonShapeProperty
Execution Time QSE Total: 7 seconds
Execution Time Sparql Total: 1 seconds
```

Listing 7.1: Result of the meta comparison between the SPARQL-ShapeValidator on the People Knowledge Graph (Version 3) with the People Knowledge Graph (Version 2) as basis

The evaluation of the execution times was conducted on the virtual machine described in Section 7.0.1 with the test data described in Section 7.0.2. Seven arbitrary combinations of versions from the test dataset were used. The results of this evaluation are shown in Table 7.1. It is evident that the SPARQL-ShapeValidator is consistently faster, with an average execution time reduction of 30%.
However, as already addressed before, the results are not always the same. The following list provides an overview of scenarios where the results of the meta-comparison differ:

- Node or property shapes were added in the new version. Similarly, for existing node shapes, if property shapes are added, the node shape changes accordingly.

- If the type of a property shape changes (e.g., from rdf:langString to xsd:string), it will be deleted by the algorithm because no triples can be found for the original type anymore.

- If a property shape has multiple constraints with different types which are listed in a SHACL-Or list and one type changes, the property shape will be modified by both algorithms. However, the modifications will differ, as the SPARQL-ShapeValidator deletes the constraint. Similarly, if a new constraint with a new type is added to the property shape, QSE will add it to the property shape, while the SPARQL-ShapeValidator will leave the shape unchanged.

- Property shapes associated with node shapes that have multiple target classes lead to ambiguity. It is unclear which class the property shape refers to. The node shapes also have multiple support entries in this case.
  There also arise problems with the $minCount = 1$ constraint since this is established when the support of the property shapes matches the support of the node shape. In such cases of ambiguity, it becomes challenging to differentiate whether this condition holds true or not. Therefore, sometimes $minCount = 1$ is added, and sometimes it is not. In this scenario, multiple SHACL-OR lists may also arise within the property shape.
  This limitation is also listed in Section 2.2.3.

- When property shapes of the same node shape have identical names, but different IRIs, two different shapes are created, distinguished by suffixes. An example are the shapes producerTelevisionShowShapeProperty and producer_1TelevisionShowShapeProperty. In this scenario, the name of the property shape does not identify a shape correctly anymore. When QSE is run several times and only one of these shapes is edited or deleted, it becomes uncertain which shape was originally meant. Because the comparison algorithm only compares the names of shapes, differences can arise here. Even if both property shapes are still present, it cannot be reliably predicted which one will receive the suffix "_1". Therefore, discrepancies can occur.
  Similarly, there are cases where multiple path triples are created for a property

Version 1	Version 2	Baseline [seconds]	SPARQL-ShapeValidator [seconds]
BearB - 1	BearB - 2	231	173
BearB - 1	BearB - 87	260	174
BearB - 10	BearB - 80	257	178
BearB - 20	BearB - 70	261	184
BearB - 30	BearB - 60	257	186
BearB - 40	BearB - 41	259	199
BearC - 1	BearC - 2	32	22
BearC - 1	BearC - 3	28	18

Table 7.1: Caption

shape (similarly to the case with different suffixes). This results in discrepancies again. An example is provided in Listing 7.2.

- If the support of a property shape changes so that it equals the support of the node shape in the second version, QSE adds the requirement *minCount* = 1. This constraint is not added by the SPARQL-ShapeValidator. However, SPARQL-ShapeValidator deletes the constraint if the condition is not met anymore.

```
<http://shaclshapes.org/residencePersonShapeProperty> rdf:type
  <http://www.w3.org/ns/shacl#PropertyShape> ;
<http://shaclshapes.org/confidence> 2E–1 ;
<http://shaclshapes.org/confidence> 2,2222E–1 ;
<http://shaclshapes.org/support> "1"^^xsd:int ;
<http://shaclshapes.org/support> "2"^^xsd:int ;
<http://www.w3.org/ns/shacl#NodeKind>
    <http://www.w3.org/ns/shacl#IRI> ;
<http://www.w3.org/ns/shacl#class>
    <http://shaclshapes.org/undefined> ;
<http://www.w3.org/ns/shacl#path>
    <http://dbpedia.org/ontology/residence> ;
<http://www.w3.org/ns/shacl#path>
    <http://dbpedia.org/property/residence> .
```

Listing 7.2: Property shape with different paths

CHAPTER 8

Discussion

CHAPTER 9

Conclusion & Future Work

APPENDIX A

Systematic Literature Review

Referenzen von aussortierten Quellen trotzdem behalten?

	Title, Reference	Exclusion Reason
	"evolving knowledge graphs"	
1	Evolving Knowledge Graphs [62]	
2	Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs [83]	Title: Reasoning is not in line thesis topic
3	Summarizing Entity Temporal Evolution in Knowledge Graphs [81]	
4	How does knowledge evolve in open knowledge graphs? [72]	
5	Analysing the Evolution of Knowledge Graphs for the Purpose of Change Verification [66]	
6	EvolveKG: a general framework to learn evolving knowledge graphs [61]	
7	KGdiff: Tracking the Evolution of Knowledge Graphs [59]	
8	Predicting the co-evolution of event and Knowledge Graphs [40]	
9	Knowledge Graphs Evolution and Preservation – A Technical Report from ISWS 2019 [20]	
10	Representing Scientific Literature Evolution via Temporal Knowledge Graphs [76]	Title: Scientific Literature Evolution not in line with thesis topic
	"data quality in knowledge graphs"	
11	GraphGuard: Enhancing Data Quality in Knowledge Graph Pipelines [37]	
12	Improving and Assessing Data Quality of Knowledge Graphs [33]	
13	Knowledge Graph Quality Management: A Comprehensive Survey [86]	

	Continuation of Table A.1	
	Title, Reference	Exclusion Reason
14	Knowledge Graph Completeness: A Systematic Literature Review [54]	
15	Steps to Knowledge Graphs Quality Assessment [53]	
16	What Are Links in Linked Open Data? A Characterization and Evaluation of Links between Knowledge Graphs on the Web [48]	Title: Links in Linked Open Data not in line with thesis topic
17	Knowledge Graphs 2021: A Data Odyssey [85]	
18	Knowledge graphs [41]	
19	A Practical Framework for Evaluating the Quality of Knowledge Graph [28]	
20	Towards Improving the Quality of Knowledge Graphs with Data-driven Ontology Patterns and SHACL [79]	
	"shacl extraction from knowledge graphs"	
21	A Library for Visualizing SHACL over Knowledge Graphs [22]	
22	SCOOP all the Constraints' Flavours for your Knowledge Graph [38]	
23	Learning SHACL shapes from knowledge graphs [8]	
24	Towards Improving the Quality of Knowledge Graphs with Data-driven Ontology Patterns and SHACL [79]	Duplicate
25	Extraction of Validating Shapes from Very Large Knowledge Graphs [74]	
26	Using Knowledge Graph Technologies to Contextualize and Validate Declarations in the Social Security Domain [29]	
27	Automatic extraction of shapes using sheXer [42]	
28	Property assertion constraints for an informed, error-preventing expansion of knowledge graphs [36]	
29	Automatic Construction of SHACL Schemas for RDF Knowledge Graphs Generated by Direct Mappings [30]	Language: Korean, not available with TU-Wien Account
30	Trav-SHACL: Efficiently Validating Networks of SHACL Constraints [44]	
	"comparing shacl shapes"	
31	Comparing ShEx and SHACL [45]	Not available with TU-Wien account
32	Using Ontology Design Patterns To Define SHACL Shapes [68]	
33	Semantic rule checking of cross-domain building data in information containers for linked document delivery using the shapes constraint language [47]	

Continuation of Table A.1		
	Title, Reference	Exclusion Reason
34	Astrea: automatic generation of SHACL shapes from ontologies [31]	
35	Trav-SHACL: Efficiently validating networks of SHACL constraints [44]	Duplicate
36	Formalizing Property Constraints in Wikidata [43]	
37	Semantics and Validation of Recursive SHACL [32]	
38	An Argument for Generating SHACL Shapes from ODPs [69]	
39	Comparison of the eye's wave-front aberration measured psychophysically and with the Shack–Hartmann wave-front sensor [77]	Title: not relevant for knowledge graphs
40	Absolute sphericity measurement:? a comparative study of the use of interferometry and a Shack–Hartmann sensor [1]	Title: not relevant for knowledge graphs
"differences between knowledge graphs versions"		
41	Summarizing entity temporal evolution in knowledge graphs [81]	Duplicate
42	KGDiff: Tracking the evolution of knowledge graphs [59]	Duplicate
43	Knowledge Graphs on the Web-An Overview. [49]	
44	Knowledge graphs: A practical review of the research landscape [58]	
45	Bias in Knowledge Graphs–an Empirical Study with Movie Recommendation and Different Language Editions of DBpedia [84]	Title: Bias not relevant for thesis topic
46	Explaining and suggesting relatedness in knowledge graphs [71]	
47	A survey on knowledge graphs: Representation, acquisition, and applications [56]	
48	Measuring accuracy of triples in knowledge graphs [63]	Title: Accuracy not relevant for thesis topic
49	AYNEC: all you need for evaluating completion techniques in knowledge graphs [25]	
50	Modelling dynamics in semantic web knowledge graphs with formal concept analysis [46]	Title: Modelling dynamics not relevant for thesis topic

Table A.1: All initial search results for all search terms

add space after table

	Title, Reference	Comments	Relevance
1	Evolving Knowledge Graphs [62]	Theoretical description of EvolveKG [61] - a framework that reveals cross-time knowledge interaction with desirable performance. This is partly relevant for this thesis since an extra framework is created with a Derivative Graph, allowing knowledge prediction. In this thesis, only snapshots of evolving graphs are considered.	2
3	Summarizing Entity Temporal Evolution in Knowledge Graphs [81]	Envisions an approach where a summary graph is created to catch the evolution of all entities across all versions.	2
4	How does knowledge evolve in open knowledge graphs? [72]	Explains the basics of evolving knowledge graphs.	1
5	Analysing the Evolution of Knowledge Graphs for the Purpose of Change Verification [66]	This paper deals with topological features of a graph to generate classifiers that can judge whether an incoming graph change is correct or incorrect.	2
6	EvolveKG: a general framework to learn evolving knowledge graphs [61]	Already mentioned in [62].	3
7	KGdiff: Tracking the Evolution of Knowledge Graphs [59]	Proposes a software that tracks changes on the schema and the individuals.	2
8	Predicting the co-evolution of event and Knowledge Graphs [40]	The goal of this paper is to predict unobserved facts by using previous temporal information from knowledge graphs and static information. This is not relevant since prediction is not part of this thesis.	3
9	Knowledge Graphs Evolution and Preservation – A Technical Report from ISWS 2019 [20]	Collection of ten papers: Papers 5,6,10 are not relevant. Paper 1 differentiates between different types of evolution and checks if machine learning can capture this evolution. Paper 2 focuses on the characteristics of an evolving knowledge graph, in this case, DBpedia. The third paper discusses changes between two versions of a knowledge graph. Paper 4 is about changes is ontologies and how they can be characterized. Paper 7 deals with the integration of data in knowledge graphs. Versioned knowledge graphs are targeted in Chapter 8. Finally, paper 9 deals with the support of interactive updates in knowledge graphs.	2

Continuation of Table A.2			
	Title, Reference	Comments	Relevance
11	GraphGuard: Enhancing Data Quality in Knowledge Graph Pipelines [37]	Introduces a framework for better data quality in knowledge graph pipelines for humans and machines. This paper is partly relevant to this thesis because the goal of QSE and SHACL is not to check data quality during data ingestion.	2
12	Improving and Assessing Data Quality of Knowledge Graphs [33]	Dissertation which focuses on including data transformations in knowledge graphs that can help to clean the data and complete knowledge graphs by calculating derived data. Furthermore, this paper is about validating knowledge graphs. Validation is done by a reasoning solution called Validatrr. Cleaning, completing and reasoning are not directly addressed by this therefore this paper is not directly relevant.	2
13	Knowledge Graph Quality Management: A Comprehensive Survey [86]	This paper provides a systematic review of quality management in knowledge graphs, also including quality management processes, such as quality assessment, error detection, error correction and completion.	2
14	Knowledge Graph Completeness: A Systematic Literature Review [54]	Different quality dimensions exist - such as accuracy or completeness. This paper summarizes terminologies related to completeness. Therefore it is not relevant for this thesis, since completeness is not a topic.	3
15	Steps to Knowledge Graphs Quality Assessment [53]	The goal of this paper is to extend existing quality assessment frameworks by adding quality dimensions and quality metrics. It is only partly relevant for this thesis since quality dimensions are evaluated with SHACL.	2
17	Knowledge Graphs 2021: A Data Odyssey [85]	This paper discusses advances and lessons learned in the history of knowledge graphs. The abstract is too general that it could be relevant for this thesis.	3
18	Knowledge graphs [41]	Entire book on knowledge graphs that includes a general introduction, how to build and use knowledge graphs and specific use cases.	2
19	A Practical Framework for Evaluating the Quality of Knowledge Graph [28]	In this paper, existing frameworks for quality of knowledge graphs are assessed and a practical framework is proposed which determines if a knowledge graph is fit for purpose.	3

Continuation of Table A.2			
	Title, Reference	Comments	Relevance
20	Towards Improving the Quality of Knowledge Graphs with Data-driven Ontology Patterns and SHACL. [79]	The approach of this paper is used in the QSE approach. It proposes a semantic profiling tool that helps to enhance the understanding of the data by using SHACL.	1
21	A Library for Visualizing SHACL over Knowledge Graphs [22]	This master thesis created a library called SHACLViewer which illustrates SHACL shapes in a 3D context.	1
22	SCOOP all the Constraints' Flavours for your Knowledge Graph [38]	The goal of the SCOOP framework is to extract SHACL shapes from already existing knowledge graphs (similar to QSE). However, it mainly uses ontologies and data schemas instead of the entities itself.	2
23	Learning SHACL shapes from knowledge graphs [8]	This paper introduces Inverse Open Path (IOP), a predicate logic formalism that presents specific shapes over connected entities from the knowledge graph. The corresponding learning method is called SHACLearner.	2
25	Extraction of Validating Shapes from Very Large Knowledge Graphs [74]	This thesis builds on this paper, therefore it is excluded.	3
26	Using Knowledge Graph Technologies to Contextualize and Validate Declarations in the Social Security Domain [29]	A master thesis, which generates SHACL shapes for a specific use case, namely forms which describe the work of employees in a quarter. Since SHACL shapes are generated manually, the thesis is not relevant to this thesis.	3
27	Automatic extraction of shapes using sheXer [42]	sheXer produces SHACL shapes similar to QSE using a python library. This paper is also mentioned in the related work of QSE.	2
28	Property assertion constraints for an informed, error-preventing expansion of knowledge graphs [36]	PAC (property assertion constraints) have the goal of checking data before it gets added to the knowledge graph. With this approach, errors can be prevented and the quality of knowledge graphs can be enhanced. PAC works by restricting the range of properties using SPARQL.	2

Continuation of Table A.2			
	Title, Reference	Comments	Relevance
30	Trav-SHACL: Efficiently Validating Networks of SHACL Constraints [44]	Trav-SHACL plans the execution and the traversal of a shapes graph so that invalid entries are detected early. For this task, the shapes graph is reordered.	2
32	Using Ontology Design Patterns To Define SHACL Shapes [68]	Similar to [38], this paper wants to reuse the Ontology Design Pattern (ODP) and contexts to automatically generate SHACL shapes.	2
33	Semantic rule checking of cross-domain building data in information containers for linked document delivery using the shapes constraint language [47]	This case study uses SHACL in the domain of Information Container for Linked Document Delivery (ICDD). Since this is a specific case study, it is not relevant for this thesis.	3
34	Astrea: automatic generation of SHACL shapes from ontologies [31]	Similar to [38] this approach also uses ontologies to automatically generate SHACL shapes. Astrea uses Astrea-KG which provides mappings between ontology constraint patterns and SHACL constraint patterns.	2
36	Formalizing Property Constraints in Wikidata [43]	This paper compares constraints in Wikidata, which uses its own RDF data model for constraints, with constraints that can be created with SPARQL and SHACL. This paper is not relevant, since Wikidata will not be used in this thesis.	3
37	Semantics and Validation of Recursive SHACL [32]	As the title suggests, this paper explores the recursion for SHACL constraints. It proposes concise formal semantics of the core elements of SHACL and validates recursion for these elements. This paper is not relevant since the recursion of SHACL elements is not a topic for this thesis.	3
38	An Argument for Generating SHACL Shapes from ODPs [69]	This book chapter extends the idea of [68].	3
43	Knowledge Graphs on the Web-An Overview. [49]	This book chapter provides an overview and comparison of publicly available knowledge graphs (DBpedia, Wikidata) and gives insights into their sizes and contents. This chapter is not relevant to this thesis since it is too general.	3

Continuation of Table A.2			
	Title, Reference	Comments	Relevance
44	Knowledge graphs: A practical review of the research landscape [58]	The cross-disciplinary nature of knowledge graphs is very important. This paper gives an overview of the major strands in the research landscapes and their different communities. This paper is too general for this thesis.	3
46	Explaining and suggesting relatedness in knowledge graphs [71]	It provides a tool called RECAP, which explains the relatedness of a pair of entities in a knowledge graph. Relatedness is not a topic of this thesis, therefore it is not relevant.	3
47	A survey on knowledge graphs: Representation, acquisition, and applications [56]	This survey covers a broad range of topics regarding knowledge graphs. However, also temporal knowledge graphs are discussed, which makes the paper partly relevant.	2
49	AYNEC: all you need for evaluating completion techniques in knowledge graphs [25]	The tool AYNEC provides a suite for the evaluation of knowledge graph completion techniques. Since knowledge graph completion is not a topic of this thesis, this paper is not relevant.	3

Table A.2: Comments and possible exclusion reasons on all papers after the first round based on the Abstract. Relevance ranges from 1 (absolutely relevant) to 3 (not relevant)

add space after table

APPENDIX B

Demonstration Knowledge Graphs

```
<http://example.org/alice>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/bob>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/jenny>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/alice>
    <http://xmlns.com/foaf/0.1/name> "Alice" .
<http://example.org/bob>
    <http://xmlns.com/foaf/0.1/name> "Bob" .
<http://example.org/jenny>
    <http://xmlns.com/foaf/0.1/name> "Jenny" .
<http://example.org/alice>
    <http://xmlns.com/foaf/0.1/knows>
    <http://example.org/bob> .
<http://example.org/bob>
    <http://xmlns.com/foaf/0.1/knows>
    <http://example.org/alice> .
<http://example.org/jenny>
    <http://xmlns.com/foaf/0.1/knows>
    <http://example.org/alice> .
```

Listing B.1: People Knowledge Graph (Version 1)

```
<http://example.org/alice>
```

```
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://xmlns.com/foaf/0.1/Person> .
    <http://example.org/bob>
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://xmlns.com/foaf/0.1/Person> .
    <http://example.org/jenny>
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://xmlns.com/foaf/0.1/Person> .
    <http://example.org/alice>
        <http://xmlns.com/foaf/0.1/name> "Alice" .
    <http://example.org/bob>
        <http://xmlns.com/foaf/0.1/name> "Bob" .
    <http://example.org/jenny>
        <http://xmlns.com/foaf/0.1/name> "Jenny" .
    <http://example.org/alice>
        <http://xmlns.com/foaf/0.1/knows>
        <http://example.org/bob> .
    <http://example.org/bob>
        <http://xmlns.com/foaf/0.1/knows>
        <http://example.org/alice> .
    <http://example.org/jenny>
        <http://xmlns.com/foaf/0.1/knows>
        <http://example.org/alice> .
    <http://example.org/orangeCat>
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://xmlns.com/foaf/0.1/Cat> .
    <http://example.org/blackCat>
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://xmlns.com/foaf/0.1/Cat> .
    <http://example.org/greyCat>
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://xmlns.com/foaf/0.1/Cat> .
    <http://example.org/orangeCat>
        <http://example.org/color> "orange" .
    <http://example.org/blackCat>
        <http://example.org/color> "black" .
    <http://example.org/greyCat>
        <http://example.org/color> "grey" .
```

Listing B.2: People Knowledge Graph (Version 2)

```
    <http://example.org/alice>
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
        <http://xmlns.com/foaf/0.1/Person> .
```

```
<http://example.org/bob>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/jenny>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/alice>
    <http://xmlns.com/foaf/0.1/name> "Alice" .
<http://example.org/bob>
    <http://xmlns.com/foaf/0.1/name> "Bob" .
<http://example.org/jenny>
    <http://xmlns.com/foaf/0.1/name> "Jenny" .
<http://example.org/alice>
    <http://xmlns.com/foaf/0.1/knows>
    <http://example.org/orangeCat> .
<http://example.org/bob>
    <http://xmlns.com/foaf/0.1/knows>
    <http://example.org/blackCat> .
<http://example.org/jenny>
    <http://xmlns.com/foaf/0.1/knows>
    <http://example.org/greyCat> .
<http://example.org/orangeCat>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Cat> .
<http://example.org/blackCat>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Cat> .
<http://example.org/greyCat>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Cat> .
<http://example.org/orangeCat>
    <http://example.org/color> "orange" .
```

Listing B.3: People Knowledge Graph (Version 3)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://shaclshapes.org/PersonShape> rdf:type
    <http://www.w3.org/ns/shacl#NodeShape> ;
  <http://shaclshapes.org/support> "3"^^xsd:int ;
  <http://www.w3.org/ns/shacl#property>
    <http://shaclshapes.org/instanceTypePersonShapeProperty> ;
  <http://www.w3.org/ns/shacl#property>
```

```
    <http://shaclshapes.org/knowsPersonShapeProperty> ;
  <http://www.w3.org/ns/shacl#property>
    <http://shaclshapes.org/namePersonShapeProperty> ;
  <http://www.w3.org/ns/shacl#targetClass>
    <http://xmlns.com/foaf/0.1/Person> .

<http://shaclshapes.org/instanceTypePersonShapeProperty>
    rdf:type <http://www.w3.org/ns/shacl#PropertyShape> ;
  <http://shaclshapes.org/confidence> 1E0 ;
  <http://shaclshapes.org/support> "3"^^xsd:int ;
  <http://www.w3.org/ns/shacl#in>
    (<http://xmlns.com/foaf/0.1/Person> ) ;
  <http://www.w3.org/ns/shacl#path> rdf:type .

<http://shaclshapes.org/knowsPersonShapeProperty>
    rdf:type <http://www.w3.org/ns/shacl#PropertyShape> ;
  <http://shaclshapes.org/confidence> 1E0 ;
  <http://shaclshapes.org/support> "3"^^xsd:int ;
  <http://www.w3.org/ns/shacl#NodeKind>
    <http://www.w3.org/ns/shacl#IRI> ;
  <http://www.w3.org/ns/shacl#class>
    <http://xmlns.com/foaf/0.1/Person> ;
  <http://www.w3.org/ns/shacl#minCount> 1 ;
  <http://www.w3.org/ns/shacl#node>
    <http://shaclshapes.org/PersonShape> ;
  <http://www.w3.org/ns/shacl#path>
    <http://xmlns.com/foaf/0.1/knows> .

<http://shaclshapes.org/namePersonShapeProperty>
    rdf:type <http://www.w3.org/ns/shacl#PropertyShape> ;
  <http://shaclshapes.org/confidence> 1E0 ;
  <http://shaclshapes.org/support> "3"^^xsd:int ;
  <http://www.w3.org/ns/shacl#NodeKind>
    <http://www.w3.org/ns/shacl#Literal> ;
  <http://www.w3.org/ns/shacl#datatype> xsd:string ;
  <http://www.w3.org/ns/shacl#minCount> 1 ;
  <http://www.w3.org/ns/shacl#path>
    <http://xmlns.com/foaf/0.1/name> .
```

Listing B.4: SHACL shapes for the People Knowledge Graph (Version 1) from QSE-Exact with no paramters for support or confidence

List of Figures

List of Tables

List of Algorithms

Referenzen überarbeiten mit Feedback von erster Version. Dblp verwenden.

Bibliography

[1] Absolute sphericity measurement:?a comparative study of the use of interferometry and a Shack–Hartmann sensor.

[2] Apache Jena - Home. \url{https://jena.apache.org/}.

[3] BEAR | BEnchmark of RDF ARchives.

[4] Dataset releases Archives.

[5] dkw-aau/qse at shactor.

[6] Free prototyping tool for web & mobile apps - Justinmind.

[7] GraphDB Downloads and Resources.

[8] Learning SHACL shapes from knowledge graphs - IOS Press.

[9] Linked Life Data - A Semantic Data Integration Platform for the Biomedical Domain.

[10] OpenLink Software: Virtuoso Homepage.

[11] OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition).

[12] RDF Schema 1.1.

[13] Shapes Constraint Language (SHACL).

[14] ShEx - Shape Expressions.

[15] SPARQL 1.1 Overview.

[16] Vaadin Docs.

[17] Wikidata.

[18] Shapes Constraint Language (SHACL), July 2017.

[19] Home, May 2024.

[20] Abbas, N., Alghamdi, K., Alinam, M., Alloatti, F., Amaral, G., d'Amato, C., Asprino, L., Beno, M., Bensmann, F., Biswas, R., Cai, L., Capshaw, R., Carriero, V. A., Celino, I., Dadoun, A., De Giorgis, S., Delva, H., Domingue, J., Dumontier, M., Emonet, V., van Erp, M., Arias, P. E., Fallatah, O., Ferrada, S., Ocaña, M. G., Georgiou, M., Gesese, G. A., Gillis-Webber, F., Giovannetti, F., Buey, M. G., Harrando, I., Heibi, I., Horta, V., Huber, L., Igne, F., Jaradeh, M. Y., Keshan, N., Koleva, A., Koteich, B., Kurniawan, K., Liu, M., Ma, C., Maas, L., Mansfield, M., Mariani, F., Marzi, E., Mesbah, S., Mistry, M., Tirado, A. C. M., Nguyen, A., Nguyen, V. B., Oelen, A., Pasqual, V., Paulheim, H., Polleres, A., Porena, M., Portisch, J., Presutti, V., Pustu-Iren, K., Mendez, A. R., Roshankish, S., Rudolph, S., Sack, H., Sakor, A., Salas, J., Schleider, T., Shi, M., Spinaci, G., Sun, C., Tietz, T., Dhouib, M. T., Umbrico, A., Berg, W. v. d., and Xu, W. Knowledge Graphs Evolution and Preservation – A Technical Report from ISWS 2019, Dec. 2020. arXiv:2012.11936 [cs].

[21] Adams, W. Conducting Semi-Structured Interviews. Aug. 2015.

[22] Alom, H. A Library for Visualizing SHACL over Knowledge Graphs. Master's thesis, Hannover : Gottfried Wilhelm Leibniz Universität Hannover, Mar. 2022.

[23] Antony, J. 2 - Fundamentals of Design of Experiments. In *Design of Experiments for Engineers and Scientists (Second Edition)*, J. Antony, Ed., second edition ed. Elsevier, Oxford, 2014, pp. 7–17.

[24] Antony, J. 4 - A Systematic Methodology for Design of Experiments. In *Design of Experiments for Engineers and Scientists (Second Edition)*, J. Antony, Ed., second edition ed. Elsevier, Oxford, 2014, pp. 33–50.

[25] Ayala, D., Borrego, A., Hernández, I., Rivero, C. R., and Ruiz, D. AYNEC: All You Need for Evaluating Completion Techniques in Knowledge Graphs. In *The Semantic Web* (Cham, 2019), P. Hitzler, M. Fernández, K. Janowicz, A. Zaveri, A. J. Gray, V. Lopez, A. Haller, and K. Hammar, Eds., Springer International Publishing, pp. 397–411.

[26] Boneva, I., Dusart, J., Fernández Alvarez, D., and Gayo, J. E. L. Shape Designer for ShEx and SHACL Constraints, Oct. 2019. Published: ISWC 2019 - 18th International Semantic Web Conference.

[27] Camburn, B., Viswanathan, V., Linsey, J., Anderson, D., Jensen, D., Crawford, R., Otto, K., and Wood, K. Design prototyping methods: state of the art in strategies, techniques, and guidelines. *Design Science 3* (2017), e13.

[28] Chen, H., Cao, G., Chen, J., and Ding, J. A Practical Framework for Evaluating the Quality of Knowledge Graph. In *Knowledge Graph and Semantic Computing: Knowledge Computing and Language Understanding* (Singapore, 2019), X. Zhu, B. Qin, X. Zhu, M. Liu, and L. Qian, Eds., Springer, pp. 111–122.

[29] Chiem Dao, D., and Université de Liège > Master ingé. civ. info., F. Using Knowledge Graph Technologies to Contextualize and Validate Declarations in the Social Security Domain. Accepted: 2023-07-12T02:11:50Z Publisher: Université de Liège, Liège, Belgique Section: Université de Liège.

[30] Choi, J.-W. Automatic Construction of SHACL Schemas for RDF Knowledge Graphs Generated by Direct Mappings. *25*, 10 (2020), 23–34.

[31] Cimmino, A., Fernández-Izquierdo, A., and García-Castro, R. Astrea: Automatic Generation of SHACL Shapes from Ontologies. In *The Semantic Web* (Cham, 2020), A. Harth, S. Kirrane, A.-C. Ngonga Ngomo, H. Paulheim, A. Rula, A. L. Gentile, P. Haase, and M. Cochez, Eds., Springer International Publishing, pp. 497–513.

[32] Corman, J., Reutter, J. L., and Savković, O. Semantics and Validation of Recursive SHACL. In *The Semantic Web – ISWC 2018* (Cham, 2018), D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, and E. Simperl, Eds., Springer International Publishing, pp. 318–336.

[33] De Meester, B. *Improving and assessing data quality of knowledge graphs.* PhD Thesis, Ghent University, 2020.

[34] Decker, K. kpdecker/jsdiff, Apr. 2024. original-date: 2011-03-29T16:25:28Z.

[35] developers, E. R. Welcome · Eclipse RDF4J™ | The Eclipse Foundation.

[36] Dibowski, H. *Property Assertion Constraints for an Informed, Error-Preventing Expansion of Knowledge Graphs.* Nov. 2021. Pages: 248.

[37] Dorsch, R., Freund, M., Fries, J., and Harth, A. GraphGuard: Enhancing Data Quality in Knowledge Graph Pipelines.

[38] Duan, X. dtai-kg/SCOOP: v1.0.0, Dec. 2023.

[39] Durakovic, B. Design of experiments application, concepts, examples: State of the art. *Periodicals of Engineering and Natural Sciences 5* (Dec. 2017), 421–439.

[40] Esteban, C., Tresp, V., Yang, Y., Baier, S., and Krompaß, D. Predicting the co-evolution of event and Knowledge Graphs. In *2016 19th International Conference on Information Fusion (FUSION)* (July 2016), pp. 98–105.

[41] Fensel, D., Simsek, U., Angele, K., Huaman, E., Kärle, E., Panasiuk, O., Toma, I., Umbrich, J., and Wahler, A. *Knowledge graphs.* Springer, 2020.

[42] Fernandez-Álvarez, D., Labra-Gayo, J. E., and Gayo-Avello, D. Automatic extraction of shapes using sheXer. *Knowledge-Based Systems 238* (Feb. 2022), 107975.

[43] Ferranti, N., Polleres, A., de Souza, J. F., and Ahmetaj, S. Formalizing Property Constraints in Wikidata.

[44] Figuera, M., Rohde, P. D., and Vidal, M.-E. Trav-SHACL: Efficiently Validating Networks of SHACL Constraints. In *Proceedings of the Web Conference 2021* (New York, NY, USA, June 2021), WWW '21, Association for Computing Machinery, pp. 3337–3348.

[45] Gayo, J. E. L., Prud'hommeaux, E., Boneva, I., and Kontokostas, D. Comparing ShEx and SHACL. In *Validating RDF Data*, J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas, Eds. Springer International Publishing, Cham, 2018, pp. 233–266.

[46] González, L., and Hogan, A. Modelling Dynamics in Semantic Web Knowledge Graphs with Formal Concept Analysis. In *Proceedings of the 2018 World Wide Web Conference* (Republic and Canton of Geneva, CHE, Apr. 2018), WWW '18, International World Wide Web Conferences Steering Committee, pp. 1175–1184.

[47] Hagedorn, P., Pauwels, P., and König, M. Semantic rule checking of cross-domain building data in information containers for linked document delivery using the shapes constraint language. *Automation in Construction 156* (Dec. 2023), 105106.

[48] Haller, A., Fernández, J. D., Kamdar, M. R., and Polleres, A. What Are Links in Linked Open Data? A Characterization and Evaluation of Links between Knowledge Graphs on the Web. *Journal of Data and Information Quality 12*, 2 (May 2020), 9:1–9:34.

[49] Heist, N., Hertling, S., Ringler, D., and Paulheim, H. Knowledge Graphs on the Web-An Overview. *Knowledge Graphs for eXplainable Artificial Intelligence* (2020), 3–22.

[50] Hevner, A. A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems 19* (Jan. 2007).

[51] Hevner, A., R, A., March, S., T, S., Park, Park, J., Ram, and Sudha. Design Science in Information Systems Research. *Management Information Systems Quarterly 28* (Mar. 2004), 75–.

[52] Hose, K., and Sallinger, E. Introduction to Semantic Systems, Nov. 2024.

[53] Huaman, E. Steps to Knowledge Graphs Quality Assessment, Aug. 2022.

[54] Issa, S., Adekunle, O., Hamdi, F., Cherfi, S. S.-S., Dumontier, M., and Zaveri, A. Knowledge Graph Completeness: A Systematic Literature Review. *IEEE Access 9* (2021), 31322–31339. Conference Name: IEEE Access.

[55] Jesús, M. Scientific Prototyping: A Novel Approach to Conduct Research and Engineer Products. pp. 32–35.

[56] Ji, S., Pan, S., Cambria, E., Marttinen, P., and Yu, P. S. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems 33*, 2 (Feb. 2022), 494–514. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

[57] Keely, A. shaclgen: Shacl graph generator.

[58] Kejriwal, M. Knowledge Graphs: A Practical Review of the Research Landscape. *Information 13*, 4 (Apr. 2022), 161. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[59] Keshavarzi, A., and Kochut, K. J. KGdiff: Tracking the Evolution of Knowledge Graphs. In *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)* (Aug. 2020), pp. 279–286.

[60] Kitchenham, B., and Charters, S. Guidelines for performing Systematic Literature Reviews in Software Engineering.

[61] Liu, J., Yu, Z., Guo, B., Deng, C., Fu, L., Wang, X., and Zhou, C. EvolveKG: a general framework to learn evolving knowledge graphs. *Frontiers of Computer Science 18*, 3 (Jan. 2024), 183309.

[62] Liu, J., Zhang, Q., Fu, L., Wang, X., and Lu, S. Evolving Knowledge Graphs. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications* (Paris, France, Apr. 2019), IEEE, pp. 2260–2268.

[63] Liu, S., d'Aquin, M., and Motta, E. Measuring Accuracy of Triples in Knowledge Graphs. In *Language, Data, and Knowledge* (Cham, 2017), J. Gracia, F. Bond, J. P. McCrae, P. Buitelaar, C. Chiarcos, and S. Hellmann, Eds., Springer International Publishing, pp. 343–357.

[64] Mayring, P. Qualitative Content Analysis. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research [On-line Journal], http://qualitative-research.net/fqs/fqs-e/2-00inhalt-e.htm 1* (June 2000).

[65] Mihindukulasooriya, N., Rashid, M. R. A., Rizzo, G., García-Castro, R., Corcho, O., and Torchiano, M. RDF shape induction using knowledge base profiling. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (Pau France, Apr. 2018), ACM, pp. 1952–1959.

[66] Nishioka, C., and Scherp, A. Analysing the Evolution of Knowledge Graphs for the Purpose of Change Verification. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)* (Jan. 2018), pp. 25–32.

[67] Omran, P. G., Taylor, K., Mendez, S. R., and Haller, A. Towards SHACL Learning from Knowledge Graphs.

[68] Pandit, H. J., O'Sullivan, D., and Lewis, D. Using Ontology Design Patterns To Define SHACL Shapes.

[69] Pandit, H. J., O'Sullivan, D., and Lewis, D. An Argument for Generating SHACL Shapes from ODPs. In *Advances in Pattern-Based Ontology Engineering.* IOS Press, 2021, pp. 134–141.

[70] Peffers, K., Rothenberger, M., Tuunanen, T., and Vaezi, R. Design Science Research Evaluation. vol. 7286, pp. 398–410.

[71] Pirrò, G. Explaining and Suggesting Relatedness in Knowledge Graphs. In *The Semantic Web - ISWC 2015* (Cham, 2015), M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, K. Thirunarayan, and S. Staab, Eds., Springer International Publishing, pp. 622–639.

[72] Polleres, A., Pernisch, R., Bonifati, A., Dell'Aglio, D., Dobriy, D., Dumbrava, S., Etcheverry, L., Ferranti, N., Hose, K., Jiménez-Ruiz, E., Lissandrini, M., Scherp, A., Tommasini, R., and Wachs, J. How does knowledge evolve in open knowledge graphs? *Transactions on Graph Data and Knowledge 1*, 1 (Dec. 2023), 11:1–11:59. Publisher: Dagstuhl.

[73] PubChem. About.

[74] Rabbani, K., Lissandrini, M., and Hose, K. Extraction of Validating Shapes from Very Large Knowledge Graphs. *Proceedings of the VLDB Endowment 16*, 5 (Jan. 2023), 1023–1032.

[75] Rabbani, K., Lissandrini, M., and Hose, K. SHACTOR: Improving the Quality of Large-Scale Knowledge Graphs with Validating Shapes. In *Companion of the 2023 International Conference on Management of Data* (New York, NY, USA, 2023), SIGMOD '23, Association for Computing Machinery, pp. 151–154. event-place: Seattle, WA, USA.

[76] Rossanez, A., Reis, J., and Torres, R. D. S. Representing Scientific Literature Evolution via Temporal Knowledge Graphs. *CEUR Workshop Proceedings* (2020). Accepted: 2021-09-06T11:20:56Z Publisher: CEUR Workshop Proceedings.

[77] Salmon, T. O., Thibos, L. N., and Bradley, A. Comparison of the eye's wave-front aberration measured psychophysically and with the Shack–Hartmann wave-front sensor. *JOSA A 15*, 9 (Sept. 1998), 2457–2465. Publisher: Optica Publishing Group.

[78] Sanders, P. Algorithm Engineering – An Attempt at a Definition. In *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, S. Albers, H. Alt, and S. Näher, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 321–340.

[79] Spahiu, B., Maurino, A., and Palmonari, M. Towards Improving the Quality of Knowledge Graphs with Data-driven Ontology Patterns and SHACL.

[80] Szeredi, P., Lukácsy, G., Benkő, T., and Nagy, Z. The Semantic Web and the RDF language. In *The Semantic Web Explained: The Technology and Mathematics behind Web 3.0*. Cambridge University Press, 2014, pp. 52–121.

[81] Tasnim, M., Collarana, D., Graux, D., Orlandi, F., and Vidal, M.-E. Summarizing Entity Temporal Evolution in Knowledge Graphs. In *Companion Proceedings of The 2019 World Wide Web Conference* (San Francisco USA, May 2019), ACM, pp. 961–965.

[82] Textor, A. atextor/turtle-formatter, Oct. 2023. original-date: 2021-02-03T05:07:11Z.

[83] Trivedi, R., Dai, H., Wang, Y., and Song, L. Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs. In *Proceedings of the 34th International Conference on Machine Learning* (July 2017), PMLR, pp. 3462–3471. ISSN: 2640-3498.

[84] Voit, M. M., and Paulheim, H. Bias in Knowledge Graphs – an Empirical Study with Movie Recommendation and Different Language Editions of DBpedia, May 2021. arXiv:2105.00674 [cs].

[85] Weikum, G. Knowledge graphs 2021: a data odyssey. *Proceedings of the VLDB Endowment 14*, 12 (July 2021), 3233–3238.

[86] Xue, B., and Zou, L. Knowledge Graph Quality Management: A Comprehensive Survey. *IEEE Transactions on Knowledge and Data Engineering 35*, 5 (May 2023), 4969–4988. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

Referenzen überarbeiten mit Feedback von erster Version. Dblp verwenden.