A Query-Driven Approach for SHACL Type Inference

David Haller[1]

[1]Supervised by Richard Lenz, Friedrich-Alexander-Universität Erlangen-Nürnberg, Professorship for Evolutionary Data Management

Abstract

The Semantic Web enables everyone to share knowledge that can be reused in different applications. While the use of a formal ontology describing the semantics of the shared data is encouraged, it cannot be enforced and is often done incorrectly, incompletely, or not at all. However, the semantics are present in the minds of those working with the data, as manifested in all the SPARQL queries they have written. Therefore, analyzing these query logs helps us to learn these semantics and allows us to construct a graph of SHACL shapes describing the types and their constraints of a data source, which can serve as the foundation for a human-in-the-loop approach to further extend and correct the generated schema.

Keywords

Semantic Web, Schema Inference, Query-Driven, Data Integration

1. Introduction

In a perfect world, data always represents information with known semantics. This is easy to achieve when the applications that produce and consume data are always built by the same people, and the underlying schema is rarely changed and well understood. In reality, data is constantly being used in new contexts, shared with external parties, and combined with other data from different sources. As a result, the meaning of data is not always clear, because everyone may have their own concept of seeing the world, so data is not interpreted in the way it was intended, which can lead to incorrect conclusions and errors in applications.

The Semantic Web has introduced standards to make it easier to share data across applications so that it can be reused in a different context. In the past, knowledge was only made available in human-readable form, such as texts and images, or in proprietary formats that could only be processed by special programs. In the present, machines can also make use of the knowledge available on the World Wide Web, similarly to humans. This was done by establishing the Resource Description Framework (RDF), a flexible data model based on directed graphs, whose nodes are globally addressable with an Internationalized Resource Identifier (IRI) [1]. The advantage of this model is that you can reference anything in another dataset by simply inserting a new edge, the same as you would use a hyperlink to point to a different web page. The disadvantage is that nobody can prevent multiple individuals talking about different things using the same terms, meaning that there is no restriction that only schema-compliant statements can be added. Of course, the Semantic Web also introduced methods to formally define an ontology as the foundation of a knowledge graph, using schema languages such as RDF Schema (RDFS), Web Ontology Language (OWL), or Shapes Constraint Language (SHACL), which even allow us to derive new statements from existing statements by applying logical conclusions. While traditional database systems typically follow the closed-world assumption (a statement is true if and only if it is explicitly stated), the Semantic Web is based on the open-world assumption (a statement can be true even if it is not present in the graph, because it may be added later or stated elsewhere).

The problem is that humans make mistakes, either in creating an ontology or in interpreting it, which can have huge consequences. For example, it is shown in [2] that even if 99.9% of a knowledge base is correct, a few wrong statements can cause the reasoner to infer types that are obviously nonsense. This issue is illustrated with a short example in RDFS:

```
:Berlin :location :Germany
:location rdfs:domain :City
:location rdfs:range :Country
```

A reasoner that is applying the rules given by `rdfs:domain` and `rdfs:range`, would infer that Berlin must be a city, and Germany must be a country, which is correct in the real world. Let's add the following triple:

```
:Zugspitze :location :Germany
```

Now the reasoner would infer that the Zugspitze is also a city, but in reality it is Germany's highest mountain and is located within Germany. The property `:location` was used without considering its domain and scope. Knowledge graphs often suffer from problems like this. Therefore, it is necessary to develop a method to identify and correct these problems.

VLDB 2023 PhD Workshop, co-located with the 49th International Conference on Very Large Data Bases (VLDB 2023), August 28, 2023, Vancouver, Canada

david.haller@fau.de (D. Haller)
https://cs6.tf.fau.eu/person/david-haller (D. Haller)
0000-0001-5287-7187 (D. Haller)

2. Approach

The semantics of the data stored in a knowledge graph is determined by the people who created it in the first place. They had a certain perception of reality in their minds, which guided their choice of data types and properties. This can be called a *mental model*. As described in [3], models used in databases or applications are not created from objectively perceivable reality, but from the individual perspectives of both their creators and their users (which may not always coincide). In cases where an ontology is not available, erroneous, or incomplete, it is impossible to validate the data using a schema consistency check. The mental model needs to be inferred from other sources. Doing so manually is a tedious and error-prone task, so it is desirable to automate the process as far as possible.

When people write queries that access, transform, or modify data, they do so based on their own mental model. As a consequence, the approach presented in this paper is based on analyzing SPARQL query logs and using the extracted information to automatically infer type constraints that can be used to create SHACL shapes for each referenced resource or property. A large set of queries can thus be translated into a SHACL graph that serves as a schema definition for the actual RDF graph containing the data. Each SPARQL query using the verbs `select`, `ask`, or `construct` contains information about the entities it is interested in. Their `where` clauses each contain a *basic graph pattern*, which is a set of triple patterns having the structure $(T \cup V) \times (I \cup V) \times (T \cup V)$, where V is the set of query variables, I the set of all IRIs, and $T = I \cup L \cup B$ with L and B being the set of literals and blank nodes. A *basic graph pattern* restricts the possible structure of the subgraph a query searches for, queries can thus be seen as partial schema definitions. They reflect, at least in part, how the author envisions their knowledge graph; this applies also to the usage of filters (boolean expressions over query variables) and solution modifiers (like `order by`). For example, someone could write a query such as:

```
select ?name ?web count(?child)
where
{
  ?person :name ?name .
  ?person :web ?web .
  ?person :child ?child .
  ?child :age ?age .
  filter (?age < 18 && ?web == "http
      ://example.com")
}
group by (?person)
order by (?name)
```

We learn from this query that its author expected the knowledge graph to have entities of type *Person*. A *Person* has a `name` attribute that is sortable, a `web` attribute that can contain URLs, and a multi-valued `child` attribute (because the edges are counted, which only makes sense if you expect the result to be greater than 1 in some occurrences). We also know that the ?person variable must be an IRI, because only IRIs are allowed as subjects in RDF and SPARQL, and the variable is also used for grouping, so it must be some kind of candidate key for a *Person*. The ?child variable must be an IRI because it has a `age` attribute, which is numeric and has 18 as a somewhat meaningful threshold.

This information extracted from a single SPARQL query can be represented as a SHACL shape. As mentioned earlier, SHACL is a schema language that can be used to describe the semantics of a knowledge graph and, like OWL and RDFS, is a W3C recommendation. Unlike OWL, SHACL focuses on restricting which triples are allowed by defining a set of patterns, called shapes, that a set of triples must match, which is similar to the closed-world assumption, while OWL focuses on building complex ontologies, which is similar to the open-world assumption. We infer the types present in the mental model from queries and create SHACL shapes for them, so that each entity in the knowledge graph can be associated with a shape with some probability. Our example query from earlier would result in the following shape graph.

```
<PersonShape> a sh:NodeShape ;
  sh:property [ sh:path :name ] ;
  sh:property [ sh:datatype xsd:string ;
                sh:path :web] ;
  sh:property  [ sh:nodeKind sh:IRI ;
                 rdfs:range <ChildShape> ;
                 sh:path :child] .
<ChildShape> a sh:NodeShape ;
  sh:property [ sh:datatype xsd:integer ;
                sh:path :age] ; .
```

By analyzing further queries and checking with the currently existing triples, we may find that persons and children share almost the same attributes and are therefore related types, such as subclasses. The more queries we have, the more detailed the results will be. From this single query, we already know that being a child has something to do with being associated with a person, suggesting at least two entities involved in a relationship.

2.1. Related Work

This approach is called *query-driven* in contrast to common *data-driven* approaches where semantics are extracted from existing instance data using data profiling methods [4]. The advantage of this approach is that the *mental model* may not be fully present in the data, but is

present in the queries; for example, a query may refer to unmet expectations that a user may have had. The two approaches can complement each other to provide a more complete view of the mental model. Other query-driven approaches are scarce, in [5] they use queries to *discover* data sources satisfying a given data need, while [6] uses queries to *explain* unsatisfactory answers in knowledge bases and *suggest* query modifications, but queries are not yet used for schema inference.

3. Contribution

The previous prototype, Pharos [7], focused on query-driven schema inference based on SQL query logs. We moved away from SQL because there are few openly available query logs that use many non-standard SQL terms, making them difficult to analyze. Since the concept itself is language agnostic, it could also be applied to SPARQL query logs, where more query logs are available and have already been extensively studied [8], perhaps due to the popularity of RDF for publishing open data, while RDBMS are used more for internal purposes.

In addition, there are standardized ways to transform SPARQL queries from their textual form into an RDF graph [9], which allows queries to be treated like graphs, for example by applying graph distance measures to them. Graph representations of queries also enable *meta querying*: using SPARQL queries to search for SPARQL queries within a query repository. Finally, SPARQL is better suited for query-driven schema inference because the information needs in the query are quite explicit, as you are required to list all the properties your desired resource shall have, which basically means giving a (partial) type definition.

We adapted our prototype to parse SPARQL query logs and infer types from them. A type can be thought of as a set of attributes. Subtypes are subsets of their parent attribute sets, while the intersection of two attribute sets could be called a category or type trait. Most SPARQL queries consist of a conjunction of RDF graph patterns that specify conditions that must all be satisfied, effectively describing the subgraph relevant to the query. Some queries also have filters that must be satisfied by the attribute values. These two types of queries, *conjunctive patterns* (CQs) and *conjunctive patterns with filters* (CQFs), are the most common types of queries encountered [8], which is why we focus on them.

Since we know that subject variables in RDF must always be IRIs and thus describe resources, we can partition the attribute sets by subject, as we did in the SHACL shape example in the last section. Each SPARQL subject variable gets a SHACL node form, each property used with that subject becomes a SHACL property. If literals were used as objects, we assign their type as a SHACL data type. If a subject variable is also used as an object variable, we can infer that the query is looking for a relationship between two entities that is likely to be present in the mental model. Therefore, we declare the property in the predicate must point to an IRI, and the scope of this property is the generated shape that was assigned to the object variable.

Applying this procedure to a large set of queries generates a large set of partial types. A major challenge of our approach is to merge all the partial type definitions describing the same type into a unified type definition. We use a density-based clustering algorithm as in [10], but instead of clustering triple instances of the dataset, we apply this approach to the SHACL shapes harvested from the query logs. Similar shapes are merged, which is simply the union of their triple sets. To speed up the process, we use a preprocessing method according to [11]. SHACL shapes generated from similar queries are grouped together because they are more likely to be merged by the clustering algorithm.

The result will not always be complete and depend on the queries being analyzed. If some properties or resources are not used in queries, they will never be discovered. This can be avoided by combining our query-driven approach with the data-driven approaches found in the literature. But even then there will be inconsistencies in the resulting shape graph that need to be corrected either manually, or by weighting conflicting statements by frequency, which means assuming that the majority of users most likely have the more correct concept about some real-world entity. Either way, much more work would be required if the shape graphs were created from scratch, as it would require both domain knowledge and the need to check for inconsistencies in the data itself.

4. Evaluation

The prototype was evaluated in a real-world scenario. The grade management and analysis software *The Grade Explorer* is a web application built on Semantic Web techniques. All data about exams, students, and grades is stored in an RDF knowledge graph managed by Apache Jena, and all data reads and writes are performed using SPARQL queries. *The Grade Explorer* was developed at our chair over several semesters in the context of a practical software development course. The application has had its stable release and is already used in production for various exams.

The students were organized as a scrum team, while chair members acted as product owners or scrum masters. In scrum, developers implement feature requests, called user stories, within an incremental process, delivering a usable product after each iteration. As a side effect, the underlying knowledge graph was being expanded on the

fly, with no semantic control. Over time, nobody had an overview of the meaning of the stored data, and the chaos was exacerbated by the fact that students changed every semester. The mental model used for handling exams, grades and students was never formalized, but hidden in the used SPARQL queries.

It can be difficult to handle exams properly, as it is necessary to carefully follow the official exam regulations and deal with various special cases, like exmatriculated students surprisingly showing up on exam day. The software should take away that burden from the shoulders of the examiners and should do that correctly, so we needed to verify that all data was being stored and interpreted as it should be. We had one student manually create a SHACL shape graph that was manually validated by us, to serve as our ground truth. Then, the modified Pharos prototype analyzed a large query log generated by typical use of *The Grade Explorer* and by running the normal test cases. It created its own SHACL shape graph based on what could be inferred from the queries alone. These two SHACL shape graphs share a Jaccard similarity of 70 %. The Jaccard similarity between two RDF graphs G_1, G_2 of type `sh:NodeShape` with their attribute sets A_1, A_2 is defined as $Jaccard(G_1, G_2) = \frac{|A_1 \cap A_2|}{|A_1 \cup A_2|}$.

The differences can be explained by the fact that some constraints in the manually created shape graph, such as `sh:maxCount` or `sh:minCount`, cannot be guessed from the provided queries or the existing triples, and require domain knowledge, for example that students cannot take an exam twice on the same day, but may be able to try again on a different date.

While the result has some ambiguity, applying the prototype to a real-world example has shown that the approach provides a time-saving benefit whenever it is necessary to reverse-engineer the schema of a knowledge graph, by automating most of the work.

5. Future Work

The approach is depended on the quality of the query logs. Not all queries contain much useful information. If no human-readable variable names were used, it is difficult to assign meaningful names to the generated shapes. However, there are methods to derive variable names from their query context [12] which we will make use of in the future. Queries can be classified into different patterns with varying degrees of usefulness for schema inference; some queries just fetch a single triple with a given IRI, while others contain detailed schema information as in the example above. Queries that were classified unuseful could be discarded beforehand to speed up the process and avoid noise in the results. An interesting phenomenon are *streaks*, a series of queries that are incrementally expanded, representing an exploratory query session by a user [13]. Analyzing these small changes could help better understand user intent, as queries that are corrected likely didn't return the desired results and don't need to be considered for schema inference. Applications can help users reformulate their queries until they meet their needs [14], for example, by using auto-completion or by displaying queries from other users working with the same data sources. Finally, our approach could be leveraged by using a large language model trained on our query logs, which can then be prompted to extract the implicit knowledge hidden in the queries, similar to what has been done in [15].

References

[1] A Semantic Web Primer, Cooperative Information Systems, 3rd ed., MIT Press, Cambridge, 2012.

[2] H. Paulheim, C. Bizer, Type inference on noisy RDF data, in: ISWC, 2013.

[3] C. Floyd, R. Klischewski, Modellierung - ein Handgriff zur Wirklichkeit, in: Modellierung '98, Proceedings des GI-Workshops in Münster, 1998.

[4] T. Papenbrock, et al., Data profiling with Metanome, VLDB J. 8 (2015).

[5] C. Diamantini, et al., A semantic data lake model for analytic query-driven discovery, in: iiWAS, 2022.

[6] L. Parkin, Cooperative techniques for dealing with unsatisfactory answers in RDF knowledge bases, in: VLDB PhD Workshop, 2021.

[7] D. Haller, R. Lenz, Pharos: Query-driven schema inference for the Semantic Web, in: Machine Learning and Knowledge Discovery in Databases, 2020.

[8] A. Bonifati, W. Martens, T. Timm, An analytical study of large SPARQL query logs, VLDB J. 29 (2020).

[9] M. Saleem, et al., LSQ: The linked SPARQL queries dataset, in: ISWC, 2015.

[10] K. Kellou-Menouer, Z. Kedad, Schema discovery in RDF data sources, in: Conceptual Modeling, 2015.

[11] R. Bouhamoum, et al., Scaling up schema discovery for RDF datasets, in: ICDEW, 2018.

[12] B. Ell, et al., Deriving human-readable labels from SPARQL queries, in: SEMANTICS, 2011.

[13] A. Bonifati, W. Martens, T. Timm, SHARQL: Shape analysis of recursive SPARQL queries, in: SIGMOD, 2020.

[14] X. Zhang, et al., Revealing secrets in SPARQL session level, in: ISWC, 2020.

[15] M. Urban, D. D. Nguyen, C. Binnig, OmniscientDB: A large language model-augmented DBMS that knows what other DBMSs do not know, in: International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, 2023.