ViziQuer: A Tool to Explore and Query SPARQL Endpoints

Martins Zviedris and Guntis Barzdins

Institute of Matematics and Computer Science, University of Latvia, Raina bulv. 29, Riga LV-1459, Latvia
Martins.Zviedris@LUMII.lv, Guntis.Barzdins@LUMII.lv

Abstract. The presented tool uses a novel approach to explore and query a SPARQL endpoint. The tool is simple to use as a user needs only to enter an address of a SPARQL endpoint of one's interest. The tool will extract and visualize graphically the data schema of the endpoint. The user will be able to overview the data schema and use it to construct a SPARQL query according to the data schema. The tool can be downloaded from http://viziquer.lumii.lv. There is also additional information and help on how to use it in practice.

Keywords: SPARQL endpoint, Visualization, Query, Ontology.

1 Introduction

SPARQL endpoints take vital role in the Semantic Web as they provide access to actual data for end-users and software agents. Thus, it is important for developers and end-users to understand structure of the underlying data in a SPARQL endpoint to use the available data efficiently. Nowadays a problem arises as there is not enough fundamental work on the SPARQL endpoint schema definition or underlying data documentation. There is ongoing work [1] on an endpoint description, but it is in early development phase and not in actual use.

Until now SPARQL endpoints have been developed just as an access point for SPARQL queries to collect semantically structured data. There is not enough work on the SPARQL endpoint management process to document and overview the underlying data. For example, in SQL like database systems a database programmer can easily extract and view the underlying data schema, thus making much easier to develop a system that works with the database data. It is obvious that for a SPARQL endpoint user it would be also much faster and easier to work with the unknown endpoint data if the user would have more information about the actual data schema (ontology) according to which instance data in the endpoint is organized.

Use of existing query tools [2, 3, 4] is mostly like black box testing of a SPARQL endpoint, because they do not provide overview of a SPARQL endpoint data schema. Existing approaches are mostly based on faceted querying meaning that a user gets overview of only a small part of the actual ontology. The user is like a real explorer without knowledge of what will be encountered in next two steps and in which direction to browse for the needed data. Alternatively, a programmer could construct a SPARQL query by looking at some ontology and hope that a SPARQL endpoint contains the needed data and that correct URI namespaces are used. Note that most SPARQL endpoints nowadays do not provide any ontology or data schema at all as documentation, thus it is quite hard and time-consuming for a programmer to guess their data structure. There exist tools that can show all classes and all properties found in a SPARQL endpoint but not the structure of the underlying data.

Also there exists an approach [5] that can visualize ontologies as diagram graphs to aid better understanding of an ontology, but these applications are standalone and cannot connect to a SPARQL endpoint to view or edit underlying ontology. Unlike [5] we do not intend to show all possible ontology details as, but we rather concentrate on providing a simplified ontology derived from the actual data in a SPARQL endpoint.

2 Overview of the Tool

We have created a tool – ViziQuer[1] that supports a connection to a SPARQL endpoint that contains typed data. By typed data we mean that objects have the class they belong to defined by the relation rdf:type. The tool can be used to explore, understand and query SPARQL semantic database endpoint of ones interest. Thus, the ViziQuer consists of three parts. Firstly, the tool connects and extracts underlying data schema from a SPARQL endpoint. Secondly, the tool allows an end-user to graphically inspect the extracted data schema. Thirdly, an end-user can use the tool to construct a SPARQL query accordingly to the underlying data schema and execute it on the SPARQL endpoint.

The connection process begins by entering a SPARQL endpoint address that an end-user wants to connect and explore. Using predefined sequence of SPARQL queries ViziQuer extracts simple data schema from the SPARQL endpoint. Extraction process can take a while since schema retrieval depends on ontology size and speed of the SPARQL endpoint. We already mentioned that we only support typed data, as it is the basic feature required to extract at least part of the underlying data schema such as classes, their attributes and relations. SPARQL endpoints without rdf:type definitions can be explored only by RDF data browsers, for example, Tabulator [6].

There is no formal limit on data schema size; still we do not recommend trying the DBpedia[2] endpoint and similar ones. Firstly, because we make an assumption that a SPARQL endpoint does not have any limit to answer size, while DBpedia limits every answer to 1000 rows. Secondly, we do not recommend a DBpedia like endpoint as it would be very slow process to extract schema and would not give enough advantage as it would not be easy to explore the endpoint that consists of more than 1000 classes, that is a bit too much for a normal end-user to grasp without a higher ontology structuring, for example [7].

When extracting data schema from the typed underlying data, rather than from the given ontology, one is faced with the lack of definitions for subclass relations between object classes. Without subclass definitions a problem arises, as we need to

[1] http://viziquer.lumii.lv/
[2] http://DBpedia.org/sparql

depict part of relations more than once – as each logical, but not formally defined subclass might have the same outgoing relation present in data resulting in explosion of duplicate relations as one can grasp in the view of naïve schema visualization in Fig. 1 where depicted is schema visualization result of the Semantic Web Dog Food endpoint[3]. We use UML like graphical notation similar to one proposed in [5].

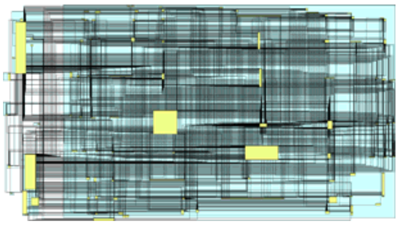

Fig. 1. The Semantic Web Dog Food endpoint full schema

As one can easily see in Fig 1, the naïve data schema visualization is opaque because subclass relations are not defined and each relation is drawn multiple times. To make it more comprehensive currently we use a hack and display each relation only once (see Fig.2) even if the relation actually occurs between more than one pair of classes. Thus, we get a comprehensive picture that is not fully semantically correct, but understandable for an end-user. In the third step when an end-user composes a SPARQL query based on the extracted schema all relation occurrences are taken into account. A semantically correct way to solve the duplicate relations problem would be to allow end-user to manually define missing subclass relations (based on meaningful names of classes) and then automatically "lift" duplicate relations to the most abstract superclass and thus making each relation to appear only once.

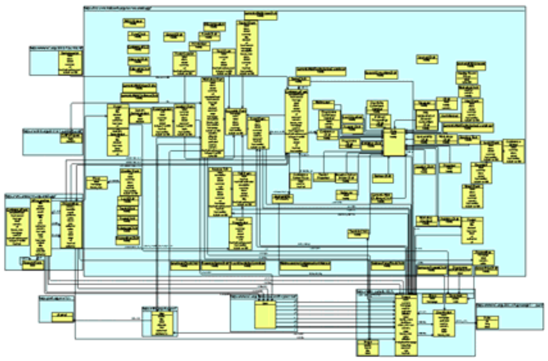

Fig. 2. The Semantic Web Dog Food with namespace and limited associations

Last important feature of the VíziQuer tool is ability to semi-automatically construct SPARQL queries according to the extracted data schema (ontology). In the

[3] http://data.semanticweb.org/sparql

ViziQuer we have implemented a subset of the GQL graphic query language [8] that provides basic features for selection of a subset of data. As queries are based on the ontology then querying is more like constructing or drawing a subset of the ontology corresponding to data that an end-user is interested in for further analysis.

A query construction is possible by two paradigms. First, a user can construct a query by selecting one class and browsing further like in faceted browsing where a user selects a possible way further from already selected classes. Second, a user can use a connection-based query construction. This means that a user selects two classes of interest by adding them to a query diagram and by drawing a line between them indicates that both classes should be interconnected. The tool uses an advanced algorithm to propose suitable paths how these two classes can be interconnected and a user just needs to select a path that best fits to desired path. Thus, when a user wants to connect some classes that are a bit further one from another, a selection of an acceptable path between classes is needed rather than a guess by manual browsing that can be quite hard in the faceted browsing paradigm if a user is not very well familiar with the ontology structure. We should also mention that both paradigms could be used in parallel when one constructs a SPARQL query.

We will briefly explain the GQL by an example query. Main idea in the GQL is selection of an interconnected subset of classes, attributes, and associations that at some point can be viewed as an ontology subset. Additionally it is possible to restrict the subset by some basic filters on class attributes. In Fig. 3 is depicted the example constructed for the Semantic Web Dog Food endpoint.

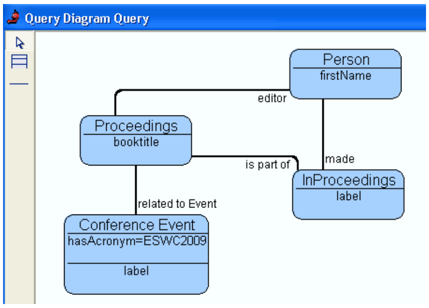

Fig. 3. Example query of the GQL based on the Semantic Web Dog Food endpoint

The query formulated in Fig. 3 could be rephrased as to “select those authors that have edited some proceedings and also have some paper in edited proceedings that are related to some conference event”. We add restriction that conference acronym is ESWC2009. We set the answer to contain a persons first name, a name of published paper, a year when proceedings where published and also a name of the conference. For limited space reasons we do not show this query translated into SPARQL.

The ViziQuer implementation is based on the Model Driven Architecture technologies that allow it to be very flexible and to connect to any SPARQL endpoint. We used the GrTP platform [9] as environment for the tool development. GrTP allows to easy manipulating graphical diagrams that is most needed to construct graphical queries and visualize a SPARQL endpoint underlying data schema. Main drawback for the platform is that it supports only the Windows operating systems, thus the ViziQuer currently works only in the Windows operating system environment.

3 Results and Conclusions

We have developed a tool that simplifies work with unknown SPARQL endpoints. The tool allows an end-user not just to build SPARQL queries, but also to get an overview of the underlying data. Still, as quality of formal ontology definitions in SPARQL endpoints is often incomplete further improvements must be made in schema management, for example, ability to manually add subclass relations between classes. Also it would be wise to consider a case when data is logically typed, but formal type definitions are missing.

Acknowledgments

This work has been partly supported by the European Social Fund within the project «Support for Doctoral Studies at University of Latvia».

References

1. SPARQL endpoint description, `http://esw.w3.org/SparqlEndpointDescription`
2. Heim, P., Ertl, T., Ziegler, J.: Facet Graphs: Complex Semantic Querying Made Easy. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010. LNCS, vol. 6088, pp. 288–302. Springer, Heidelberg (2010)
3. Longwell RDF Browser, SIMILE (2005), `http://simile.mit.edu/longwell/`
4. List of SPARQL faceted browsers, `http://esw.w3.org/SemanticWebTools#Special_Browsers`
5. Barzdins, J., Barzdins, G., Cerans, K., Liepins, R., Sprogis, A.: OWLGrEd: a UML Style Graphical Notation and Editor for OWL. In: Clark, K., Sirin, E. (eds.) Proc. 7th International Workshop OWL: Experience and Directions, OWLED-2010 (2010)
6. Berners-Lee, T., Hollenbach, J., Lu, K., Presbrey, J., Prud'ommeaux, E., Schraefel, M.C.: Tabulator Redux: Browsing and writing Linked Data. In: Proc. WWW 2008 Workshop: LDOW (2008)
7. Zviedris, M.: Ontology repository for User Interaction. In: d'Aquin, M., Castro, A.G., Lange, C., Viljanen, K. (eds.) ORES-2010 Workshop on Ontology Repositories and Editiors for the Semantic Web, CEUR (2010), `http://CEUR-WS.org/Vol-596/`
8. Barzdins, G., Rikacovs, R., Zviedris, M.: Graphical Query Language as SPARQL Frontend. In: Grundspenkis, J., Kirikova, M., Manolopoulos, Y., Morzy, T., Novickis, L., Vossen, G. (eds.) Local Proceedings of 13th East-European Conference (ADBIS 2009), pp. 93–107. Riga Technical University, Riga (2009)
9. Barzdins, J., Zarins, A., Cerans, K., Kalnins, A., Rencis, E., Lace, L., Liepins, R., Sprogis, A.: GrTP:Transformation Based Graphical Tool Building Platform. In: Proceedings of the MoDELS 2007 Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI-2007), CEUR Workshop Proceedings, vol. 297 (2007)