ViziQuer: A Web-Based Tool for Visual Diagrammatic Queries Over RDF Data

Kārlis Čerāns[1,2(✉)], Agris Šostaks[1,2], Uldis Bojārs[1,2], Jūlija Ovčiņņikova[1,2], Lelde Lāce[1,2], Mikus Grasmanis[1], Aiga Romāne[1], Artūrs Sproģis[1], and Juris Bārzdiņš[3]

[1] Institute of Mathematics and Computer Science, University of Latvia, Riga, Latvia
karlis.cerans@lumii.lv
[2] Department of Computing, University of Latvia, Riga, Latvia
[3] Department of Medicine, University of Latvia, Riga, Latvia

Abstract. We demonstrate the open source ViziQuer tool for web-based creation and execution of visual diagrammatic queries over RDF/SPARQL data. The tool supports the data instance level and statistics queries, providing visual counterparts for most of SPARQL 1.1 select query constructs, including aggregation and subqueries. A query environment can be created over a user-supplied SPARQL endpoint with known data schema (a data schema exploration service is available, as well). There are pre-defined demonstration query environments for a mini-university data set, a fragment of synthetic similar to reality hospital data set, and a variant of Linked Movie Database RDF data set.

Keywords: Visual query tool · Ad-hoc queries · Rich queries
RDF data · SPARQL

1 Introduction

The textual SPARQL 1.1 [1] select query language over RDF data allows creating rich data selection queries, possibly involving subqueries, aggregations, unions and rich expression notation. The visual/diagrammatic environments for query creation support, such as Optique VQs [2], Query VOWL [3] and early versions of ViziQuer [4], however, stay significantly behind the expressivity of SPARQL 1.1 select queries by not supporting e.g. the subqueries and aggregation. The possibility of introducing aggregated fields and rich expression notation into a diagrammatic UML-style RDF data query environment has been shown in the earlier work of authors [5, 6], while [7] provides a more refined set of extended UML class diagram constructs for visual SPARQL query definition including: (i) separation of aggregated and grouping fields in query node attribute lists; (ii) visual notation for subqueries, (iii) separate query control nodes for query structuring and (iv) integrated textual SPARQL fragments.

We describe and demonstrate here the web-based open source ViziQuer tool supporting the notation of [7] and providing basic user services both for query environment configuration and visual query creation. The resource point http://viziquer.lumii.lv provides links both to the online tool and the source code repository.

A. Gangemi et al. (Eds.): ESWC 2018 Satellite Events, LNCS 11155, pp. 158–163, 2018.
https://doi.org/10.1007/978-3-319-98192-5_30

We provide preliminary results on query notation and tool usability, as well.

In what follows, Sect. 2 briefly reviews the visual query notation, Sect. 3 describes the query tool usage and implementation and Sect. 4 concludes the paper.

2 Visual Notation Overview

For query notation illustration we use a mini-hospital data schema, extracted from [8] and depicted in UML style notation in Fig. 1. The role names, if not specified, coincide with target class names with lowercase first letter; the attributes and roles are assumed by default to have minimum and maximum cardinalities 1.

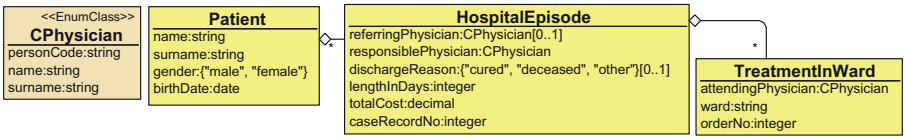

Fig. 1. Mini-hospital data schema

A basic visual query (cf. [4, 7]) is a UML class diagram style graph with the nodes describing data instances, the edges describing their connections and the attributes forming the query selection list from the node instance attributes and their expressions; every node can specify both the instance class and additional conditions on the instance. One of the graph nodes is the main query node (shown as orange round rectangle in the diagram); the structural edges (all edges except the condition ones) within the graph form its spanning tree with the main query node being its root.

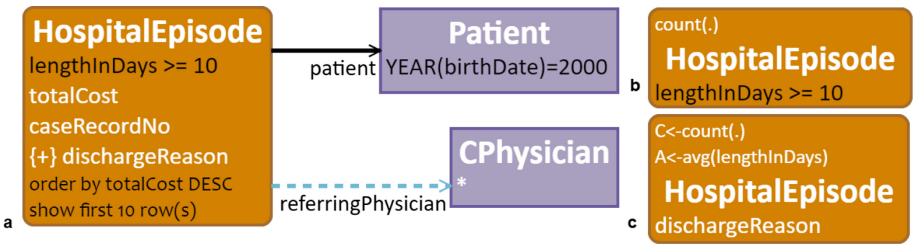

Fig. 2. Class-attribute-link-condition and statistics queries

Figure 2 shows initial query examples following the basic class-attribute-link-condition paradigm (a), similar to [2, 4], and simple statistics queries (b) and (c), as in [7]:

(a) *select top 10 most expensive hospital episodes with discharge reason specified, lasting for at least 10 days, for patients born in 2000; show episode total cost, case record number, discharge reason and all attributes of referring physician, if specified;*

(b) *count the hospital episodes lasting for at least 10 days and*
(c) *compute the hospital episode count and average length in days, grouped by the episode discharge reason.*

The visual notation supports links that are required, optional and negated. The attributes in node fields by default are optional, not to bypass entire solution rows because of missing attribute values, an attribute is marked as required by a {+} decoration in the visual notation is (cf. *dischargeReason* attribute in Fig. 2 (a)).

Figure 3 demonstrates more advanced query examples using subqueries (edges with black bullets at the end) (a), control nodes ([] stands for an outer query scope for collecting, filtering, projecting, further aggregating of subquery result lists) and non-model links (denoted by ++ in (b)), as well as schema-level variables (c):

(a) *count the patients with at least 3 hospital episodes having at least 5 wards each;*
(b) *count all wards with more than 1000 treatment in ward cases, and*
(c) *select all data classes together with their instance count.*

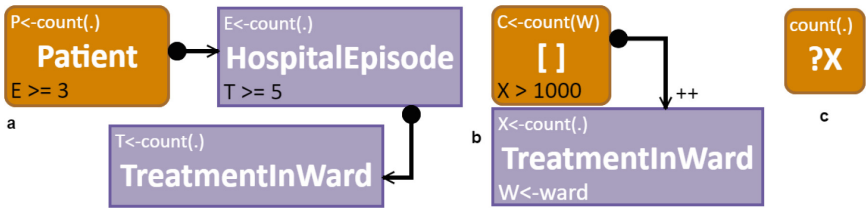

Fig. 3. Advanced query constructs

For more details and further notation examples one may consult [7], as well as the demonstration examples provided within the query environment.

3 Query Tool Usage and Implementation

The user's work with the ViziQuer tool is arranged in projects. Users can create projects that consist of query diagrams each capable of hosting multiple queries. Each project needs a supplied data schema and a SPARQL endpoint (pre-configured ViziQuer instances with built in schema and endpoint information are possible, as well). There are prototype services for ViziQuer schema extraction from an OWL ontology and from SPARQL endpoint data.

A ViziQuer project needs also a SPARQL engine type to enable query translation optimizations for vendor-specific SPARQL endpoints. The practical tool usage up to now has been oriented towards OpenLink Virtuoso SPARQL endpoints, although a "General SPARQL" option is available, as well.

Each query diagram allows for new query creation, starting from a query symbol selection from the symbol palette (cf. Fig. 4), followed by class name condition, plain and aggregate attribute setting using the property dialogues (there are basic suggestion

services for class and attribute names). Adding of another class into the query can be performed by introducing the class via symbol palette, or by using the offered "Add Link" service (cf. Fig. 4) offering the link names and their target classes making sense in the context of the selected host class. The defined queries in the diagrams (either the connected components or parts thereof) can be either translated into the textual SPARQL form, or directly executed over the specified SPARQL endpoint.

There are available demonstration query environments for a mini-university data set, a fragment of synthetic hospital data set resembling the data of Children's hospital in Riga, Latvia [8] and a variant of Linked Movie Database RDF data set [9].

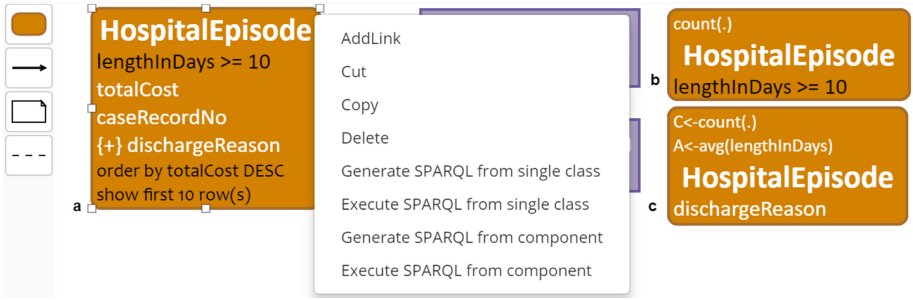

Fig. 4. Basic working environment

The tool is created using *ajoo* – a generic platform for web-based diagrammatic tool building [10]. The ViziQuer tool is defined within the platform by creating a JSON-style configuration for the node, edge and compartment types in the diagrams, and writing JavaScript functions for the tool specific functionality, including query translation into SPARQL. Ajoo and ViziQuer use Meteor framework [11] and MongoDB for diagram storage and exchange, and for user management and collaboration features.

The ajoo framework and ViziQuer query tool are open source and have been made available on GitHub, so enabling local installations of the ViziQuer query engine.

The tool architecture allows for de-coupling of the ajoo platform and custom diagram handling code from the Meteor server, should there a need arise for integrating it into some other diagram serving infrastructure.

4 Conclusions

The presented ViziQuer tool demonstrates the feasibility of visual querying of the RDF data just from a web browser window. The supported query notation allows visual creation of most of the SPARQL select query language constructs, including aggregates, subqueries and query structuring [7]. The initial experiments with potential tool end users without a formal IT background have indicated that this user group could well read and understand e.g. the queries involving subquery construct formerly considered to be out of scope for visual query formulation systems (cf. e.g. [2]).

A preliminary query composition experiment with undergraduate 4th year IT students at University of Latvia has indicated a potential usability of the visual notation and tool for generally IT literate persons. The 14 student participants (with some background in SQL and no specific training in RDF/SPARQL) were given brief introductions about the hospital data model, RDF, SPARQL and ViziQuer (each about 10 min) and then were split into two groups of 7 and given 10 query writing tasks. One group was asked to create queries in ViziQuer and the other – using textual SPARQL notation. After the training period students had 60 min to work on the assigned tasks. The numbers of successfully completed tasks by the students in the visual notation group were 10, 5, 5, 4, 4, 3, 2 (in average 4.7), while in the SPARQL group there were 5, 5, 3, 3, 2, 2, 1 completed tasks (in average 3.0). The ViziQuer group outperformed the SPARQL group on all query subsets of data instance queries, simple aggregate queries and queries with subquery structure.

It can be expected that the tool interface re-work that is to be provided as a future work (the property sheets with very basic name suggestion services currently available) would make the notation and tool a potential alternative or complement to other SPARQL query creation approaches both for semantic technology experts and lay users. This future work involves also more definite usability tests that are to be run.

Acknowledgements. This work has been partially supported by research organization base financing at Institute of Mathematics and Computer Science, University of Latvia and the University of Latvia project AAP2016/B032 “Innovative information technologies”.

References

1. SPARQL 1.1 Query Language. W3C Recommendation 21 March 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/
2. Soylu, A., Giese, M., Jimenez-Ruiz, E., Vega-Gorgojo, G., Horrocks, I.: Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users. Univ. Access Inf. Soc. **15**(1), 129–152 (2016)
3. Haag, F., Lohmann, S., Siek, S., Ertl, T.: QueryVOWL: visual composition of SPARQL queries. In: Gandon, F., Guéret, C., Villata, S., Breslin, J., Faron-Zucker, C., Zimmermann, A. (eds.) ESWC 2015. LNCS, vol. 9341, pp. 62–66. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25639-9_12
4. Zviedris, M., Barzdins, G.: ViziQuer: a tool to explore and query SPARQL endpoints. In: Antoniou, G., et al. (eds.) ESWC 2011. LNCS, vol. 6644, pp. 441–445. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21064-8_31
5. Cerans, K., Ovcinnikova, J., Zviedris, M.: SPARQL aggregate queries made easy with diagrammatic query language ViziQuer. In: Proceedings of the ISWC 2015 Posters & Demonstrations Track, CEUR, vol. 1486 (2015). http://ceur-ws.org/Vol-1486/paper_68.pdf
6. Čerāns, K., Ovčiņņikova, J.: ViziQuer: notation and tool for data analysis SPARQL queries. In: Proceedings of VOILA 2016, Kobe, Japan, CEUR Workshop Proceedings, CEUR-WS.org, vol. 1704, pp. 151–159 (2016). http://ceur-ws.org/Vol-1704/paper15.pdf
7. Cerans, K., et al.: Extended UML class diagram constructs for visual SPARQL queries in ViziQuer/web. In: Voila!2017, CEUR Workshop Proceedings, vol. 1947, pp. 87–98 (2017)

8. Barzdins, J., Grasmanis, M., Rencis, E., Sostaks, A., Barzdins, J.: Ad-Hoc querying of semistar data ontologies using controlled natural language. In: Frontiers of AI and Applications, Databases and Information Systems IX, vol. 291, pp. 3–16. IOS Press (2016). http://ebooks.iospress.com/volumearticle/45695
9. Linked Movie Database. http://www.cs.toronto.edu/∼oktie/linkedmdb/
10. Sprogis, A.: ajoo: WEB based framework for domain specific modeling tools. In: Frontiers of AI and Applications, Databases and Information Systems IX, vol. 291, pp. 115–126. IOS Press (2016). http://ebooks.iospress.com/volumearticle/45704
11. Strack, I.: Getting Started with Meteor JavaScript Framework. Packt Publishing Ltd, Birmingham (2012)