Integrating Sparklis and ViziQuer for Enhanced SPARQL Querying and Visualization

Uldis Bojārs, Jūlija Ovčiņņikova, Lelde Lāce, Artūrs Sproģis, Mikus Grasmanis and Kārlis Čerāns

Institute of Mathematics and Computer Science, University of Latvia, Riga, Latvia

Abstract

This paper introduces a multimodal SPARQL query system that combines the capabilities of Sparklis and ViziQuer, two powerful tools for SPARQL query building and visualization. Sparklis offers a faceted user interface for constructing SPARQL queries, while ViziQuer provides a rich visual interface for constructing and visualizing SPARQL queries. By integrating the two applications, the system facilitates the automatic visualization of SPARQL queries constructed in Sparklis within the ViziQuer visual environment.

Keywords

RDF, SPARQL, query visualization, Sparklis, ViziQuer

1. Introduction

SPARQL queries can be difficult to write by non-technical users [1]. There are various SPARQL query building assistants that can help with this task, including tools using form-based interfaces (e.g. PepeSearch [2] and WYSIWYQ [3]), controlled natural language snippets (e.g. Sparklis [4]), and visual diagrams (cf. e.g. [1,5,6,7]).

Each of the notations have their strengths and weaknesses, especially when it comes to the design of rich SPARQL queries (involving, e.g., aggregation and subqueries). For instance, the Sparklis notation allows for rich query composition using the natural language snippets that can be expected to be suitable for a less technical end-user, while the created query formulations may still appear difficult to understand by some end-users. On the other hand, ViziQuer may provide a structural overview and further refinement of the query.

Both Sparklis and ViziQuer provide means for rich SPARQL query definition including complex expressions, grouping and aggregation. The ViziQuer functionality for visualizing SPARQL queries [8] opens the way for combining the strengths of both query building methods into a multi-modal query creation environment, where the query can be initially built in Sparklis and then translated into ViziQuer (from the SPARQL query form that is provided by Sparklis).

This paper presents a prototype of such a multi-modal system that integrates Sparklis and ViziQuer tools[2] and allows users to visualize SPARQL queries created in Sparklis. The integrated system combines the benefits of the two tools: users can use the faceted UI and controlled natural language approach of Sparklis, and can visualize and refine these SPARQL queries in the ViziQuer visual query environment. These visualizations are based on the UML-like notation implemented in ViziQuer and ViziQuer's query visualization functionality [7,8]. Apart from the particular integration of two SPARQL query composition assistants, SPARKLIS and ViziQuer, this paper establishes the concept of integration of several such assistants into a single multi-modal environment.

Proceedings Acronym: Proceedings Name, Month XX–XX, YYYY, City, Country
uldis.bojars@lumii.lv (U. Bojārs); karlis.cerans@lumii.lv (K. Čerāns)
0000-0001-7444-565X (U. Bojārs); 0000-0002-0154-5294 (K. Čerāns)

[2] http://viziquer.lumii.lv/sparklis/

The rest of the paper consists of main information about Sparklis (Section 2) and ViziQuer (Section 3), followed by a description of the integration of these tools (Section 4) and examples of SPARQL queries in both notations (Section 5). The paper is completed by a summary of related work (Section 6), and conclusions and future work (Section 7).

2. Sparklis

Sparklis[3] is an open source SPARQL query builder tool that uses controlled natural language and a faceted search user interface, and allows people to explore and query SPARQL endpoints without knowledge of SPARQL or the vocabulary used by a particular SPARQL endpoint [4]. It is a web application that runs entirely in the browser.

Sparklis covers a large subset of SPARQL 1.1 SELECT queries: basic graph patterns (BGP) including cycles, UNION, OPTIONAL, NOT EXISTS, FILTER, BIND, complex expressions, aggregations, GROUP BY and ORDER BY. Its configuration panel offers options to adapt to different endpoints. Sparklis also includes the YASGUI editor to let advanced users access and modify the SPARQL translation of the query. We extend the Sparklis functionality by letting users also access and modify the visual representation of the SPARQL query.

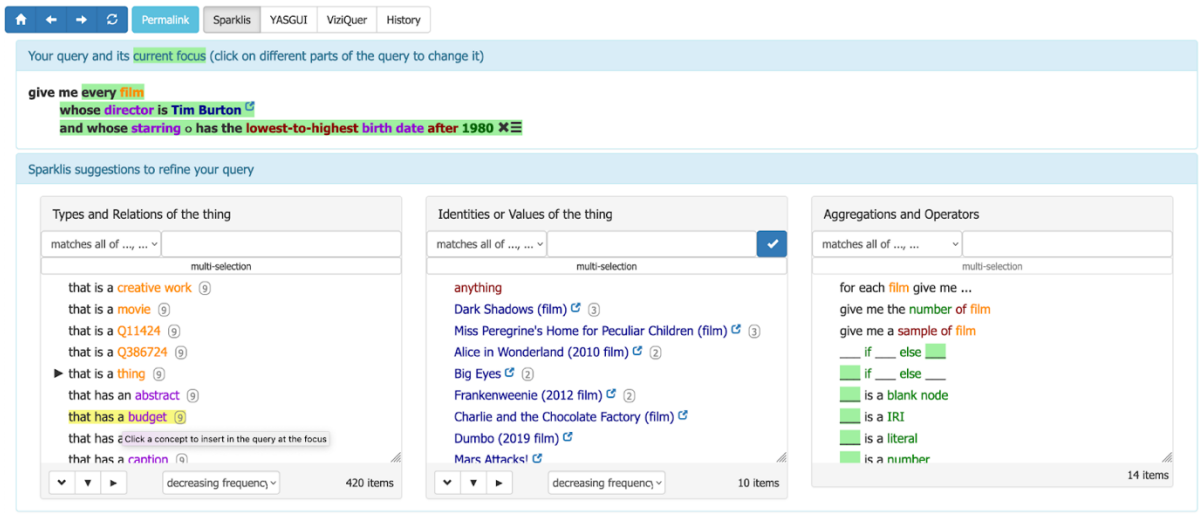

Figure 1: Sparklis query example

The query builder functionality lets Sparklis users incrementally build complex queries by combining elementary queries. Elementary queries can be a class (e.g. "a film"), a property (e.g. "that has a director"), an RDF node (e.g. "Tim Burton"), a reference to another node (e.g. "the film"), or an operator (e.g. "average"). Sparklis queries are verbalized in controlled natural language, hiding the SPARQL queries generated by this tool from the user. Figure 1 shows an example of a Sparklis query for films by Tim Burton that star somebody born after 1980. Additional query examples can be found on the Sparklis website[4].

3. ViziQuer

ViziQuer[5] is an open source UML-style visual SPARQL query tool that allows users to define SPARQL queries visually using the ViziQuer visual notation. The ViziQuer notation and environment (cf. [7,9]) provides visual means for rich query definition, involving BGPs, value filters, optional and negated constructs, as well as unions, aggregation, grouping and subqueries,

[3] https://github.com/sebferre/sparklis
[4] http://www.irisa.fr/LIS/ferre/sparklis/examples.html
[5] https://viziquer.lumii.lv/

coming close to the full SPARQL 1.1 SELECT query visualization [8]. ViziQuer source code is available on Github[6].

ViziQuer has been extended with a query visualization functionality that allows users to transform SPARQL queries into their visual representation. The visualization of SPARQL SELECT queries produces a visual extended UML-style diagram that describes the entire query contents. For a simple query consisting of the graph patterns, the pattern subject and object variables and resources are depicted as the query graph nodes, with important optimizations for representing the variable/resource classes in the dedicated class name compartments and single-use SPARQL triple objects within node attribute fields, so obtaining a compact query presentation [7].

The visualization of a SPARQL query could originally be achieved by copying the SPARQL query text into ViziQuer's SPARQL query pane and activating a "Visualize SPARQL" context menu item. Before doing this, users also needed to log in to the ViziQuer tool. In order to perform automatic integration of ViziQuer with external tools such as Sparklis, ViziQuer was extended in order to allow anyone to work with SPARQL queries (incl. visualizing queries) without logging in.

4. Implementation

The integration of Sparklis and ViziQuer was realized by creating a Sparklis plugin (viziquer.js) that retrieves the SPARQL query and associated information from Sparklis and sends it to the ViziQuer API. Sparklis source code was augmented in order to add the ViziQuer tab to the Sparklis UI. The user interface allows ViziQuer to be launched either in the same Sparklis screen (button "Show here") or in a new browser tab (button "Show in a new tab"). The modified version of Sparklis and its ViziQuer plugin are available on Github[7].

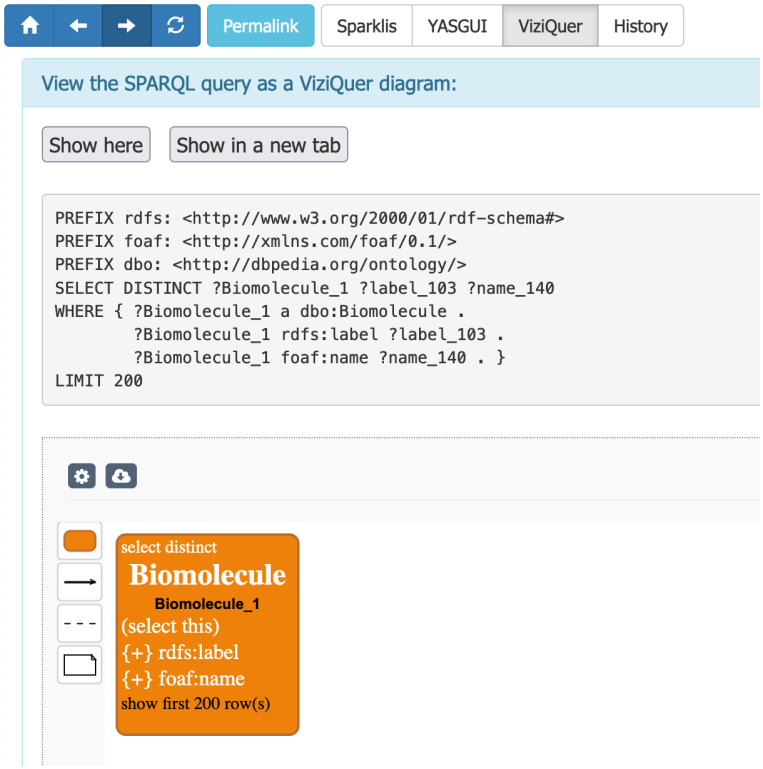

Figure 2: ViziQuer section in the Sparklis query builder

[6] https://github.com/LUMII-Syslab/viziquer/tree/development
[7] https://github.com/LUMII-Syslab/sparklis

The required ViziQuer API call can be invoked as an HTTP POST request to the API endpoint provided by the public ViziQuer service[8]. Alternatively, users can launch their own instances of ViziQuer. The parameters of this API call include the type of the query (SPARQL), the SPARQL endpoint URI and the SPARQL query to be visualized along with a flag that indicates if the visualization action should be started automatically (the other alternative is to just send the SPARQL query to ViziQuer and let users launch the visualization action manually). Figure 3 shows the parameters passed to ViziQuer's JSON API call.

```
const json_data = JSON.stringify({
    "query": sparklis.currentPlace().sparql(),
    "endpoint": sparklis.endpoint(),
    "queryType": "SPARQL",
    "isVisualizationNeeded": VQ_VISUALIZATION
})
```

Figure 3: Parameters for the ViziQuer API call

In a similar way, the ViziQuer tool can be integrated into other querying environments, e.g., into a YASGUI based frontend of a SPARQL endpoint.

Query visualization in ViziQuer works best if ViziQuer is "aware" of the data schema of the given SPARQL endpoint but it is also possible to visualize SPARQL queries for endpoints for which ViziQuer does not have a data schema available.

5. Results

Through the integration of Sparklis and Viziquer, users can visualize SPARQL queries constructed in Sparklis. This section provides some examples of queries expressed in the controlled natural language of Sparklis [4] along with the corresponding SPARQL queries and their visualization in ViziQuer.

UML-style visual queries in ViziQuer notation consist of nodes describing variables or resources where each node can have a possible class name and attribute specification. One of the nodes is marked as the main node of a query (orange round rectangle). The edges that connect the nodes correspond to links among the query variables or resources. Furthermore, condition/filter fields, as well as aggregation and query nesting links can be used in query construction [7,9].

ViziQuer allows users to edit and fine-tune the visual representation of SPARQL queries. We used this functionality to manually tune the presentation of visualizations shown in this section.

Figure 4 shows a simple query for information about biomolecule class instances from DBpedia in Sparklis and ViziQuer representation. The following SPARQL query is represented by Sparklis and ViziQuer notations in this example:

```
SELECT DISTINCT ?Biomolecule_1 ?label_103 ?name_140
WHERE { ?Biomolecule_1 a dbo:Biomolecule .
        ?Biomolecule_1 rdfs:label ?label_103 .
        ?Biomolecule_1 foaf:name ?name_140 . }
LIMIT 200
```

[8] https://viziquer.app/api/public-diagram

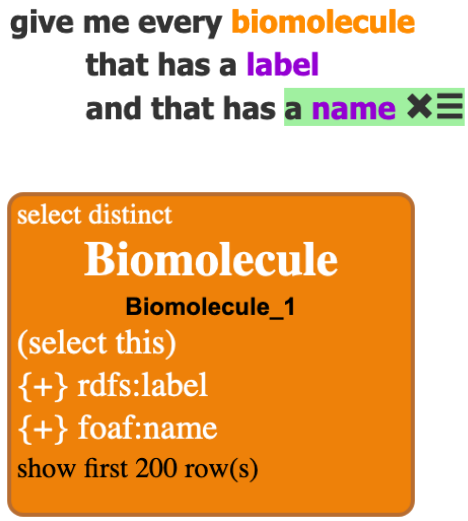

Figure 4: SPARQL query for biomolecule information in Sparklis and in ViziQuer

We observe that the ViziQuer representation of the query matches the structure of the original Sparklis query, explicating at the same time some of the assumptions regarding property and variable names and the query limit that are left implicit in the original Sparklis notation.

Figure 5 illustrates a query that uses aggregation to calculate the number of languages spoken in Colombia.

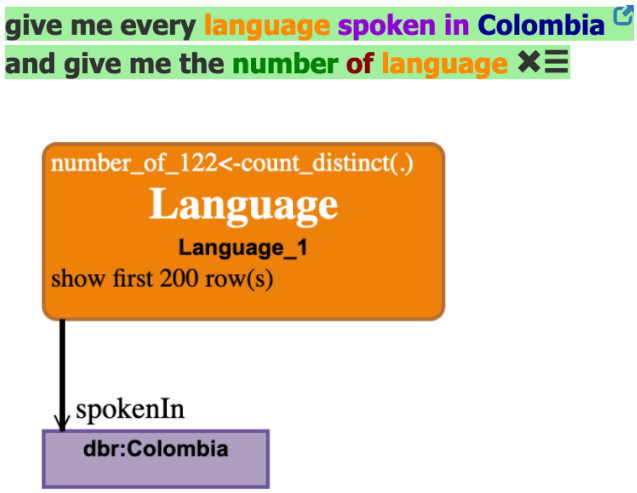

Figure 5: SPARQL query using aggregation in Sparklis and in ViziQuer

The visual query representation distinguishes the nodes corresponding to the language and the country. The full visual appearance of the query is somewhat overburdened by the explicit variable names (*Language_1* and *number_of_122*) introduced by the Sparklis tool in the generated SPARQL query:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT (COUNT(DISTINCT ?Language_1) AS ?number_of_122)
WHERE { ?Language_1 a dbo:Language .
        ?Language_1 dbo:spokenIn dbr:Colombia . }
LIMIT 200
```

Figure 6 illustrates a more complex query that uses aggregation to calculate the number of books written by poets and returns the results ordered in decreasing order of aggregated values. It shows the original Sparklis representation and four options how the corresponding query can be represented in ViziQuer. Option (a) is auto-generated from the SPARQL query produced by Sparklis, while option (b) is a cleaned-up form of it. The options (c) and (d) were created manually by placing the Person class at the main node of the query. Option (c) uses the joined classes construction, while option (d) uses a subquery to calculate the number of books authored by a person (option (d) would translate to a different SPARQL query form).

It might be an interesting future work to understand which of the query representations (b), (c) or (d) would be the easiest-to-understand by various groups of potential end users. This ease of perception may depend on users mastering the concepts of aggregation with grouping (option (b)), instance aliases (option (c)) and subqueries (option (d)). Each of these representations provides an explicit query structure of nodes linked by properties that was not present in the initial Sparklis formulation. Given the obtained understanding, further work would be to implement an automated creation of the desired visual query form.

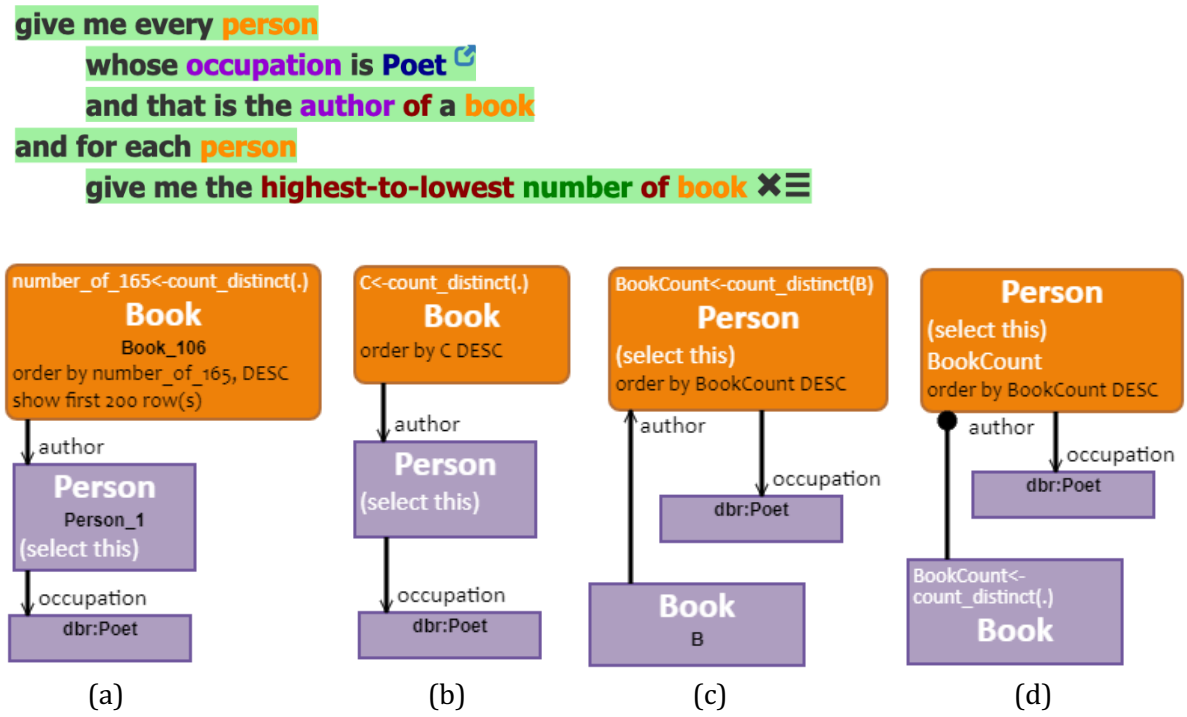

Figure 6: Options for visualizing an aggregated SPARQL query

The following SPARQL query corresponds to the Sparklis and ViziQuer query representation in Figure 6 (a):

```
SELECT DISTINCT ?Person_1 (COUNT(DISTINCT ?Book_106) AS ?number_of_165)
WHERE { ?Person_1 a dbo:Person .
      ?Person_1 dbo:occupation dbr:Poet .
      ?Book_106 a dbo:Book .
      ?Book_106 dbo:author ?Person_1 . }
GROUP BY ?Person_1
ORDER BY DESC(?number_of_165)
LIMIT 200
```

6. Related Work

The visual presentation of information can facilitate its perception. Facet-based tools such as PepeSearch [2] and WYSIWYQ [3] aim to make it easier to create SPARQL queries by using data forms and facets. Visual query composition tools allow users to define the query visually and can be classified into tools that display attribute values in separate graph nodes and tools that use a UML-style notation that offers us a more compact query representation. Examples of the former group are QueryVOWL [6], RDF Explorer [10] and GRUFF [11] which support just the simplest forms of conjunctive SPARQL queries. FedViz [13] lets users visually select query classes and properties, yet it does not provide a full visual representation of the query. FedViz and SPARQLGraph [12] differ from other tools in that they allow users to build federated queries. In the UML-style group, Optique VQS [1] and LinDA [5] also support outer-level aggregation. Compared to ViziQuer, the visual query constructs in Optique VQS are more limited, as they do not support subqueries, or the optional and negation modalities of join queries. There is also a more limited expression language and expressions are not shown explicitly in the Optique VQS visual query presentation.

Compared to these tools, the ViziQuer notation and environment (cf. [7,8,9]) provides visual means for rich query definition, including basic graph patterns, value filters, optional and negated constructs, as well as unions, aggregation, grouping and subqueries, coming close to the full SPARQL 1.1 SELECT query visualization [8]. Sparklis, while using a different approach – controlled natural language and faceted exploration – also supports a large subset of SPARQL 1.1 SELECT queries (OPTIONAL, NOT EXISTS, FILTER, BIND, etc.) [4].

The integration of Sparklis and ViziQuer was made possible by the extension mechanism in Sparklis and the ViziQuer visualization API. While there are other tools available for defining SPARQL queries, after exploring related work we did not find other instances of integrating SPARQL query tools in the way described in this paper.

7. Conclusions and Future Work

In this paper we presented the integration of two powerful tools for defining SPARQL queries – Sparklis and ViziQuer – resulting in a multi-modal system that allows users to define SPARQL queries in Sparklis and visualize and refine them in ViziQuer. We described the implementation details of this integration and demonstrated it with SPARQL queries in their Sparklis and ViziQuer representation. The obtained results show that the visual query representation that is auto-generated from the technical SPARQL query requires some cleaning up to improve its usefulness for the potential end-users. Implementation of such cleaning is left to future work. It may also be worthwhile to consider making use of higher-level Sparklis concepts in the visual query generation. While it is essential for the ViziQuer tool to maintain the principle of generating the visual queries from their SPARQL encoding, incorporating some form of annotations into this encoding could let us preserve the principal query generation architecture and still reach the necessary effects of user-friendliness.

The core of the Sparklis and ViziQuer integration described in this paper is the ability of the ViziQuer tool to create a visual representation of a given SPARQL query. This feature also allows integrating the SPARQL query visualization functionality in other contexts where SPARQL queries are available. The technical solution of invoking ViziQuer without logging in, described in Section 3, can be used to support such visualizations. It could also be useful in simpler use cases where a visual query environment can be seamlessly offered to the users for visually composing queries over a given SPARQL endpoint.

Other potential areas for future exploration are the integration of ViziQuer and Sparklis in the opposite direction (from ViziQuer visual notation to Sparklis queries), which would require a method for reverse translation of SPARQL queries into their Sparklis representation, as well as the integration of the ViziQuer SPARQL visualization functionality with other Semantic Web tools.

Acknowledgements

This work has been partially supported by a Latvian Science Council Grant lzp-2021/1-0389 "Visual Queries in Distributed Knowledge Graphs".

References

[1] Soylu A., Kharlamov, E., Zheleznyakov, D., Jimenez Ruiz, E., Giese M., Skjaeveland, M.G., Hovland, D., Schlatte, R., Brandt, S., Lie, H., Horrocks, I. (2018). OptiqueVQS: a Visual Query System over Ontologies for Industry, Semantic Web 9(5), 627-660, IOS Press.

[2] Vega-Gorgojo, G., Giese, M., Heggestøyl, S., Soylu, A., Waaler, A. (2016). PepeSearch: semantic data for the masses. PloS one, 11(3), e0151573. https://doi.org/10.1371/journal.pone.0151573 0151573

[3] Khalili, A., Merono-Penuela, A. (2017). WYSIWYQ–What You See Is What You Query, Proceedings of Voila 2017, CEUR Workshop, 1947, 123-130.

[4] Ferré, S. (2017). Sparklis: An Expressive Query Builder for SPARQL Endpoints with Guidance in Natural Language. Semantic Web 8(3): 405-418. IOS Press.

[5] The LinDA Query Designer, available at https://2015.semantics.cc/sites/2015.semantics.cc/files/files/LinDAQueryDesigner.pdf

[6] Haag, F., Lohmann, S., Siek, S., Ertl, T. (2015). QueryVOWL: A Visual Query Notation for Linked Data, in Gandon, F., Guéret, C., Villata, S., Breslin, J., Faron-Zucker, C., Zimmermann, A. (ed.), Proceedings of The Semantic Web: ESWC 2015 Satellite Events. ESWC 2015. Lecture Notes in Computer Science, Vol. 9341. Springer, Cham, pp. 387–402. https://doi.org/10.1007/978-3-319-25639-9_51.

[7] Čerāns, K., Šostaks, A., Bojārs, U., Ovčiņņikova, J., Lāce, L., Grasmanis, M., Romāne, A., Sproģis, A., Bārzdiņš, J. (2018). ViziQuer: A Web-Based Tool for Visual Diagrammatic Queries Over RDF Data, in Gangemi, A., et al. (ed.), Proceedings of The Semantic Web: ESWC 2018 Satellite Events. ESWC 2018. Lecture Notes in Computer Science, Vol. 11155. Springer, Cham, pp. 158–163. https://doi.org/10.1007/978-3-319-98192-5_30

[8] Čerāns, K., Ovčiņņikova, J., Grasmanis, M., Lāce, L., Romāne, A. (2021). Visual Presentation of SPARQL Queries in ViziQuer. In Voila!2021, CEUR Workshop Proceedings, Vol.3023

[9] Čerāns, K., Šostaks, A., Bojārs, U., Bārzdiņš, J., Ovčiņņikova, J., Lāce, L., Grasmanis, M. and Sproģis, A., (2018). ViziQuer: A Visual Notation for RDF Data Analysis Queries. In Research Conference on Metadata and Semantics Research. Springer CCIS, Vol.846, pp.50-62.

[10] Vargas, H., Aranda, C.B., Hogan, A. (2019). RDF Explorer: A Visual Query Builder for Semantic Web Knowledge Graphs, Proceedings of The Semantic Web – ISWC 2019 Auckland, New Zealand, October 26–30, Proceedings, Part I. Springer-Verlag, Berlin, Heidelberg, 647–663. https://doi.org/10.1007/978-3-030-30793-6_37

[11] Aasman, J. (2017). Graph visualization with a time machine, In Dataconomy August 29, 2017, available at http://dataconomy.com/2017/08/graph-visualization-time-machine/

[12] Schweiger, D., Trajanoski, Z., Pabinger, S. (2014). SPARQLGraph: a web-based platform for graphically querying biological Semantic Web databases, BMC Bioinformatics 15, 279. https://doi.org/10.1186/1471-2105-15-279

[13] e Zainab, S. S., Saleem, M., Mehmood, Q., Zehra, D., Decker, S., Hasnain, A. (2015). FedViz: a Visual Interface for SPARQL Queries Formulation and Execution. In: Proceedings of the International Workshop on Visualizations and User Interfaces for Ontologies and Linked Data (VOILA 2015), Bethlehem, Pennsylvania, USA. CEUR Workshop, vol. 1456, p. 49.