SparqlFilterFlow: SPARQL Query Composition for Everyone

Florian Haag, Steffen Lohmann, and Thomas Ertl

Institute for Visualization and Interactive Systems (VIS),
University of Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany
{florian.haag,steffen.lohmann,thomas.ertl}@vis.uni-stuttgart.de

Abstract. SparqlFilterFlow provides a visual interface for the composition of SPARQL queries, in particular `SELECT` and `ASK` queries. It is based on the intuitive and empirically well-founded filter/flow model that has been extended to address the unique specifics of SPARQL and RDF. In contrast to related work, no structured text input is required but the queries can be created entirely with graphical elements. This even allows users without expertise in Semantic Web technologies to create complex SPARQL queries with only little training. SparqlFilterFlow is implemented in C#, supports a large number of SPARQL constructs and can be applied to any SPARQL endpoint.

Keywords: SPARQL, RDF, visual querying, filter/flow, semantic web, linked data, triplestore, query language, visualization, faceted search.

1 Introduction

SPARQL is currently the de facto standard for querying RDF data. It is supported by most triplestores, and many RDF datasets provide SPARQL endpoints [4,8]. However, writing SPARQL queries is not an easy task and requires knowledge about Semantic Web concepts and technologies. Since average users cannot be expected to have the necessary skills, visual interfaces are needed that hide the SPARQL syntax and provide graphical support for query building.

We present SparqlFilterFlow, a novel approach for visual SPARQL querying based on the filter/flow model. [1] It is implemented in C# and uses the Windows Presentation Foundation (WPF) for the graphical user interface. In contrast to related work, no structured text input is required. Instead, the queries can be created entirely with graphical elements. SparqlFilterFlow considers most features of SPARQL and can hence also be used for the construction of complex query expressions. In particular, it enables the creation of `SELECT` and `ASK` queries, though it may also be used for other query forms (i.e. `CONSTRUCT` and `DESCRIBE` queries) with only little variation.[2]

[1] While this demo paper presents the interactive implementation, the concept of applying the filter/flow model to SPARQL querying is described more in-depth in [10].

[2] A screencast of SparqlFilterFlow and a lightweight web demo with limited functionality are publicly available at http://sparql.visualdataweb.org.

2 Related Work

Several attempts to assist in the creation of SPARQL queries have been presented in the last couple of years. For instance, SPARQLViz [9] provides a form-based wizard that guides the user through the query building process. Other form-based approaches are the Graph Pattern Builder of the DBpedia project [6] or Konduit VQB [5]. However, these tools represent the queries in a way that is closely related to the triple syntax of RDF and SPARQL. They do not relieve the users from the need to know how SPARQL queries are structured.

An alternative is the use of visual query languages that provide graphical representations for the different SPARQL elements and combine them to node-link diagrams. NITELIGHT [16], iSPARQL [2], and RDF-GL [13] are examples of tools based on visual query languages. A slightly higher degree of abstraction is provided by approaches that use UML-like diagrams to compose SPARQL queries [7]. While these attempts help to lower the barrier for creating correct queries, they still require knowledge of the structure and syntax of SPARQL.

SparqlFilterFlow is more related to the idea of using visual pipes to process RDF data. This idea is implemented in the tools DERI pipes [15] and MashQL [14], both of which are inspired by the mashup framework Yahoo! Pipes [3]. However, these attempts focus on rearranging, sorting and transforming data and not on the composition of SPARQL queries.

3 Filter/Flow Model

SparqlFilterFlow is based on the idea of filter/flow graphs originally introduced by Young and Shneiderman in the context of relational databases and SQL querying [17]. The filter/flow model provides an intuitive representation of Boolean expressions that can be used for data filtering. The expressions are visualized as directed acyclic graphs, where the nodes define the filter criteria and the edges depict the flow of data. The thickness of the edges indicates the number of data items contained in the flow. Conjunctions are modeled as sequential paths and disjunctions as parallel paths.

Several improvements to the original filter/flow idea have been proposed over the years. We developed an extended filter/flow model that incorporates the most common ones [11]. In that model, flows are linked to explicit connection points on the filter nodes called *receptors* and *emitters*. This allows the filter nodes to receive data from several inbound flows that can be processed in different ways. Likewise, there can be several outbound flows, each representing another filter function. Along with these changes, filter nodes in the extended model are not restricted to atomic operations but can consider several filtering parameters. Finally, the extended model defines special nodes that display the result of filtering and can be placed at arbitrary positions in the graph, like any other filter node. This way, not only the final result set but also intermediate results can be shown.

Overall, the filter/flow graphs of the extended model have a smaller size and complexity, with positive effects on their readability, as we found in a comparative user study [12].

4 SparqlFilterFlow

SparqlFilterFlow implements our approach of applying the extended filter/flow model to SPARQL querying [10]. Users can visually compose queries by adding filter nodes and using drag-and-drop to connect them with flows.

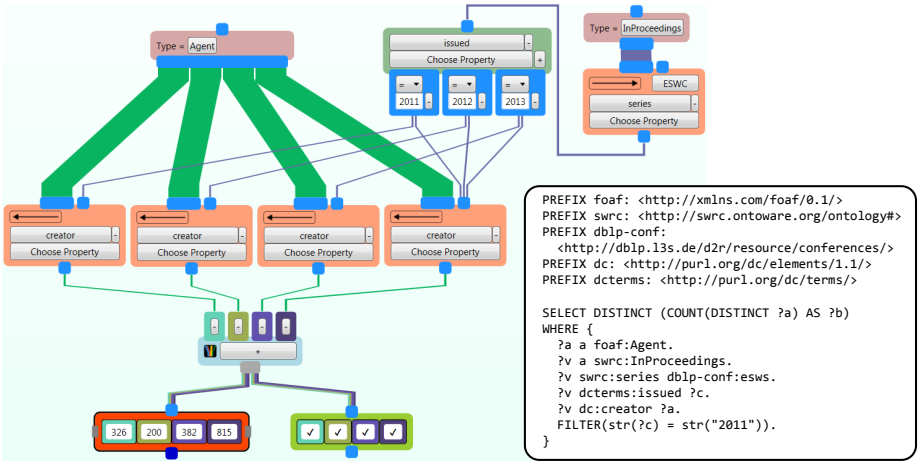

Fig. 1. Screenshot of SparqlFilterFlow: This graph counts authors of papers presented at ESWC in the years 2011 to 2013, using RDF data of Faceted DBLP [1]. One of the SPARQL queries generated by the graph is shown in the box on the right.

Example Figure 1 shows a screenshot of a filter/flow graph created with SparqlFilterFlow on the RDF dataset of Faceted DBLP [1]. Examples like this one can be created for different RDF datasets and will be shown in the ESWC demo.

The filtering starts in the initial nodes of the graph, which are the ones without inbound flows, in this case the two *type* nodes selecting all authors (`foaf:Agent`) and proceedings papers (`swrc:InProceedings`). Both sets are then gradually reduced by the subsequent filter nodes. Following the filter/flow metaphor, the thickness of the flows indicates the relative size of the sets. This helps users determine whether a given filter node has a significant effect on the data—which is the case if the thickness of the outbound flows is visibly reduced compared to the inbound flows—or even blocks the whole set.

In the example of Figure 1, only papers presented at ESWC (`dblp-conf:esws`) in the years 2011 to 2013 (`dcterms:issued`) are considered. This set of papers is then used to filter the set of authors by selecting only those people that co-authored one of the papers (`dc:creator`). The data stream is additionally split up into four sets, with the first three containing the ESWC authors from the individual years, and the fourth set containing the ESWC authors from all three years. Finally, the four sets of authors are bundled with a corresponding node.

Filter Nodes The example illustrates some of the filter nodes provided by SparqlFilterFlow. Their settings can be directly manipulated by users. The basic group of filters compares IRIs and literals, such as strings, numbers or dates, with operators like equality, greater or less than. Certain attributes of a literal can also be restricted in these filters, such as its language tag or its length in characters. Another group of filters examines the RDF graph structure, for instance the existence of a given property. There are also filters that help organize the structure of the filter/flow graph, including filters that bundle different sets to run in a single flow. Finally, there are specializations of general filter nodes that predefine frequently applied restrictions to ease query composition. An example is the *type* filter in Figure 1, which is a specialization of a comparison filter.

Result Nodes Once the desired restrictions have been defined by the combination of filter nodes, users can add result nodes that apply the restrictions in `SELECT` or `ASK` queries and display the result. In Figure 1, two result nodes have been inserted—one showing the number of authors in each of the sets (by using a SPARQL `COUNT` function along with the `SELECT` query), and one showing whether there are any results at all (by applying a SPARQL `ASK` query).[3]

The results reveal that the total number of ESWC authors was lower in the year 2012 than in the other two years. In addition, it gets apparent that the total number of authors throughout the three considered years is barely lower than the sum of the author counts per year, indicating that many of the authors contributed only in one of the considered years.

SPARQL Queries Several SPARQL queries are generated and processed during the composition of the graph. Most obviously, the result nodes issue one or more SPARQL queries when they are inserted into the graph to retrieve the values to be displayed. As an example, the SPARQL query generated to get the number of ESWC authors for the year 2011, as given by the first value in the left result node, is shown in Figure 1.

However, SPARQL queries are also generated at other points in the graph, in particular for every emitter, in order to determine the thickness of the outbound flows. The expression generator of SparqlFilterFlow always traverses the graph in upstream direction, starting at the emitter that issued the SPARQL query. It gradually constructs the query that comprises of the conjunctions, disjunctions and filter functions defined by the partial graph reachable upstream, usually but not exclusively by adding statements to the `WHERE` clause of the query. Whenever any part of the graph structure or filter node settings changes, all nodes reachable downstream from the changed graph part may be affected and are thus notified, whereupon they reissue their SPARQL queries.

5 Conclusion and Future Work

SparqlFilterFlow enables the composition of SPARQL queries using exclusively graphical elements and simple text strings, while avoiding any structured text

[3] The result node for the `ASK` query is only added for illustration purposes in this case, as it is somewhat redundant to the result node applying the `COUNT` function.

input. It requires no knowledge of Semantic Web concepts beyond a basic understanding of the RDF idea. It can be applied to any SPARQL endpoint and allows for creating complex SPARQL queries with only little training. Results from a qualitative user study indicate that the approach is comparatively usable and easy to learn [10].

Future work includes support for the creation of `DESCRIBE` and `CONSTRUCT` queries besides `SELECT` and `ASK` queries. This will require the integration of additional visualization and interaction concepts, such as an intuitive way to specify the graph structure for the result of the `CONSTRUCT` query. Another goal of future work is the development of features that suggest appropriate filter nodes and values based on the schema information available in the RDF data.

References

1. Faceted DBLP. http://dblp.l3s.de.
2. OpenLink iSPARQL. http://oat.openlinksw.com/isparql/.
3. Pipes: Rewire the web. http://pipes.yahoo.com/pipes/.
4. SPARQL endpoints status. http://sparqles.okfn.org.
5. O. Ambrus, K. Möller, and S. Handschuh. Konduit VQB: a visual query builder for SPARQL on the social semantic desktop. In *Proc. VISSW '10*. CEUR-WS, vol. 565, 2010.
6. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *Proc. ISWC '07/ASWC '07*, pages 722–735. Springer, 2007.
7. G. Barzdins, S. Rikacovs, and M. Zviedris. Graphical query language as SPARQL frontend. In *Proc. ABDIS '09, Workshops and DC*, pages 93–107, 2009.
8. C. Bizer, T. Heath, and T. Berners-Lee. Linked data – the story so far. *Int. J. Semant. Web. Inf.*, 5(3):1–22, 2009.
9. J. Borsje and H. Embregts. Graphical query composition and natural language processing in an RDF visualization interface. Bachelor thesis, EUR, 2006.
10. F. Haag, S. Lohmann, S. Bold, and T. Ertl. Visual SPARQL querying based on extended filter/flow graphs. In *Proc. AVI '14*, to appear.
11. F. Haag, S. Lohmann, and T. Ertl. Simplifying filter/flow graphs by subgraph substitution. In *Proc. VL/HCC '12*, pages 145–148. IEEE, 2012.
12. F. Haag, S. Lohmann, and T. Ertl. Evaluating the readability of extended filter/flow graphs. In *GI '13*, pages 33–36. CIPS, 2013.
13. F. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak. RDF-GL: A sparql-based graphical query language for RDF. In *Emergent Web Intelligence: Advanced Information Retrieval*, pages 87–116. Springer, 2010.
14. M. Jarrar and M. D. Dikaiakos. MashQL: A query-by-diagram topping SPARQL. In *Proc. ONISW '08*, pages 89–96. ACM, 2008.
15. C. Morbidoni, A. Polleres, D. L. Phuoc, and G. Tummarello. Semantic web pipes. Technical Report 2007-11-07, DERI, 2007.
16. A. Russell, P. Smart, D. Braines, and N. Shadbolt. NITELIGHT: A graphical tool for semantic query construction. In *Proc. SWUI '08*. CEUR-WS, vol. 543, 2008.
17. D. Young and B. Shneiderman. A graphical filter/flow representation of boolean queries: a prototype implementation and evaluation. *J. Am. Soc. Inf. Sci.*, 44(6):327–339, 1993.