SPARQL Playground: a Block Programming Tool to Experiment with SPARQL

Paolo Bottoni and Miguel Ceriani

Sapienza, University of Rome, Italy
bottoni@di.uniroma1.it, ceriani@di.uniroma1.it

Abstract. SPARQL is a powerful query language for Semantic Web data sources but one which is quite complex to master. As the block programming paradigm has been succesfully used to teach programming skills, we propose a tool that allows users to build and run SPARQL queries on an endpoint without previous knowledge of the syntax of SPARQL and the model of the data in the endpoint (vocabularies and semantics). This user interface attempts to close the gap between tools for the lay user that do not allow to express complex queries and overtly complex technical tools.

1 Introduction

While the available Linked Data sources are increasing in quantity and diversity, their usage is still limited. One of the barriers for the adoption of Semantic Web standards, even by technology-savvy users, is their perceived complexity.

Whether someone wants to explore an RDF dataset or in general the Linked Data cloud, the options are usually either to use a Linked Data browser for a purely resource-centric view or to switch to writing queries using the SPARQL language [9], the standard query language for RDF. Writing SPARQL requires knowledge of the syntax and also a basic knowledge of the model underlying the dataset (vocabularies that are used, semantics that are implemented). As community of developers and consumers of Semantic Web technologies, we should challenge us to close this gap. There is a need for tools that may be used in a modular and progressive way to guide the users from the design of simple queries to complex ones.

Block programming languages, in which coding is done by dragging and connecting fragments shaped like jigsaw puzzle pieces, have been successfully used to introduce programming to non-experts. Recently, tens of milions of users have been exposed to the basics of programming using Blockly [5] as part of code.org's *Hour of Code* [1]. The same metaphor was used in Scratch [11] to create animations and games and in MIT App Inventor [16] to build Android Apps.

In [3] we proposed to use the block programming paradigm to design queries on Linked Data sources. Apart from the goals stated above, we decided, as in the philosophy of Block Programming, to design our tool as a way to gradually experiment the structure of the "real" underlying language and in the end to be able to switch to directly programming in that language. For these reasons the visual language mimics the

[1] https://code.org/about

structure of the syntax of SPARQL, while at the same time trying to avoid excessive verbosity that would lead to cognitive overload for the user. Compared to previous uses of block programming languages, this proposal addresses novel challenges due to two main specific properties: 1) the heterogeneous nature of Linked Data, that requires the ability to explore graph datasets even without any *a priori* knowledge; 2) the structural difference between procedural imperative languages for which this paradigm was previously used and a functional query language like SPARQL. To deal with these challenges we proposed a novel paradigm, favouring direct reuse of query results through the integration of the visual space used for query design and results visualization.

In the present paper we present a live demo of the tool. Through this online demo we want to promote discussion on the topic of visual interfaces for SPARQL and specifically evaluate the usability and reception of our proposal. Moreover, the user interface has been enhanced to permit the execution of queries on multiple SPARQL endpoints.

In the rest of the paper, Section 2 reports on related work while Section 3 presents the tool. Section 4 gives details on the implementation and the presented demo and Section 5 summarizes the proposal and draws some conclusions.

2 Related Work

Several interactive tools have been proposed to support the structured querying of RDF data sources, at various levels of abstraction and using different paradigms. A basic distinction can be made between: 1) tools that require writing and reading SPARQL syntax and 2) tools that provide other metaphors (usually visual) aimed at lowering the learning curve and providing more intuitive interaction. The first kind of UIs include advanced editors as YASGUI [12] or integrated environments as Twinkle[2], but to design the query the user has still to know SPARQL and the vocabularies used.

UIs of the second kind provide interaction with another representation of the query –textual or visual– that is then transformed to SPARQL to be executed. The text based UIs use forms –such as SPARQLViz [2]– or controlled construction of natural language statements –such as SPARKLIS [4]. These systems do not scale well when the query complexity increases and do not easily permit code reuse. As for the visual tools, most of them use a graph-based paradigm (NITELIGHT [13], QueryVOWL [8]), others use a dataflow-based paradigm (SparqlFilterFlow [7]), and at least one uses a combination of both (VQS [6]). Graph-based interfaces fit the RDF graph pattern matching model very well, while dataflow-based interfaces are effective in representing SPARQL functional operators (e.g., `UNION`). Nevertheless, both types of interfaces are highly inefficient in terms of space on user screen and often present problems with interaction.

A previous important proposal for using block programming for SPARQL queries is the SPARQL/CQELS Visual Editor designed for the Super Stream Collider framework [10]. In that case the blocks strictly follow the language structure and syntax and the tool requires at least basic knowledge of SPARQL to be used. Conversely, the user interface we propose is designed to provide blocks that should be mostly self describing and usable without knowing the SPARQL syntax in advance. Finally, for most of

[2] `http://www.ldodds.com/projects/twinkle/`

the existing tools the visualization of the result set is passive and often presented in an indipendent panel/window (e.g., in many Web-based interfaces the result page replaces the query page). In our proposal, on the contrary, results and query share there same workspace to allow for an exploratory pattern of interaction.

3 Proposed User Interface

The following were the basic requirements around which the user interface was defined:

1. users should not care about the syntax – hence visual clues and constraints should prevent syntax errors;
2. the need to input text by users should be minimized;
3. there should be direct ways to build commonly used structures;
4. users should be able to use the tool as a step to learn the SPARQL (textual) syntax – hence the used blocks should follow the structure of the language;
5. users should be able to work even without prior knowledge of the dataset – hence exploratory queries should be explicitly supported.

The queries are designed composing the set of available blocks. For example, Figure 1 represents a *select* query –against LinkedGeoData data set [15]– to get the names of the first three regions (the first administrative subdivision) of Italy by alphabetical order. The query is represented by the *select all* block and its sub-blocks. Among them, the sub-block connected to the *where* connection is a graph pattern and corresponds to the `where` clause of the query. Inside graph patterns and expressions, different types of graph terms can be used: *IRIs* (represented in brown and using the prefixed notation), *variables* (using Blockly appearance of variables for consistency), and *literals* (represented in different colours according to their type, numeric, string or boolean). The SPARQL query corresponding to Figure 1 is:

```
SELECT DISTINCT * WHERE {
  lgdt:relation365331 lgdo:members [?p ?member].
  ?member
    lgdo:role 'subarea';
    lgdo:ref [rdfs:label ?subareaName].
  FILTER(LANGMATCHES(LANG(?subareaName), 'en'))
}
ORDER BY (?subareaName) LIMIT 5
```

However, in order to design such a query some knowledge about the specific dataset (that there exists a resource `lgdt:relation365331` representing Italy) and the used vocabularies (that the property `lgdo:members` associates an area with a container of items, of which the ones with `gdo:role` equal to "subarea" are administrative subdivisions of the area) is still required.

If the dataset is unknown this information is usually gathered through preliminary, explorative queries. We thus designed the user interface especially to favour the reuse of query results in the same or new queries. The *execution* block is used to execute a query and to show the result set as soon as it is available. The produced result set is shown again in the form of blocks, which can be dragged to other parts of the workspace and connected to other blocks. Results from a query can thus easily be used as parts of another query. For the previous query the tabular results can be seen in Figure 1 as well.

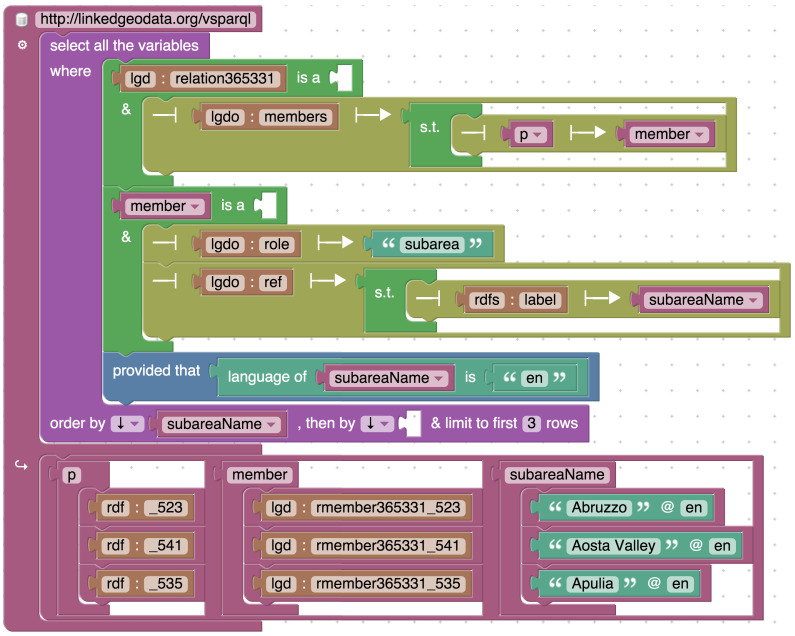

Fig. 1. Execution of a query to get the first three English names of regions of Italy.

The execution block provides also a field to set the remote SPARQL endpoint to which the query is sent –in this case the public SPARQL endpoint for LinkedGeoData[3]. There is no formal limit to the number of execution blocks that can be used, new queries may thus be built while keeping aside previously built queries and their results.

Ideally, the results of execution would arrive within a small time. While this may be true for not-too-complex queries against SPARQL endpoints that perform well, it cannot be guaranteed for the general case. For that reason query executions are non-blocking, i.e. the user interface stays reactive while waiting for a result from the server. The user is thus able to keep working at the same query or other queries while the query is being executed. If a query is modified during execution, the execution is aborted and restarted with the updated query.

When exploring a new dataset, knowing the used vocabularies and looking for specific resources are a common need. For this reason the toolbox already contains some pre-built queries that can be used to look for resources, classes and properties used in the dataset. These pre-built queries are just sets of pre-connected blocks that can be freely rearranged and decomposed on the workspace. Figure 2 shows the query prepared to look for specific resources of a certain type, modified to look for a resource labelled "Italy" and of type `lgdm:Relation` (that is the class used in LinkedGeoData for geographical composite elements). The result of the query, together with the result of similar explorative queries, can be reused to write a query as the one in Figure 1.

[3] `http://linkedgeodata.org/vsparql`

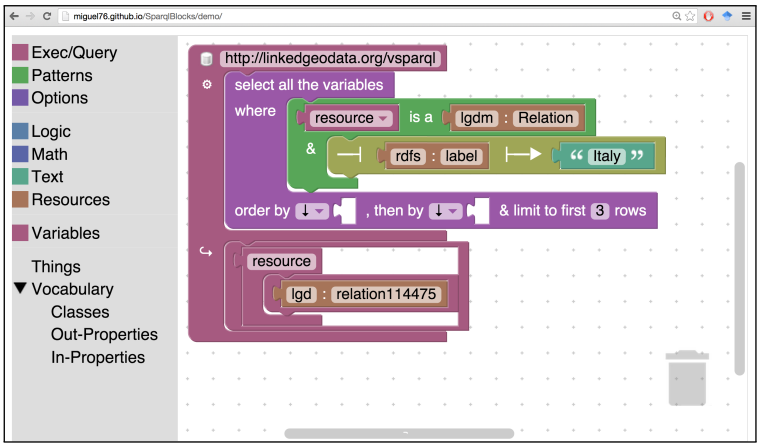

Fig. 2. View of the UI after getting a resource corresponding to Italy in LinkedGeoData.

Figure 2 also shows the user interface (enlarged for readability). The toolbox on the left –from which new blocks may be dragged on the workspace– has categories corresponding to the different types of blocks. The last two categories –Things and Vocabulary, with its subcategories– contain the aforementioned pre-built queries.

4 Implementation and Demo

The tool is based on an extension of the Blockly JavaScript library, working entirely on the client side. We extended the library to supply the specific blocks needed for SPARQL queries and execution. We also added the necessary code to generate SPARQL fragments from the blocks. The SPARQL execution block listens for changes in its query connection; each time the query changes, the corresponding SPARQL query is generated and sent to a SPARQL endpoint. The SPARQL endpoint used is set as a field of the execution block. The results are used to dynamically generate the result block and its sub-blocks. The standard prefix definitions from *prefix.cc*[4] are used to add prefix declarations in the query sent to the endpoint and to convert the IRIs in the result to the prefixed notation.

The online demo[5] is an instance of the tool provided to everyone willing to experiment with this new user interface. By default the queries are executed against the SPARQL endpoint of DBPedia[6] [1], but –as shown in the examples in Section 3– any other public SPARQL endpoint can be accessed[7]. Using the context menu on the blocks, queries and fragments of queries may be exported as SPARQL. Query results may also

[4] `http://prefix.cc/`
[5] `http://miguel76.github.io/SparqlBlocks/demo`
[6] `http://live.dbpedia.org/sparql`
[7] As the tool runs on the browser, the endpoints have to be CORS-enabled; non CORS-enabled endpoints may be reached through a proxy.

be exported in JSON format[14]. Apart from using the online demo, the code itself may be also freely forked or donwloaded from GitHub[8].

5 Conclusions

We developed a new visual user interface to allow non-experts to build SPARQL queries. The tool does not require prior knowledge of the used dataset and vocabularies, favouring an exploratory and constructive way of building queries. An online demo –from which any public SPARQL endpoint can be queried– has been setup to showcase the user interface to the communities of Semantic Web developers and reasearchers, in order to have feedback from a wider audience and foster discussion in this field.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. In: Proc. of ISWC 2007. pp. 722–735. Springer (2007)
2. Borsje, J., Embregts, H.: Graphical query composition and natural language processing in an RDF visualization interface. B.S. Thesis, Erasmus School of Economics and Business Economics, Erasmus University, Rotterdam (2006)
3. Bottoni, P., Ceriani, M.: Linked Data Queries as Jigsaw Puzzles: a Visual Interface for SPARQL Based on Blockly Library. In: Proc. of CHItaly 2015. p. [To Appear]. ACM (2015)
4. Ferré, S.: Sparklis: a SPARQL Endpoint Explorer for Expressive Question Answering. In: Proc. of ISWC 2014 Posters & Demonstrations Track. vol. 1272. CEUR-WS (2014)
5. Fraser, N., et al.: Blockly: a visual programming editor (2013)
6. Groppe, J., Groppe, S., Schleifer, A.: Visual Query System for Analyzing Social Semantic Web. In: Proc. of the WWW '11. pp. 217–220. ACM (2011)
7. Haag, F., Lohmann, S., Bold, S., Ertl, T.: Visual SPARQL Querying based on Extended Filter/Flow Graphs. In: Proc. of AVI 2014. pp. 305–312. ACM (2014)
8. Haag, F., Lohmann, S., Siek, S., Ertl, T.: QueryVOWL: Visual Composition of SPARQL Queries. In: Proc. of ESWC 2015 Satellite Events. Springer (2015)
9. Harris, S., et al.: SPARQL 1.1 Query Language. W3C REC 21 March 2013
10. Quoc, H.N.M., Serrano, M., Le-Phuoc, D., Hauswirth, M.: Super Stream Collider-Linked Stream Mashups for Everyone. In: Proc. of the Semantic Web Challenge at ISWC 2012 (2012)
11. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al.: Scratch: programming for all. Communications of the ACM 52(11), 60–67 (2009)
12. Rietveld, L., Hoekstra, R.: YASGUI: Not Just Another SPARQL Client. In: Proc. of ESWC 2013 Satellite Events. pp. 78–86. Springer (2013)
13. Russell, A., Smart, P.R., Braines, D., Shadbolt, N.R.: NITELIGHT: A Graphical Tool for Semantic Query Construction. In: Proc. of SWUI '08. vol. 543. CEUR-WS (2008)
14. Seaborne, A.: SPARQL 1.1 Query Results JSON Format. W3C REC 21 March 2013
15. Stadler, C., Lehmann, J., Höffner, K., Auer, S.: LinkedGeoData: A Core for a Web of Spatial Open Data. Semantic Web Journal 3(4), 333–354 (2012)
16. Wolber, D., Abelson, H., Spertus, E., Looney, L.: App Inventor. O'Reilly Media, Inc. (2011)

[8] `https://github.com/miguel76/SparqlBlocks`