SPARQL2Git: Transparent SPARQL and Linked Data API Curation via Git

Albert Meroño-Peñuela[1] and Rinke Hoekstra[1,2]

[1] Department of Computer Science, Vrije Universiteit Amsterdam, NL
{albert.merono, rinke.hoekstra}@vu.nl
[2] Faculty of Law, University of Amsterdam, NL

Abstract. In this demo, we show how an effective and application agnostic way of curating SPARQL queries can be achieved by leveraging Git-based architectures. Often, SPARQL queries are hard-coded into Linked Data consuming applications. This tight coupling poses issues in *code maintainability*, since these queries are prone to change to adapt to new situations; and *query reuse*, since queries that might be useful in other applications remain inaccessible. In order to enable decoupling, version control, availability and accessibility of SPARQL queries, we propose SPARQL2Git, an interface for editing, curating and storing SPARQL queries that uses cloud based Git repositories (such as GitHub) as a backend. We describe the query editing and management capabilities of SPARQL2Git, its convenience for SPARQL users that lack Git knowledge, and its combination with `grlc` to easily generate Linked Data APIs.

Keywords: SPARQL, Git, Query curation, Query update, Query history

1 Introduction

The SPARQL Protocol and RDF Query Language [10] is a well known method of accessing Linked Data that allows users to query a wide variety of Linked Data sources [9]. Its implementation over HTTP, and the availability of libraries for using it in various programming frameworks, has also enabled its use by *Linked Data consuming applications*.

Usually, developers who want to retrieve Linked Data hard-code SPARQL queries into their code. This gives raise to two important issues. First, SPARQL queries become a critical component in the execution of such applications, making these applications *harder to maintain*. For instance, changes introduced in datasets of the queried endpoints may oblige these queries to change accordingly; often in more than one place, if the same query is used among various applications. Secondly, queries buried into application source code are *harder to discover and reuse*, even if the application is open source. Users are forced to scan code and copy-paste these queries, with little attribution to the provenance of the reused queries.

Some query catalogs have been developed to overcome these issues. For instance, LinkedWiki[3] has a query sharing service where SPARQL queries can be found and reused, including code snippets for various programming languages. Another example is LSQ, the Linked SPARQL Queries dataset [8], which describes queries extracted from the logs of public SPARQL endpoints as Linked Data. In the CEDAR [4] and CLARIAH [3] projects, we have adopted a Git-centric approach, in which we curate queries independently of consuming applications. This query centralization has decoupled SPARQL queries from the various applications, like map visualizations[4], query interfaces[5], and even on-the-fly generated Linked Data APIs [6] with `grlc` [6], that depend on them to function. Moreover, by using Git and the API of Git repository managers such as GitHub, we enable *versioning*, *unique identification*, and *de-referenceability* of queries at a fine-grained, commit level, among other features of modern distributed version control systems.

In this paper, we describe SPARQL2Git, a system that builds on these foundations, and leverages the Web-based SPARQL editor library YASQE [7] and features of the Linked Data API generator `grlc` [5], to enable the curation of SPARQL queries, and their associated Linked Data APIs, in an effective, application-decoupled, and Git-agnostic way (Section 2). In Section 3 we show the contents of our demonstration, focusing on the user interaction workflow and the technology involved, and we discuss future work.

2 SPARQL2Git

SPARQL2Git is an open source[7] server and user interface for editing, documenting, and committing SPARQL queries to GitHub repositories. The public instance of SPARQL2Git is available at `http://sparql2git.com`.

The *welcome screen* asks the user to log in using GitHub's OAuth, thus a GitHub account is required in order to use SPARQL2Git. SPARQL2Git needs the user to grant permission to read/write the user's public repositories, and the user's personal information (username and email). In the next screen, the *repository selection screen* shows the complete list of repositories of the authenticated user. The user can either choose and existing repository, or create a new one. This repository will be used to store SPARQL queries.

The next screen is the *query editing screen*, shown in Figures 1 and 2. In this screen, users first select the query they are interested in editing, from the left pane (Figure 1). Users can also create a new query, or delete an existing one. After this, users create or edit a query in two steps: query *metadata*, and query *body*. Query metadata are necessary to create compliant API specifications on

[3]See `http://linkedwiki.com/searchExample.php`

[4]See `http://www.nlgis.nl/`

[5]See `http://lod.cedar-project.nl/data.html`

[6]See `http://grlc.io/api/CEDAR-project/Queries/` and `http://grlc.io/api/CLARIAH/wp4-queries/`

[7]See `https://github.com/albertmeronyo/SPARQL2Git`

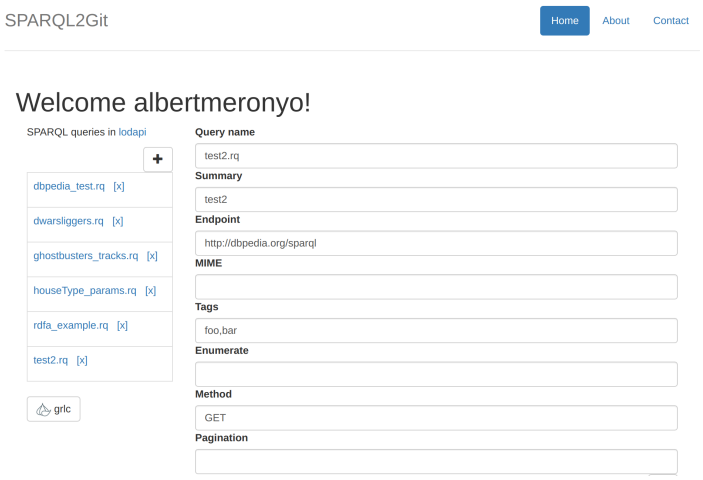

Fig. 1. SPARQL2Git metadata form. Users can select SPARQL queries in the left pane, and edit their API relevant metadata (query name, summary, endpoint, etc.) on the right.

top of SPARQL queries, and consist of (see Figure 1): a query name; a brief summary; the SPARQL endpoint where the query should be sent; a MIME type (if the endpoint is an RDF dump or an HTML page with embedded RDFa); one or more tags, which are used to neatly organize queries in equivalent APIs; enumerations, which are used to create dropdown lists for parameter values in equivalent APIs; HTTP method (GET, POST, etc); and a pagination number n, provided the user wants the query results to be returned in pages of n elements.

The query body is the SPARQL query itself, and can be edited below the metadata as shown in Figure 2. We use the YASQE and YASR UI libraries of [7] for prefix autocompletion, syntax highlighting, and other user friendly features. Users can press the *play* button to test their queries against the endpoint specified in Figure 1, and see the results in the table below the SPARQL editor (Figure 2). Once they are satisfied with the result, users can click on the commit button, which is placed on top of the query editor with a cloud sign (Figure 2). After this, a dialog appears requesting a comment on the commit; this will be used as a commit message with the GitHub API interaction. After confirming, SPARQL2Git sends a request to the GitHub API to commit a new version of the file, with the supplied comment, metadata and body, over the file's last commit SHA hash. Users can always click on the link to the GitHub page of their repository to check the outcomes.

Many of the features of SPARQL2Git are in place to generate `grlc` compliant APIs, in addition to the curation of SPARQL queries. SPARQL2Git transforms the contents of the data supplied in this screen (Figures 1 and 2) to `grlc`'s notation for Linked Data APIs [6]. This way, Linked Data APIs can be generated on the fly right after users commit changes to their queries. To check these APIs, users can click on the `grlc` link below the query list pane (see Figure 1).

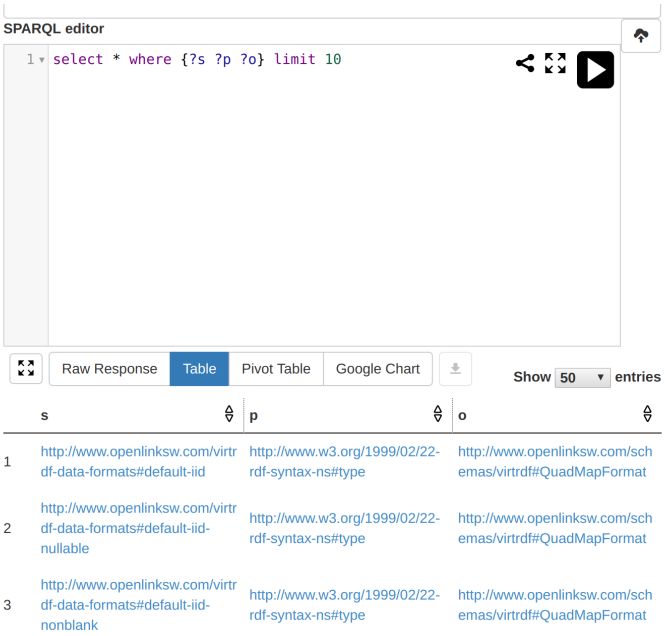

Fig. 2. SPARQL2Git query editor. We use the YASQE and YASR UI libraries [7] for user friendly editing and testing of SPARQL queries. The commit button at the top allows users to commit and push the query to their GitHub repo, documenting the changes.

3 Demonstration

The demonstration will consist of the following parts: (1) basic SPARQL2Git workflow; (2) coherence with the GitHub repository contents; and (3) generating Linked Data APIs with `grlc`. In the first part, visitors to the demo will be able to interact with their GitHub repositories using their own credentials, following the workflow described in Section 2. A screencast of the general SPARQL2Git workflow is available online[8], and will be used as main guide for this part. In the second part, users will check the results of the first part by exploring their modified GitHub repository, exploring the resulting annotated SPARQL queries, and whether their commit history is coherent with their previous edits in SPARQL2Git. Additionally, we will prompt users to use well-known tools, such as Git2PROV [1] and PROV-O-Viz [2], to better understand this commit history as PROV triples and visualizations. Finally, in the third part visitors turn their SPARQL queries into Linked Data APIs using `grlc` with no additional effort, combining different API specification values (see Figure 1). Moreover, different Linked Data access methods other than SPARQL will be used in combination with the SPARQL2Git curated queries to generate universal access APIs to Linked Data.

[8] See `https://vimeo.com/207296874`

We plan to further extend SPARQL2Git in different ways. First, we will integrate additional Git features into SPARQL2Git, such as branching, specific commit SHA hash editing, etc. Second, we will integrate and make better accessible the PROV generated by Git2PROV and visualized by PROV-O-Viz into SPARQL2Git; and we will link the PROV triples of the commit history of SPARQL queries with the PROV of `grlc` generated at API creation time. Finally, we will implement caching mechanisms for a more efficient synchronisation and editing of SPARQL queries and their related metadata.

Acknowledgements This work was funded by the CLARIAH project of the Dutch Science Foundation (NWO) and by the Dutch national programme COMMIT.

References

1. De Nies, T., Magliacane, S., Verborgh, R., Coppens, S., Groth, P., Mannens, E., Van de Walle, R.: Git2PROV: Exposing version control system content as W3C PROV. In: Poster and Demo Proceedings of the 12th International Semantic Web Conference (Oct 2013), http://www.iswc2013.semanticweb.org/sites/default/files/iswc_demo_32_0.pdf
2. Hoekstra, R., Groth, P.: PROV-O-Viz - Understanding the Role of Activities in Provenance. In: 5th International Provenance and Annotation Workshop (IPAW 2014). LNCS, Springer-Verlag, Berlin, Heidelberg (2014)
3. Hoekstra, R., Meroño-Peñuela, A., Dentler, K., Rijpma, A., Zijdeman, R., Zandhuis, I.: An Ecosystem for Linked Humanities Data. In: Proceedings of the 1st Workshop on Humanities in the Semantic Web (WHiSe 2016), ESWC 2016 (2016), under review
4. Meroño-Peñuela, A., Guéret, C., Ashkpour, A., Schlobach, S.: CEDAR: The Dutch Historical Censuses as Linked Open Data. Semantic Web – Interoperability, Usability, Applicability 8(2), 297—310 (2015)
5. Meroño-Peñuela, A., Hoekstra, R.: grlc Makes GitHub Taste Like Linked Data APIs. In: The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Revised Selected Papers. pp. 342–353. Springer (2016)
6. Meroño-Peñuela, A., Hoekstra, R.: The Song Remains The Same: Lossless Conversion and Streaming of MIDI to RDF and Back. In: The Semantic Web: ESWC Satellite Events (ESWC 2016). Springer (2016)
7. Rietveld, L., Hoekstra, R.: The YASGUI family of SPARQL clients. Semantic Web 8(3), 373–383 (2017), http://dx.doi.org/10.3233/SW-150197
8. Saleem, M., Ali, M.I., Hogan, A., Mehmood, Q., Ngomo, A.N.: LSQ: the linked SPARQL queries dataset. In: The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II. pp. 261–269 (2015), http://dx.doi.org/10.1007/978-3-319-25010-6_15
9. Vandenbussche, P.Y., Umbrich, J., Matteis, L., Hogan, A., Buil-Aranda, C.: Sparqles: Monitoring public sparql endpoints. Semantic Web Journal (2016)
10. W3C: SPARQL 1.1 Overview. https://www.w3.org/TR/sparql11-overview/