grlc Makes GitHub Taste Like Linked Data APIs

Albert Meroño-Peñuela[1,2] and Rinke Hoekstra[1,3]

[1] Department of Computer Science, Vrije Universiteit Amsterdam, NL
{albert.merono,rinke.hoekstra}@vu.nl
[2] Data Archiving and Networked Services, KNAW, NL
[3] Faculty of Law, University of Amsterdam, NL

Abstract. Building Web APIs on top of SPARQL endpoints is becoming a common practice to enable universal access to the integration favorable dataspace of Linked Data. However, the Linked Data community cannot expect users to learn SPARQL to query this dataspace, and Web APIs are the most common way of enabling programmatic access to data on the Web. However, the implementation of Web APIs around Linked Data is often a tedious and repetitive process. Recent work speeds up this Linked Data API construction by wrapping it around SPARQL queries, which carry out the API functionality under the hood. Inspired by this, in this paper we present `grlc`, a lightweight server that translates SPARQL queries curated in GitHub repositories to Linked Data APIs on the fly.

Keywords: SPARQL, git, GitHub, Linked Data APIs

1 Introduction

Despite their known benefits for data integration, the Linked Data technologies of RDF and SPARQL still operate in a niche. There is a gap with what average Web-applications and developers have come to expect. RDF and SPARQL remain relatively unknown to the wider Web community, albeit being still a requirement to access Linked Data. Both have steep learning curves that many developers refuse to face. The W3C specification of SPARQL 1.1 has a limited adoption even within the Linked Data community [14]. Linked Data APIs[4] have emerged as a solution to this problem, proposing the deployment of Web APIs on top of Linked Data resources, e.g. as an interface to SPARQL endpoints. This use of APIs to apply the basic principle of encapsulation, and their deployment in large scale Linked Data applications [4], has proved to solve this problem.

However, the construction of these APIs by Linked Data developers is still a cumbersome task. The deployment of effective Linked Data APIs requires careful management of SPARQL queries, reliable storage, abundant documentation, and the overhead of software maintenance. The latter has been recently addressed by Daga et al. [1], who propose a system that builds API operations automatically by taking a SPARQL query and an endpoint location as input. However, the question of how to effectively store and organise such API-translated SPARQL

[4] `https://github.com/UKGovLD/linked-data-api`

queries remains. As shown in this paper, users require organised APIs that adapt to their existing query curation workflows. This requires a paradigm shift where not just the data, but the queries themselves also become *first class citizens*.

In recent Linked Data projects such as CEDAR [9] and CLARIAH-SDH [6] we followed a practice of storing, curating, and publishing illustrative SPARQL queries of their use cases using GitHub repositories. These queries are then used by various client applications to access Linked Data. In this paper, we investigate how the current practice of curating queries in open GitHub repositories can be decoupled from, and used to lower the costs of, constructing APIs for Linked Data applications. Concretely, the contributions of this paper are:

- A mapping specification between the Swagger RESTful API, and SPARQL query repositories accessible through the GitHub API
- A decorator syntax to enrich SPARQL queries in git repositories with metadata about their intended use (Section 3.2)
- A description of the `grlc` service, that automatically exposes such enriched SPARQL queries in GitHub repositories as Linked Data APIs (Section 4)

As for the rest of the paper, we survey relevant related work in Section 2, evaluate our approach in two use cases in Section 5, and conclude in Section 6.

2 Related Work

The organization and management of SPARQL queries is central to the study of their efficiency, nature, and use at improving Linked Data applications. SPARQL query logs have been used to study differences between queries by humans and machines [11]. These logs are also useful to understand semantic relatedness of queried entities [7]. Saleem et al. [12] propose to "create a Linked Dataset describing the SPARQL queries issued to various public SPARQL endpoints". To the best of our knowledge, no previous work addresses the use of collaborative code platforms to ease deployment of Web APIs.

The Semantic Web has developed significant work on the relationship between the Linked Data and Web Services [3,10]. In [13], authors propose to expose REST APIs as Linked Data. These approaches suggest the use of Linked Data technology on top of Web services. Our work is related to results in the opposite direction, concretely the Linked Data API specification[5] and the W3C Linked Data Platform 1.0 specification, which "describes the use of HTTP for accessing, updating, creating and deleting resources from servers that expose their resources as Linked Data"[6]. The OpenPHACTS Discovery Platform for pharmacological data [4], and the BASIL server [1], which also builds Linked Data APIs compliant with the Swagger RESTful API specification[7], are work we directly build on. Our contribution proposes additional decoupling of Linked Data APIs with SPARQL query curation infrastructures, in order to lower the costs of building and maintaining such APIs.

[5] https://github.com/UKGovLD/linked-data-api

[6] https://www.w3.org/TR/2015/REC-ldp-20150226/

[7] https://github.com/OAI/OpenAPI-Specification

3 From GitHub Repos to Linked Data APIs

It is becoming increasingly popular to maintain a great variety of data projects beyond software code using git repositories, especially those in GitHub[8] [8]. For example, we have used GitHub to store important SPARQL queries and templates for the CEDAR and CLARIAH projects. This has brought two key outcomes for these projects. First, it has contributed a *better maintainability* of the life cycle of SPARQL queries. By leveraging git features and GitHub's infrastructure, queries become easily reusable (since they get unique, dereferenceable URIs), their provenance better traceable [2], their development (through frictionless branching) less error-prone, and their versioning trivial. Second, it *lowers coupling* between SPARQL queries and applications, by separating their workflows while keeping queries accessible. Consequently, these queries are less frequently hard-coded and retyped. The idea behind `grlc` is to use this decoupling to simplify the infrastructure of building and exposing Linked Data APIs.

This section investigates how the organisational characteristics of GitHub repositories can be used to build, manage and maintain a Swagger-spec compliant API. First, in Section 3.1 we study the requirements of Swagger-compliant APIs and map them to elements of the GitHub API. Since these elements are insufficient for a complete API spec, in Section 3.2 we propose to complete it with non-intrusive SPARQL decorators.

3.1 Mapping Swagger and GitHub

We propose to align the metaphor of the *repository* with that of the *API*, since both share abstract notions of organizing files and operations in a way that is meaningful for their users. For this reason, in this section we study a possible mapping between the two. Table 1 shows the mapping between the attribute requirements of the Swagger RESTful API specification, and how these correspond with either attributes of the GitHub API (repository organisation elements) or attributes of the SPARQL usage decorator (usage metadata elements). The latter are discussed in Section 3.2.

3.2 SPARQL Decorators

To complete the mapping of the GitHub API to the Swagger RESTful API specification shown in Table 1, we propose *SPARQL decorators* to add metadata in queries as comments. We assume SPARQL queries organised as `.rq` files in git repositories. Each of these files will translate into an API operation. We propose to comment them in the first file lines, with the syntax depicted in the following example[9]:

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

[9] Additional examples can be found at `https://github.com/CEDAR-project/Queries` and `https://github.com/CLARIAH/wp4-queries`

Swagger attribute	**Scope**	**Description**	**Mapping**
Swagger version	API	Version number of compliant Swagger RESTful API specification	**Static**: independent of the LDA. Currently version 2.0 of the Swagger RESTful API spec is supported
API version	API	Version number of the API	**GitHub API**: last repo release from the release API through `GET /repos/:owner/:repo/releases/latest`
Title	API	Title of the API	**GitHub API**: name of the repository through `GET /repos/:owner/:repo`
Contact name	API	Author and contact information	**GitHub API**: login name of the repository owner through `GET /repos/:owner/:repo`
Contact URL	API	URL to be followed for additional information	**GitHub API**: link to the HTML page of the repository owner through `GET /repos/:owner/:repo`
License	API	License under which the API is released	**Repository file**: a link to the raw `LICENSE` file of the repo if it exists; empty otherwise
Host	API	Host name to compose the API calls	**grlc parameter**: supplied host name in grlc's configuration; `localhost` by default
Base path	API	Base path to compose the API calls	**GitHub API**: the string `/:owner/:repo` in `GET /repos/:owner/:repo`
Schemes	API	Supported schemes to compose the API calls	**Static**: `http` is supported
Path name	Operation	Name of the API operation	**GitHub API**: the file name, without the extension, of any `.rq` file found in the repository
Path method	Operation	HTTP method for the operation (`GET`, `POST`)	**Static**: `GET` is supported
Path tags	Operation	Tags under which the operation will be classified	**SPARQL decorator**: the parsed list of the decorator *tags* in `.rq` files
Path description	Operation	Description of the API operation	**SPARQL decorator**: the parsed *description* decorator in `.rq` files
Path parameters	Operation	Parameters of the operation	**SPARQL decorator**: all parameter placeholders parsed in the query (see Section 4)
Path responses	Operation	Responses of the operation	**SPARQL decorator**: response codes on success, datatypes of parameters (see Section 4)

Table 1. Mappings between the Swagger RESTful API and the GitHub API/SPARQL decorators. Such decorators, and the query itself, are parsed through accessing any file with the extension `.rq` in the repo via `GET raw.githubusercontent.com/:owner/:repo/master/`

```
#+ summary: A brief summary of what the query does
#+ endpoint: http://example.org/sparql
#+ tags:
#+     - UseCase1
#+     - Awesomeness
```

This indicates the *summary* of the query (which will document the API operation), the *endpoint* to send the query, and the *tags* under which the operation falls in. The latter helps to keep operations organized within the API. In addition, we suggest to include two special files in the repository. The first is a `LICENSE` file containing the license for the SPARQL queries and the API. The second is the `endpoint.txt` file, with the URI of a default endpoint to direct all queries of the repository. When parsing the repository (see Section 4) the target endpoint will be the one indicated by the `#+ endpoint` decorator, the `endpoint.txt` file, or `http://dbpedia.org/sparql`, in this order of preference.

4 grlc

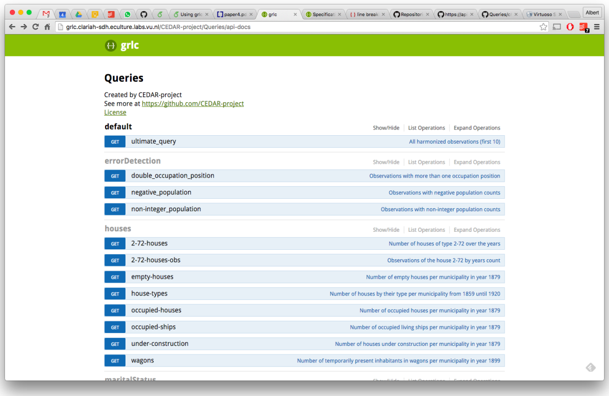
`grlc`[10] is a thin gateway that automatically builds complete, well documented, and neatly organized Linked Data APIs on the fly, with no input required from users beyond a GitHub user and repository name. To do so, it implements the GitHub API mappings proposed in Section 3.1, and uses the SPARQL decorators described in Section 3.2. It provides three basic operations: *(1)* generates the Swagger spec of a specified GitHub repository; *(2)* generates the Swagger UI to provide an interactive user-facing frontend of the API contents; and *(3)* translates `http` requests to call the operations of the API against a SPARQL endpoint with several parameters. If the GitHub repository at `https://github.com/:owner/:repo` contains decorated SPARQL queries, `grlc` uses these, together with organisational repo information from the GitHub API, to build the API interface automatically. Assuming that `grlc` is running in `:host`, these operations are available at the following routes:

- `http://:host/:owner/:repo/spec`: JSON Swagger-compliant specification, using the mappings of Section 3
- `http://:host/:owner/:repo/api-docs`: Swagger-UI, rendered using such mappings, as shown in Figure 1.
- `http://:host/:owner/:repo/:operation?p_1=v_1...p_n=v_n`: `http GET` request to `:operation` with parameters $p_1, ..., p_n$ taking values $v_1, ..., v_n$.

`grlc` composes the Swagger spec as follows: *(1)* the user requests the URI `http://:host/:owner/:repo/spec`[11] to a host running `grlc`; *(2)* `grlc` issues the `http GET` request to the GitHub API at `https://api.github.com/repos/:owner/:repo`, using the owner and repo names indicated in the previous step; *(3)* for each `.rq` file described in the response, `grlc` derefences `https://raw.githubusercontent.com/:owner/:repo/master/file.rq` to get the SPARQL file contents; *(3)* `grlc` parses these file contents to extract: *(a)* the values of

[10] Source code at `https://github.com/CLARIAH/grlc`; demo instance at `http://grlc.clariah-sdh.eculture.labs.vu.nl/`

[11] Requested from any `http` compliant client: a Web browser, `curl`, etc.



Queries

Created by CEDAR-project
See more at https://github.com/CEDAR-project
License

default — Show/Hide | List Operations | Expand Operations

GET	ultimate_query	All harmonized observations (first 10)

errorDetection — Show/Hide | List Operations | Expand Operations

GET	double_occupation_position	Observations with more than one occupation position
GET	negative_population	Observations with negative population counts
GET	non-integer_population	Observations with non-integer population counts

houses — Show/Hide | List Operations | Expand Operations

GET	2-72-houses	Number of houses of type 2-72 over the years
GET	2-72-houses-obs	Observations of the house 2-72 by years count
GET	empty-houses	Number of empty houses per municipality in year 1879
GET	house-types	Number of houses by their type per municipality from 1859 until 1920
GET	occupied-houses	Number of occupied houses per municipality in year 1879
GET	occupied-ships	Number of occupied living ships per municipality in year 1879
GET	under-construction	Number of houses under construction per municipality in year 1879
GET	wagons	Number of temporarily present inhabitants in wagons per municipality in year 1899

maritalStatus

Fig. 1. Screenshot of the Swagger user interface generated by `grlc`.

the decorators (if any), and *(b)* any parameter placeholders in the query; *(4)* `grlc` uses all the gathered data to compose the Swagger spec, and returns it to the client as JSON. The composition of the Swagger UI is analogous: first the JSON spec is composed and, after, it is used to render the Swagger UI template[12].

Operations of the form `http://:host/:owner/:repo/:operation?p_1=v_1...p_n=v_n`, are executed by `grlc` by first retrieving the raw query at `https://raw.githubusercontent.com/:owner/:repo/master/:operation.rq`, and then rewriting it using the parameter values supplied in the corresponding placeholders.[13]. After this, the query is submitted to the endpoint indicated by the methods described in Section 3.2, using supplied `http` headers (e.g. content-type). The endpoint results are forwarded to the client application.

5 Preliminary Evaluation

In this section we evaluate requirements satisfied by `grlc` in two use cases.

Dutch Historical Census Data. The CEDAR project[14] has published the Dutch historical censuses (1795–1971) as 5-star Linked Data [9]. Key queries and templates to interrogate this dataset are available at GitHub[15]. These queries are used in various client applications[16,17]. Before `grlc`, we decided to imple-

[12] `https://github.com/swagger-api/swagger-ui`

[13] `grlc` is compliant with BASIL's convention for Web API parameters mapping, see `https://github.com/the-open-university/basil/wiki/SPARQL-variable-name-convention-for-WEB-API-parameters-mapping`

[14] `http://www.cedar-project.nl/`

[15] `https://github.com/CEDAR-project/Queries`

[16] YASGUI-based browsing: `http://lod.cedar-project.nl/cedar/data.html`

[17] Drawing historical maps with census data: `http://lod.cedar-project.nl/maps/map_CEDAR_women_1899.html`

ment a minimal effort Web API using our own instance of BASIL[18]. However, the queries needed to be retyped in the system, and caused ramifications with respect to the ones in our existing applications. Moreover, it was not possible to mimic the organisation these queries had in the original GitHub repo in the API spec. After `grlc`, we could create this API without interfering with the original applications and queries, effectively reusing them. Furthermore, `grlc` permitted an ecosystem where SPARQL and non-SPARQL savvy applications coexist.

Born Under a Bad Sign. In CLARIAH[19], querying structured humanities data from combined sources is central. This particular use case focuses on validating the hypothesis that prenatal and early-life conditions have a strong impact on socioeconomic and health outcomes later in life, by using 1891 census records of Canada and Sweden. These were converted to Linked Data with QBer [6], and analyzed in the statistical environment `R`. Before `grlc`, loading the data to be analyzed implied the manual download of a SPARQL query resultset in a file, and then loading this file in `R`. This was mitigated with the `R` SPARQL package [5]. However, this resulted in hard-coded, hardly reusable, and difficult to maintain queries. After better organising these queries in a GitHub repository, an API using them became immediately available through `grlc`. The `R` code became *clearer* due to the decoupling with SPARQL; and *shorter*, since a `curl` one-liner calling a `grlc` enabled API operation sufficed to retrieve the data.

6 Conclusion and Future Work

In this paper we have presented `grlc`, a novel approach to automatically build Linked Data APIs by using SPARQL queries stored and documented in git repositories. Our approach addresses two pitfalls of current practice in constructing Linked Data APIs: *(1)* the coupling of SPARQL curation workflows and the API infrastructure, which hampers query reuse and forces query retyping and ramifications; and *(2)* the common lack of organisation in Linked Data APIs. `grlc` maps the Swagger specification with GitHub API features and a proposed SPARQL decorator notation, and builds and maintains Linked Data APIs automatically with minimal effort. We argue that this approach enables a better coexistence of SPARQL and non-SPARQL savvy applications, and allows developers to switch their efforts from API infrastructure to applications.

We plan to extend this work in several ways. First, we will support additional repository elements and SPARQL decorators. Second, we will add compatibility with other collaborative coding platforms, like Bitbucket and GitLab, enabling private APIs and authentication. Finally, we plan to create a `grlc` companion to facilitate the curation of SPARQL queries in git repositories.

[18] `https://github.com/the-open-university/BASIL`

[19] `http://clariah.nl/`

References

1. Daga, E., Panziera, L., Pedrinaci, C.: A BASILar Approach for Building Web APIs on top of SPARQL Endpoints. In: Services and Applications over Linked APIs and Data – SALAD2015 (ISWC 2015). vol. 1359. CEUR Workshop Proceedings (2015), http://ceur-ws.org/Vol-1359/
2. 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
3. Fielding, R.T.: Architectural styles and the design of network-based software architectures (2000)
4. Groth, P., Loizou, A., Gray, A.J., Goble, C., Harland, L., Pettifer, S.: API-centric Linked Data integration: The Open PHACTS Discovery Platform case study. Web Semantics: Science, Services and Agents on the World Wide Web 29(0), 12 – 18 (2014), http://www.sciencedirect.com/science/article/pii/S1570826814000195, life Science and e-Science
5. van Hage, W.R., with contributions from: Tomi Kauppinen, Graeler, B., Davis, C., Hoeksema, J., Ruttenberg, A., Bahls., D.: SPARQL: SPARQL client (2013), http://CRAN.R-project.org/package=SPARQL, R package version 1.15
6. 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
7. Huelss, J., Paulheim, H.: The Semantic Web: ESWC 2015 Satellite Events, chap. What SPARQL Query Logs Tell and Do Not Tell About Semantic Relatedness in LOD, pp. 297–308. Springer International Publishing, Cham (2015), http://dx.doi.org/10.1007/978-3-319-25639-9_44
8. McMillan, R.: From Collaborative Coding to Wedding Invitations: GitHub Is Going Mainstream. Wired Magazine (2013, February 9), http://www.wired.com/2013/09/github-for-anything/all
9. Meroño-Peñuela, A., Guéret, C., Ashkpour, A., Schlobach, S.: CEDAR: The Dutch Historical Censuses as Linked Open Data. Semantic Web – Interoperability, Usability, Applicability (2015), in press
10. Pedrinaci, C., Domingue, J.: Toward the next wave of services: Linked Services for the Web of data. Journ. of Universal Computer Science 16(13), 1694—1719 (2010)
11. Rietveld, L., Hoekstra, R.: Man vs. Machine: Differences in SPARQL Queries. In: Proceedings of the 4th USEWOD Workshop on Usage Analysis and the Web of of Data, ESWC 2014 (2014), http://usewod.org/files/workshops/2014/papers/rietveld_hoekstra_usewod2014.pdf
12. Saleem, M., Ali, M.I., Mehmood, Q., Hogan, A., Ngomo, A.C.N.: LSQ: Linked SPARQL Queries Dataset. In: The Semantic Web - ISWC 2015. LNCS, vol. 9367, pp. 261–269. Springer
13. Speiser, S., Harth, A.: Integrating linked data and services with linked data services. In: The Semantic Web: Research and Applications. pp. 170—184. Springer (2011)
14. Vandenbussche, P.Y., Aranda, C.B., Hogan, A., Umbrich, J.: Monitoring the Status of SPARQL Endpoints. In: Proceedings of the ISWC 2013 Posters and Demonstrations Track, 12th International Semantic Web Conference (ISWC 2013). pp. 81–84. CEUR-WS (2013)