Simplod: Simple SPARQL Query Builder for Rapid Export of Linked Open Data in the Form of CSV Files

Antonín Jareš
jares.antonin@gmail.com
Charles University

Department of Software Engineering, Faculty of
Mathematics and Physics
Prague, Czechia

Jakub Klímek
jakub.klimek@matfyz.cuni.cz
Charles University

Department of Software Engineering, Faculty of
Mathematics and Physics
Prague, Czechia

ABSTRACT

In the last decade, linked open data (LOD) became the de-facto highest standard of publishing data on the Web, a.k.a. 5-star open data. One of the advantages of LOD is better data discovery thanks to the linkable nature of the data. Unfortunately, in the wider IT expert community, RDF and SPARQL is considered unnecessarily complex, hard to understand and hard to process, especially when transferring open data in bulk. However, it is this wider IT expert community which is the one supposed to both publish data produced in the public administration as open data, and to build consumer-facing applications on top of the data. In this paper, we propose an approach to making LOD easily accessible to the community used to CSV files. We propose a tool called Simplod focused on rapid, straightforward and customizable formulation of a SPARQL SELECT query intended for customizable bulk transformation of LOD published in SPARQL endpoints into CSV files. In addition, Simplod configurations can be stored in and shared via Solid pods.

CCS CONCEPTS

• **Information systems** → **Web interfaces; Web searching and information discovery**; *Resource Description Framework (RDF); Ontologies*; • **Human-centered computing** → *Accessibility systems and tools*.

KEYWORDS

SPARQL, query builder, RDF, CSV, Solid

ACM Reference Format:

Antonín Jareš and Jakub Klímek. 2021. Simplod: Simple SPARQL Query Builder for Rapid Export of Linked Open Data in the Form of CSV Files. In *The 23rd International Conference on Information Integration and Web Intelligence (iiWAS2021)*, November 29–December 1, 2021, Linz, Austria. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3487664.3487790>

1 INTRODUCTION

In the last decade, linked open data (LOD) became the de-facto highest standard of publishing data on the Web, a.k.a. 5-star open data. It is backed by a family of Web standards such RDF for data

representation and SPARQL for data querying. While nowadays, it is not hard to find an understanding of the advantages of publishing data as LOD in the computer science, biochemistry and academia communities, this is unfortunately not the case in the wider IT expert community. However, it is this wider IT expert community which is the one supposed to publish data produced in the public administration as open data and to build consumer-facing applications on top of the published data.

In this wider IT expert community, RDF is considered unnecessarily complex, hard to understand and impractical to process, especially when transferring open data in bulk. It is then difficult to explain its advantages, including better data discovery thanks to the linkable nature of the data. A common complaint against RDF and SPARQL is that the users miss a data schema, an equivalent to XML Schema for XML, JSON Schema for JSON and CSV on the Web descriptor for CSV files. Note that the RDF Schema vocabulary is used for building other vocabularies, not for describing the structure of the published LOD, which typically reuses a number of different vocabularies. Members of the wider IT expert community then try to persuade the public administrations not to publish data as LOD and to revert to a more traditional format such as JSON or CSV. This in turn is a source of conflict with the more LOD focused community. It seems that a solution to this conflict is to publish data both as LOD and e.g. in CSV files. However, this will, quite naturally, again fail to satisfy all users of the data, because there are multiple options of publishing a dataset in CSV. One of those ways is publishing the data as one big redundant CSV file, usable e.g. in the machine-learning communities. Another way of publishing the data is as a set of normalized CSV files mimicking the structure of a normalized relational database. In both of those ways, the publisher is the one fixating the schema of the CSV file with no possibility of customization on the consumer's side.

In this paper, we propose a hybrid approach to making LOD accessible to the community used to CSV files. We propose a tool called **Simplod** focused on rapid, straightforward and customizable formulation of a SPARQL SELECT query intended for customizable bulk transformation of LOD published in SPARQL endpoints into CSV files. The tool works with a data schema extracted from data published in SPARQL endpoints, and offers two complementary views on the schema - a graphical one, and a list view. Using those views, a user can quickly select an interesting sub-graph of the data and receive it in the form of a CSV file.

Simplod aims at ease of use for bulk data transformations rather than at complete SPARQL language coverage or being a query

This work is licensed under a Creative Commons Attribution International 4.0 License.

iiWAS2021, November 29–December 1, 2021, Linz, Austria
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9556-4/21/11.
<https://doi.org/10.1145/3487664.3487790>

builder for specific question answering. The tool offers collaboration and configuration sharing via Solid¹ pods, and exporting the resulting CSV generating SPARQL query in multiple ways, e.g. as a URL, which can be used e.g. as a CSV distribution download URL in data catalogs, or a link to YASGUI [3].

The rest of this paper is structured as follows. In section 2 we briefly summarize related work. In section 3 we introduce our approach and the tool, Simplot. In section 4 we discuss the way the SPARQL SELECT query is generated. In section 5 we evaluate the tool with real users, and in section 6 we conclude.

2 RELATED WORK

In this section we provide a brief overview of similar approaches. RDF Explorer [5] is a query builder which might seem very similar at first sight, but the problem it solves is different from the goal of Simplot. The aim of RDF Explorer is specific question answering via a graphical UI based on iterative expansion of the query. The users can access e.g. the Wikidata endpoint² in a simple yet effective way and navigate through the dataset how they see fit, querying arbitrary data and imposing constraints on their properties. In contrast to Simplot, the users need to know where they are going in their iterations, as they cannot see the structure of the whole dataset at once.

Another query builder, VSB: Visual SPARQL Builder³ aims at providing similar solution to the problem area as Simplot. Unfortunately, not the solution itself nor the page provide any information on how to use the application. According to its webpage, the application allows users to visually create SPARQL queries against the DBpedia⁴ endpoint. The users should be able to fine tune the queries by selecting properties as optional, hiding properties from the result set or negating their existence. However, the solution seems to be able to provide only properties from the domain of the chosen endpoint, severely limiting its use. There is also no option to run the application with a custom endpoint without cloning the repository and deploying the application. The demo instance does not work anymore.

OptiqueVQS [4] loads an OWL ontology and allows users to visually build a SPARQL query using that ontology. However, a typical LOD source uses multitude of different vocabularies, not described by OWL. Finally, Sparklis [1] employs natural language understanding to build a SPARQL query, which is a different use case than Simplot aims at.

3 THE TOOL - SIMPLOD

In this section, the created tool, Simplot⁵, is presented. The description is split into individual steps starting from loading the dataset schema and finishing with saving the configuration in a Solid pod and sharing the CSV download link. Note that Simplot expects the published data to be available through a SPARQL endpoint capable of performing SPARQL SELECT queries. We will illustrate the tool on a running example about Nobel prizes and their laureates⁶.

¹<https://solidproject.org/>

²<https://query.wikidata.org/sparql>

³<https://leipert.github.io/vsb/>

⁴<https://dbpedia.org/sparql>

⁵<https://jaresan.github.io/simplod/>

⁶<http://data.nobelprize.org/store/sparql>

3.1 Input dataset schema

In addition to the SPARQL endpoint IRI, the other required input is a dataset schema to be used for the construction of the resulting SPARQL query. For our datasets, we used a LinkedPipes ETL [2] pipeline⁷, which can be run e.g. on the LinkedPipes ETL demo instance⁸. It takes a list of SPARQL endpoints on the input, and through SPARQL querying it extracts the dataset schema from each one, producing a zip file containing all the schemas.

1	?s rdf:type ?st .	1	<urn:uuid:XXX>
2	?s ?p ?o .	2	rdf:subject ?st ;
3	?o rdf:type ?ot .	3	rdf:predicate ?p ;
		4	rdf:object ?ot .

(a) Classes connected by predicate ?p

(b) Schema representation of classes connected by predicate ?p

Figure 1: Input dataset schema examples

For each RDF object ?t of an RDF statement ?s rdf:type ?t, an artificial entity instance ?new rdf:type ?t is created in the schema, representing the fact that the dataset contains instances of ?t. In addition, for each RDF predicate ?p of an RDF graph in Figure 1a an artificial representation in Figure 1b represents this relation type with its domain and its range. Finally, in the case of a datatype property where ?o rdf:type ?ot is missing in the dataset, we represent this as <urn:uuid:XXX> rdf:object rdf:Literal. An example in RDF Turtle syntax can be seen in Figure 2.

```

1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
3 @prefix nobel: <http://data.nobelprize.org/terms/> .
4 @prefix dbpedia: <http://dbpedia.org/property/> .
5
6 <urn:uuid:e6782822-087a-4fb7-bc3c-d3d88044bec>
7   rdf:subject <http://data.nobelprize.org/terms/Laureate/instance> ;
8   rdf:predicate foaf:name ;
9   rdf:object rdf:Literal .
10
11 <urn:uuid:c450b448-459c-48e2-ad3e-bec716c30813>
12   rdf:subject <http://data.nobelprize.org/terms/Laureate/instance> ;
13   rdf:predicate dbpedia:dateOfBirth ;
14   rdf:object rdf:Literal .
15
16 <urn:uuid:a99257d-8905-473a-80ff-c14d7822eeb>
17   rdf:subject <http://data.nobelprize.org/terms/NobelPrize/instance> ;
18   rdf:predicate nobel:laureate ;
19   rdf:object <http://data.nobelprize.org/terms/Laureate/instance> .
20
21 <urn:uuid:3d78a7db-f486-4415-91cd-b3c91a0131b>
22   rdf:subject <http://data.nobelprize.org/terms/NobelPrize/instance> ;
23   rdf:predicate nobel:category ;
24   rdf:object rdf:Literal .
25
26 <urn:uuid:9d744327-1d99-42f8-8ddc-fef3e320ba12>
27   rdf:subject <http://data.nobelprize.org/terms/NobelPrize/instance> ;
28   rdf:predicate nobel:year ;
29   rdf:object rdf:Literal .
30
31 <http://data.nobelprize.org/terms/NobelPrize/instance>
32   a nobel:NobelPrize ;
33   <http://data.nobelprize.org/terms/Laureate/instance> a nobel:Laureate .

```

Figure 2: Data schema example in RDF Turtle

The application requires the data schema to be provided in a form of a publicly accessible URL, so that the application configuration can be easily shared on the Web.

3.2 Graphical schema view

When the dataset schema is loaded, it is visualized in a UML Class diagrams style, which can be seen in Figure 3a.

The reader might have noticed that while in RDF, all predicates are directed, in the schema visualization view, the edges are not directed. This is by design as in fact, the edge in the visualization represents the whole group of predicates between the two classes to make the visualization less crowded, which may be in both directions.

In the graphical schema view, the user can select classes they are interested in, and their properties, simply by clicking on them.

⁷https://raw.githubusercontent.com/jaresan/lo-d-cloud/master/simplod_data_schema_pipeline.jsonld

⁸<https://demo.etl.linkedpipes.com/>

Figure 3: Dataset schema visualization in Simplod

In addition, the classes can be hidden from the view and deleted from the schema. For the case there is a need to have two separate instances of the same class in the query result, the class can also be duplicated. The downside of using an auto-laid-out schema view can be that user can get lost easier as the schema gets bigger. For those cases, there is a complementary list view.

3.3 Schema list view

Similarly to the graphical schema view, the user can see and work with the schema in a complementary list view as can be seen in Figure 3b. The list view displays all the classes and properties found in the dataset schema. It contains two basic parts - all available classes and properties, and selected classes and properties. In the list view, besides the usual selection operations, the user can see a human readable label of a class or a property, if available⁹ and rename the variables, which will later form the resulting CSV's column names. In addition, the user can reorder the variables here to set the order of the columns in the resulting CSV.

3.4 Generated SPARQL query

The next step in using Simplod is the generation of the SPARQL query itself. In Figure 4a, a simple example of a subset of a dataset schema can be seen. In Figure 4b, the corresponding generated query is visible in the embedded YASGUI [3] interface. From there, the query can be run on the original SPARQL endpoint, producing results, downloadable as a CSV file.

3.5 Sharing options of the resulting query

The resulting query configuration can be shared in multiple ways. Simplod generates a set of sharing links:

- **Link to YASGUI** with the SPARQL endpoint and the resulting query pre-filled
- **CSV URL** directly resulting in the CSV file by encoding the query in a request directed at the SPARQL endpoint. This URL is also usable e.g. as a CSV distribution URL in data catalogs

⁹Simplod dereferences the IRIs of classes and properties on-the-fly in order to get their human readable labels.

- **Web URL** resulting in the query being evaluated in the web interface of the SPARQL endpoint
- **cURL request** for evaluation of the query using the cURL¹⁰ command-line tool

3.6 Configuration sharing via Solid pods

The configuration of the project in Simplod can be saved in a Solid pod, and from there, shared with other Web users. Simplod contains a simple integrated Solid pod browser for storage, sharing and retrieval of project configurations, allowing users to reuse projects.

4 SPARQL QUERY GENERATION

In this section, we will highlight some of the interesting parts of the algorithm generating the resulting SPARQL query based on the input dataset schema.

The basic algorithm is straightforward, illustrated by Figure 4a and Figure 4b. On the input, it takes the sub-graph of the dataset schema selected by the user. For each selected class, a variable is created for its instances together with the corresponding `rdf:type` statement, e.g. `?Laureate rdf:type nobel:Laureate` . and `?NobelPrize a nobel:NobelPrize` .

Next, each datatype property of a class, including links to outside datasets, generates a corresponding triple pattern, e.g. `?Laureate foaf:name ?name` . Each object property of a class also generates a triple graph pattern linking two class instances, e.g. `?NobelPrize nobel:laureate ?Laureate` .

One common problematic situation often encountered by users is that they select classes without connecting them by an object property. This results in a query producing a Cartesian product in the result, i.e. combination of each instance of one type with each instance of another type, which is almost always undesirable. Simplod detects this situation and warns the user about it to prevent unnecessarily complex, useless queries to be executed on the SPARQL endpoints.

5 EVALUATION

We evaluated Simplod with real users by letting them go through the simple Nobel prize example, then a more advanced Nobel prize

¹⁰<https://curl.se/>

Figure 4: Schema subset selection and corresponding SPARQL SELECT query

example and, finally, another example regarding German books with configuration loading and storage in Solid pods. Next, we collected feedback from the users using the industry standard System Usability Scale (SUS)¹¹. We received answers from 4 users, three of who had no prior RDF experience, resulting in the slightly below-average SUS score of 63.125, where scores larger than 68 are average. The user with prior RDF experience achieved score of 72.5. The feedback included unintuitive tool controls and missing in-tool on-boarding experience in addition to introductory texts.

6 CONCLUSIONS

In this paper, we have introduced Simplod, a tool for simple graphical building of SPARQL SELECT queries based on dataset schemas. The approach and the tool is aimed at providing access to linked open data stored in SPARQL endpoints to users familiar with CSV. The primary focus of Simplod is to quickly provide a CSV data dump of all instances of certain classes, their properties and relations rather than to allow users to construct a question answering SPARQL query. The tool has been evaluated by 4 users unfamiliar with linked open data, and it scored a slightly below-average score. One known limitation of the approach is when the resulting CSV file would be too large, causing SPARQL endpoint timeouts. Another limitation becomes evident when the dataset schema is simply too complex, e.g. in the case of Wikidata. There, a cooperation of the data publisher is needed so that they pre-create and interesting sub-graph of the dataset for consumers to customize.

ACKNOWLEDGMENTS

This work was supported by the Czech Science Foundation (GAČR), grant number 19-01641S.

REFERENCES

- [1] Sébastien Ferré. 2017. Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web* 8, 3 (2017), 405–418. <https://doi.org/10.3233/SW-150208>
- [2] Jakub Klímeck and Petr Škoda. 2017. LinkedPipes ETL in use: practical publication and consumption of linked data. In *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2017, Salzburg, Austria, December 4-6, 2017*, Maria Indrawan-Santiago, Matthias Steinbauer, Ivan Luiz Salvadori, Ismail Khalil, and Gabriele Anderst-Kotsis (Eds.). ACM, 441–445. <https://doi.org/10.1145/3151759.3151809>
- [3] Laurens Rietveld and Rinke Hoekstra. 2017. The YASGUI family of SPARQL clients. *Semantic Web* 8, 3 (2017), 373–383. <https://doi.org/10.3233/SW-150197>
- [4] Ahmet Soylu, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ernesto Jiménez-Ruiz, Martin Giese, Martin G. Skjæveland, Dag Hovland, Rudolf Schlatte, Sebastian Brandt, Hallstein Lie, and Ian Horrocks. 2018. OptiqueVQS: A visual query system over ontologies for industry. *Semantic Web* 9, 5 (2018), 627–660. <https://doi.org/10.3233/SW-180293>
- [5] Hernán Vargas, Carlos Buil Aranda, and Aidan Hogan. 2019. RDF Explorer: A Visual Query Builder for Semantic Web Knowledge Graphs. In *Proceedings of the ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 26-30, 2019 (CEUR Workshop Proceedings, Vol. 2456)*, Mari Carmen Suárez-Figueroa, Gong Cheng, Anna Lisa Gentile, Christophe Guéret, C. Maria Keet, and Abraham Bernstein (Eds.). CEUR-WS.org, 229–232. <http://ceur-ws.org/Vol-2456/paper60.pdf>

¹¹<https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>