Bridges between GraphQL and RDF

Ruben Taelman, Miel Vander Sande, Ruben Verborgh

IDLab, Department of Electronics and Information Systems, Ghent University – imec

Abstract. GraphQL offers a highly popular query languages for graphs, which is well known among Web developers. Each GraphQL data graph is scoped within an interface-specific schema, which makes it difficult to integrate data from multiple interfaces. The RDF graph model offers a solution to this integration problem. Different techniques can enable querying over RDF graphs using GraphQL queries. In this position statement, we provide a high-level overview of the existing techniques, and how they differ. We argue that each of these techniques have their merits, but standardization is essential to simplify the link between GraphQL and RDF.

1. Introduction

This is a position statement for participation to the W3C Workshop on Web Standardization for Graph Data 2019 *(https://www.w3.org/Data/events/data-ws-2019/)*. This position statement lies within the topic of graph query languages, and focuses on the GraphQL [1] query language, and its link to the RDF [2] data model.

We focus on GraphQL, because it provides an easy-to-use and highly popular language for querying graphs. Unfortunately, these queries only have semantics in isolated servers. In order to add meaning *across* data sources, GraphQL queries can be aligned with the RDF data model.

In the next section, we will list existing techniques for using GraphQL to query RDF data. After that, we conclude with a proposal on a way forward to harmonize these approaches, which we hope to discuss during the workshop.

2. Existing Approaches

To the best of our knowledge, four solutions exist for querying RDF via GraphQL queries:

- HyperGraphQL [3] (*enterprise*)
- TopBraid [4] (*enterprise*)
- Stardog [5] (*enterprise*)
- GraphQL-LD [6] (*academic*)

The key properties of these solutions will be discussed hereafter.

2.1. HyperGraphQL

HyperGraphQL [3] is an open-source GraphQL interface wrapper around RDF sources, maintained by Semantic Integration Ltd. The main goal of this system is to support federated querying over multiple RDF sources and make it accessible through a single interface.

Each HyperGraphQL instance uses a configuration file and an *annotated* GraphQL schema. The configuration file is used to define the RDF services to fetch data from. The annotated GraphQL schema is a GraphQL schema that is annotated with GraphQL directives that indicate for each type and argument in which RDF service the type can be fetched, and what their full URI is.

Users can send plain GraphQL queries to an HyperGraphQL instance, and the instance will reply with a GraphQL response that is enhanced with a JSON-LD context [7]. This implicitly transforms the GraphQL response into RDF. Optionally, the server provides different content types to convert the response into alternative RDF serializations.

2.2. TopBraid

TopBraid [4] is a commercial set of semantic data governance tools by TopQuadrant. In a recent release, GraphQL was added as a way to query the stored RDF.

TopBraid's approach is similar to HyperGraphQL. It also uses GraphQL schemas to expose a GraphQL interface over RDF sources. TopBraid's schemas are automatically generated based on data shape definitions, which can be defined using SHACL [8].

These schemas are more expressive than HyperGraphQL's schema, as it supports things like SHACL-based constraint filters, full-text-search, aggregations, ordering and field transformations. These features are defined via custom query functions in the schema. In contrast to HyperGraphQL, TopBraid attaches no JSON-LD context to its GraphQL query responses.

2.3. Stardog

Stardog [5] is a commercial graph store that recently (as of release 5.3.3) adds GraphQL query support.

Stardog makes GraphQL schemas optional for querying RDF using GraphQL. If no schema is used, then GraphQL term to RDF conversion will be done using the default namespace in that graph, but this can be overridden by using the `@prefix` directive inside GraphQL queries.

Stardog assumes that the top-level node of any query is assumed to refer to a type. Any deeper query nodes refer to predicate links from the parent node.

Stardog also adds several features to the GraphQL query side. Compared to TopBraid, Stardog uses *directives* for this, instead of dedicated arguments. It adds support for functionality such as filters and bindings.

Just like TopBraid, no JSON-LD context is attached to its GraphQL query responses.

2.4. GraphQL-LD

Finally, GraphQL-LD [6] is an academic approach that is able to convert GraphQL queries to SPARQL [9], and SPARQL results to the corresponding GraphQL query results, independent of a query engine. As an example, GraphQL-LD was implemented in the Comunica engine [10], and can be tried out via a Web interface *(https://gist.github.com/rubensworks/9d6eccce996317677d71944ed1087ea6)*. Compared to the previously explained approaches, it does not use the GraphQL schema at all, but exploits JSON-LD contexts [7] to handle the conversion of GraphQL terms to RDF terms. This offers a separation of concerns where developers can use simple GraphQL queries, and domain experts can offer the JSON-LD context which *semantifies* these queries.

GraphQL-LD also supports custom features such as filtering and ordering through directives.

GraphQL query responses in this approach are plain JSON objects, and are compatible with the user-provided JSON-LD context. As such, responses are usable as both JSON and RDF.

3. Conclusions

In the previous section, we provided an overview of four approaches for querying RDF with GraphQL queries. These approaches are significantly different and mostly incompatible with each other. As such, one GraphQL query that works for one approach will most likely not work in the other approaches, or it can have different semantics.

Therefore, it is important to achieve a better understanding of the different ways of querying RDF with GraphQL. Ideally, and if possible, a consensus should reached on a single way.

In order to do this, we propose the following plan of action:

1. Determine the different methods of querying RDF with GraphQL.
2. Decompose and categorize the existing approaches in these methods.
3. Determine the advantages and disadvantages of each method.
4. Propose a standard approach and semantics for querying RDF with GraphQL.

Clarifying the differences and a path to unification will lower the barrier for developers that have knowledge on GraphQL to start querying RDF.

Biography

Ruben Taelman is a PhD student at IDLab, Ghent University – imec, Belgium. His research concerns the server and client trade-offs for Linked Data publication and querying, and has experience with RDF development in JavaScript such as implementing a graph-based query engine.

References

1. Facebook, I.: GraphQL. Working Draft, Oct. 2016. http://facebook.github.io/graphql/October2016/
2. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: Concepts and Abstract Syntax. W3C, https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (2014).
3. Semantic Integration Ltd.: HyperGraphQL. http://hypergraphql.org/
4. TopQuadrant: TopBraid Enterprise Data Governance. https://www.topquadrant.-com/products/topbraid-enterprise-data-governance/
5. Stardog Union: The Knowledge Graph Platform for the Enterprise. https://www.stardog.com/
6. Taelman, R., Vander Sande, M., Verborgh, R.: GraphQL-LD: Linked Data Querying with GraphQL. In: Proceedings of the 17th International Semantic Web Conference: Posters and Demos (2018).
7. Consortium, W.W.W., others: JSON-LD 1.0: a JSON-based serialization for linked data. (2014).
8. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL). W3C, https://www.w3.org/TR/2017/REC-shacl-20170720/ (2017).
9. Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language. W3C, https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (2013).
10. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: a Modular SPARQL Query Engine for the Web. In: Proceedings of the 17th International Semantic Web Conference (2018).