GraphQL-LD: Linked Data Querying with GraphQL

Ruben Taelman, Miel Vander Sande, Ruben Verborgh

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

Abstract. The Linked Open Data cloud has the potential of significantly enhancing and transforming end-user applications. For example, the use of URIs to identify things allows data joining between separate data sources. Most popular (Web) application frameworks, such as React and Angular have limited support for querying the Web of Linked Data, which leads to a high-entry barrier for Web application developers. Instead, these developers increasingly use the highly popular GraphQL query language for retrieving data from GraphQL APIs, because GraphQL is tightly integrated into these frameworks. In order to lower the barrier for developers towards Linked Data consumption, the Linked Open Data cloud needs to be queryable with GraphQL as well. In this article, we introduce GraphQL-LD, an approach that consists of a method for transforming GraphQL queries coupled with a JSON-LD context to SPARQL, and a method for converting SPARQL results to the GraphQL query-compatible response. We demonstrate this approach by implementing it into the Comunica framework. This approach brings us one step closer towards widespread Linked Data consumption for application development.

1. Introduction

The SPARQL query language is a W3C recommendation for querying RDF data. While this language has gained a lot of attention in the research domain, its widespread usage within commercial applications remains limited. One of the reasons for this is many developers are not experienced in the handling of (RDF) triples. Instead, they are better equiped to handle nested objects. Furthermore, more libraries and frameworks exist for the latter.

In order to bridge this gap between RDF and developers, several works have been proposed [1, 2] to simplify the definition of queries and the shaping of results. These approaches either only semantify the query results, or require a custom domain-specific language for defining queries.

GraphQL is a query language that has proven to be a popular among developers. In 2015, the GraphQL framework [3] was introduced by Facebook as an alternative way of querying data through interfaces. Since then, GraphQL has been gaining increasing attention among developers, partly due to its simplicity in usage, and its large collection of supporting tools. One major disadvantage of GraphQL compared to SPARQL is the fact that it has no notion of semantics, i.e., it requires an interface-specific schema. This therefore makes it difficult to combine GraphQL data that originates from different sources. This is then further complicated by the fact that GraphQL has no notion of global identifiers, which is possible in RDF through the use of URIs. Furthermore, GraphQL is however not as expressive as SPARQL, as GraphQL queries represent trees [4], and not full graphs as in SPARQL.

In this work, we introduce GraphQL-LD, an approach for extending GraphQL queries with a JSON-LD context [5], so that they can be used to evaluate queries over RDF data. This results in a query language that is less expressive than SPARQL, but can still achieve many of the typical data retrieval tasks in applications. Our approach consists of an algorithm that translates GraphQL-LD queries to SPARQL algebra [6]. This allows such queries to be used as an alternative input to SPARQL engines, and thereby opens up the world of RDF data to the large amount of people that already know GraphQL. Furthermore, results can be translated into the GraphQL-prescribed shapes. The only additional requirement is their queries would now also need a JSON-LD context, which could be provided by external domain experts.

In related work, HyperGraphQL [7] was introduced as a way to expose access to RDF sources through GraphQL queries and emit results as JSON-LD. The difference with our approach is that HyperGraphQL requires a service to be set up that acts as a intermediary between the GraphQL client and the RDF sources. Instead, our approach enables agents to directly query RDF sources by translating GraphQL queries client-side.

In the next section, we summarize the architecture of our approach and the SPARQL algebra translation algorithm. After that, we explain our demonstration in Section 3, after which we conclude in Section 4.

2. Approach

We define a GraphQL-LD query as a GraphQL query paired with a JSON-LD context. In this demonstration, we handle GraphQL-LD queries using two standalone modules:

- **GraphQL to SPARQL algebra**: Parses a GraphQL query and JSON-LD context to SPARQL algebra.
- **SPARQL results to tree**: Converts SPARQL query results to a tree structure.

These modules will be explained in more detail hereafter. We plugged these modules into the Comunica [8] framework in order to evaluate SPARQL queries.

Fig. 1 shows an overview of our approach, where GraphQL queries and JSON-LD contexts are passed to our GraphQL to SPARQL algebra module, the resulting SPARQL algebra is queried with Comunica, and the results are shaped with the SPARQL results to tree module.

Fig. 1: Flow of GraphQL queries and JSON-LD contexts to SPARQL algebra, to SPARQL results, and to a tree shape.

Just like Comunica, our modules are implemented in JavaScript, and are compatible with the API specification by the RDFJS W3C community group (https://www.w3.org/community/rdfjs/). This enables interaction between different JavaScript

applications that supports this API.

2.1. GraphQL to SPARQL algebra

The GraphQL to SPARQL algebra module is responsible for parsing a GraphQL query to SPARQL algebra, based on a JSON-LD context. The conversion algorithm is based on translating GraphQL's tree-based structures to links of triple patterns in SPARQL.

Triple patterns are however not sufficient to express all of GraphQL's features. For instance, GraphQL allows queries to contain reusable fragments. These fragments are reusable query blocks, which must be applied on a certain type. If this type does not match, then the fragment will be ignored. We consider fragments to use the left-join semantics, which translates to the `OPTIONAL` keyword in SPARQL. Furthermore, GraphQL queries also allow pagination with the `first` and `offset`, which we translate to SPARQL `OFFSET` and `LIMIT`.

This module is available under the open MIT license on GitHub (https://github.-com/rubensworks/graphql-to-sparql.js), where more information on the conversion process can be found.

2.2. SPARQL results to tree

The SPARQL results to tree module can convert SPARQL query results to a tree-based structure. This is done by splitting combining variables prefix-based, and aggregating results in a tree structure. In order to determine whether a certain variable binding should be seen as an array or a single value, we require a mapping to be passed inside the context that defines which variables are singular. If variables are not defined in this mapping, they are considered plural by default.

This module is also available under the open MIT license on GitHub (https://github.com/rubensworks/sparqljson-to-tree.js).

3. Demonstration Overview

In our demonstration during the conference, we will offer the live evaluation of a set of GraphQL-LD queries using the Comunica framework. All of these queries, together with a guide to run the demonstration can be found as a gist (https://gist.github.com/rubensworks/9d6eccce996317677d71944ed1087ea6).

For example, Listing 1 and Listing 2 contain respectively a query and context that will produce the results from Listing 3 when executing against a DBpedia endpoint.

```
{ label
  writer(label_en: "Michael Jackson")
  artist { label }
}
```

Listing 1: GraphQL query to find all bands that Michael Jackson has written a song for.

```
{ "label": "http://www.w3.org/2000/01/rdf-schema#label",
  "label_en": { "@id": "http://www.w3.org/2000/01/rdf-schema#label
  "writer": "http://dbpedia.org/ontology/writer",
  "artist": "http://dbpedia.org/ontology/musicalArtist" }
```

Listing 2: JSON-LD context for the query in Listing 1.

```
[ {
    "artist": { "label": "Barry Gibb" },
    "label": "All in Your Name"
  }, ... ]
```

Listing 3: Results of the GraphQL-LD query in Listing 1 and Listing 2.

4. Conclusions

In this work, we propose GraphQL-LD as a technique for combining the worlds of GraphQL and the Semantic Web. We provide an implementation of this approach, and demonstrate this with a set of example queries.

In future work, we intend to formalize our GraphQL-LD conversion algorithm. Furthermore, we intend to improve the way in which we determine which variables should be considered singular or plural. OWL's `InverseFunctionalProperty` or JSON-LD framing are potential options that we consider for this.

In summary, this work allows GraphQL developers to query the Linked Open Data cloud. But also Linked Data experts can use it an an alternative to SPARQL.

References

1. Mynarz, J.: sparql-to-jsonld. https://github.com/jindrichmynarz/sparql-to-jsonld
2. Lisena, P., Troncy, R.: Transforming the JSON Output of SPARQL Queries for Linked Data Clients. In: Companion of the The Web Conference 2018 on The Web Conference 2018 (2018).
3. Facebook, I.: GraphQL. Working Draft, Oct. 2016. http://facebook.github.io/graphql/October2016/
4. Hartig, O., Pérez, J.: Semantics and Complexity of GraphQL. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web (2018).
5. Consortium, W.W.W., others: JSON-LD 1.0: a JSON-based serialization for linked data. (2014).
6. Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language. W3C, https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (2013).
7. Semantic Integration Ltd.: HyperGraphQL. http://hypergraphql.org/
8. 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).