The Role of the GraphQL Framework within the Semantic Web

Simone Auriemma\textsuperscript{a}, Maria Angela Pellegrino\textsuperscript{a,*} and Carmine Spagnuolo\textsuperscript{a}

\textsuperscript{a} Dipartimento di Informatica, Università degli Studi di Salerno, via Giovanni Paolo II, 132, 84084 Fisciano (Salerno) Italy

E-mails: s.auriemma5@studenti.unisa.it, mapellegrino@unisa.it, cspagnuolo@unisa.it

Abstract. When Facebook publicly released the GraphQL framework in 2015, entrepreneurs and researchers working in the Semantic Web community felt GraphQL would rewrite the Semantic Web. This hypothesis is based on the fact that, on one side, the Semantic Web attempts to create a standard and machine-understandable format to describe and expose entities and relationships on the Internet and, on the other side, GraphQL might have an impact on the way data is created and consumed on the Internet. This article performs a systematic literature review on the exploitation of GraphQL in the Semantic Web to verify if and to what extent the academic and developers communities agree on the revolutionary role of GraphQL in the Semantic Web. We looked for academy and enterprise solutions published as or cited by peer-reviewed contributions on Scopus, ACM Digital library, and IEEE Explore, and developers’ contributions publicly available on GitHub. As a result, this article demonstrates that the academy, enterprises, and developers have experimented with using GraphQL for accessing Knowledge Graphs, integrating or replacing SPARQL, or as a SPARQL proxy, and how Semantic Web solutions can be exploited to author GraphQL solutions. However, GraphQL still requires time to gain an official and standard role in the Semantic Web. We summarise the potentialities and the limitations to guarantee a wider diffusion of GraphQL in the Semantic Web community.

Keywords: Literature review, GraphQL, Semantic Web, Academy, Industry, Developers

1. Introduction

GraphQL has powered Facebook’s mobile apps since 2012. After developing and using it internally for three years, in 2015, Facebook released a specification and a reference implementation of its GraphQL framework \cite{1}. GraphQL, conceived as an alternative to REST APIs \cite{2}, is a query language for APIs and a server-side runtime for fulfilling queries over existing data \cite{3}. It provides a complete and understandable description of the data in an API, lets clients ask precisely for what they need, and makes it easier to evolve APIs over time \cite{3}. While REST APIs require loading from multiple URLs, GraphQL APIs get all the required data in a single request \cite{3}. Since its release, GraphQL has gained significant momentum and is now available in many environments and used by teams of all sizes, such as Coursera, GitHub, Pinterest, Airbnb, Netflix, and Twitter.

The popularity of GraphQL in the Web engineering community has recently prompted several investigations into the integration of Semantic Web principles and general compatibility with Linked Data (LD) in the form of RDF data, such as recent works that explored the interoperability between GraphQL and RDF, moving from data in a SPARQL endpoint to results in the JSON format \cite{4}. The interest in exploring the opportunities to exploit GraphQL in the Semantic Web lies in the objective of the Semantic Web that intends to model and expose the information

\*Corresponding author. E-mail: mapellegrino@unisa.it.

1570-0844/$35.00 © 0 – IOS Press. All rights reserved.
on the Internet in a standardized, self-describing, and machine-readable format. Even if living up to the promise of the Semantic Web was never a stated objective of GraphQL’s inventors, GraphQL has the potential to impact the way data are created and consumed on the Internet, moving a step toward unifying data across the Internet in a way that is meaningful and machine-readable [5]. Such are the expectations in GraphQL’s abilities to make a substantial contribution to the realization of the principles of the Semantic Web that entrepreneurs working in the Semantic Web field have come to grasp that “GraphQL will rewrite the Semantic Web” [6], mainly supported by the evidence that GraphQL is a less complex query language than SPARQL to access and exploit Semantic Web data [7].

This article aims to verify if and to what extent the literature supports these statements. Hence, it analyzes the role of GraphQL in the Semantic Web by performing a systematic literature review of academic articles retrieved from Scopus, ACM Digital Library, and IEEE Explore, developers’ contributions publicly available on GitHub, and commercial solutions cited by the literature. It results in 32 projects classified according to the community (i.e., academia, industry, developers), the use of GraphQL, and the coverage of GraphQL peculiarities.

Section 2 defines the GraphQL framework; Section 3 overviews related work and historical underpinnings at the base of GraphQL; Section 4 summarises the methodology used to perform the review on the basis of this article; Section 5 overviews and classifies contributions; Section 6 discusses potentialities and limitations of GraphQL in the Semantic Web, and summarises what is missing to achieve wider exploitation of GraphQL in the Semantic Web community; then, the article concludes with final remarks.

2. The GraphQL framework

According to the official introduction to the GraphQL framework, GraphQL is a query language for APIs and a runtime for fulfilling those queries with existing data, interpreted as the environment in which your program executes. GraphQL provides a complete and understandable description of the data in the API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools. GraphQL is a promising approach to implementing web-based services centered on high-level abstractions, such as schemas, queries, and mutations.

GraphQL is a query language for graph-structured Web APIs, flexible and structured, designed to build client applications, their data requirements, and interactions, while query responses are returned in the JSON format [8]. Hence, GraphQL endpoints typically let users explicitly ask for the supported queries by the schema introspection feature, where queries are retrieved based on schema grammar with object type and field (property) definitions.

Essential within GraphQL is a schema that describes types, their attributes (fields), and the relationships of those fields to other types or datatypes. A GraphQL schema describes all the queries one can make in a GraphQL API and all the types they return. Informally, such a schema defines types of objects by specifying a set of fields for which the objects may have values; the possible values can be restricted to a specific type of scalars or objects. The GraphQL schema is strictly bound to the data source, and it might be manually defined or automatically extracted. Hence, a domain expert has to analyze a dataset, design the corresponding GraphQL schema, and map the dataset to the schema. Schema-building is done in the strongly-typed Schema Definition Language.

A trivial consideration is that when you create a GraphQL query, you expect a response. You get a response because of the way the GraphQL runtime executes your request. After being validated, a GraphQL server executes a GraphQL query thanks to the execution of functions or methods attached to each field by the GraphQL server developer. Hence, this developer is in charge of implementing GraphQL resolvers for all fields in a specific programming language [9]. Resolvers are per-field functions that, based on a parent object, arguments, and the execution context, are responsible for returning a result for that field. Hence, it describes the relationship between the defined GraphQL types/fields and the data sources [9]. The parent object contains the result returned from the resolver on the parent field or, in the case of a top-level query field, the root value passed from the server configuration. This argument enables the nested nature of GraphQL queries. While arguments are fields expected in the query, the context includes anything that should be considered when resolving the query, including authentication information. Resolvers cannot be included in the GraphQL schema language, so they must be added separately. The collection of resolvers is called the resolver map.
Finally, the GraphQL server returns a result that mirrors the shape of the requested query, typically as JSON. Queries and results are modeled as tree-based objects, formatted as JSON objects [10] with different levels of nested nodes. However, results might also be returned in the RDF format. Resolvers can return null or undefined that indicates the object could not be found; an array of results, a promise when resolvers do asynchronous actions like fetching from a database or backend API; a scalar or object value.

The GraphQL schema can support filtering and sorting options that require to be implemented in the corresponding resolver. Moreover, the GraphQL schema can model a single source or result from a merging phase of multiple sources, supporting federated queries. GraphQL federation allows you to set up a single GraphQL API, or a gateway, that fetches from multiple resources. While users will interact only with the GraphQL schema, they can implicitly query one or more resources according to what is needed. Resolvers are responsible for dealing with multiple sources and returning results coherent with the GraphQL schema. Besides accessing KGs in a read-only way, GraphQL can be configured to provide users with the mutation support to modify server-side data.

3. Related work

3.1. GraphQL evaluation in the Semantic Web Community

On the top of KGs, there must be an easy-to-use method for users and applications to extract this information [11], such as building a web service for users to access entities and relationships [12]. The Semantic Web is a valuable resource of data, but a structural gap exists that is limiting a broader consumption of RDF data by the community of Web developers [12, 13]. Data might be web-accessed via SPARQL endpoints [14], but it is uncontrolled and unpredictable, compromising the performance, the response time, and even the availability of the service [13]. Moreover, exposing data directly as RDF triples is not compatible with Web APIs based on HTTP and JSON [12, 15]. Several solutions have been proposed in the literature to simplify the consumption of RDF by web applications.

Beltline.js is a JavaScript library aiming to increase the adoption of RDF [16]. REST APIs have been proposed on top of semantic data sources [15, 17, 18]. The GraphQL framework represents an alternative to the REST APIs that exposes an explicit data model described in terms of the so-called GraphQL schema to retrieve only required data [19]. Moreover, GraphQL natively formats RDF data in JSON format [4]. The GraphQL specification does not define the assumed data model. Even if GraphQL is not meant for querying graphs, the reference to graphs becomes obvious: there are nodes (the types) and edges (the fields) that point to other nodes [8]. In fact, the specification implicitly assumes a logical data model implemented as a virtual, graph-based view over some underlying DBMS. Such a graph is a directed, edge-labeled multigraph in which each node has a type and properties [10].

Werbrouck et al. [8] compare open-source GraphQL-based solutions for querying the semantic web and SPARQL in terms of schema usage, updating functionality, intermediary servers, federated, type, and reverse querying. They focus on HyperGraphQL [20] and GraphQL-LD [21]. As GraphQL-LD uses a serverless and schemaless deployment, it is considered more flexible to implement in web applications. However, HyperGraphQL better support querying datasets by only providing access to data of interest.

Seifer et al. [22] compare the usage of graph query languages, i.e., SPARQL, Cypher, Gremlin, and GraphQL, in Java projects on GitHub. They compare the number of projects, contributors, and repository activity over time. SPARQL is the most comment graph query language, while the number of Java projects using GraphQL is limited even if it proliferates since its publication.

To deal with multiple and heterogeneous data sources data integration architectures may opt either for lifting all data sources to a unified graph repository, a.k.a. data warehousing integration, or for option for a virtual integration by creating a mediator tier where legacy data sources hold their data but are accessed through a virtual graph integration layer, such as a GraphQL middleware [23]. The virtual graph solution has the advantage that it does not require additional storage capacity nor licensing for a dedicated server. However, the access time is higher than having the data lifted to a graph repository.

Several authors compared RDF query languages by empirical evaluations. Candel et al. [24], while proposing SkIQL, a generic language to query schemas, performed a comparison between SPARQL, Cypher, and GraphQL. Cypher and SPARQL can query more complex data structures if compared with GraphQL, but GraphQL is easier...
to learn than SPARQL. It is likely due to the lower complexity of the language. Similarly, Chaves-Fraga et al. [9] propose the Morph-GraphQL framework to generate GraphQL resolvers and schema from declarative mapping rules modeled in R2RML. In evaluating Morph-GraphQL, the authors compared it with two equivalent Semantic Web approaches by implicitly comparing SPARQL and GraphQL using the LinGBM benchmark [25]. For big datasets, GraphQL significantly degrades performance if compared with alternative approaches based on SPARQL.

3.2. Is GraphQL a really original idea? - API mechanisms on Linked Data

Linked data offers a set of best practices for publishing, sharing, and linking data and information on the web. It is based on the use of HTTP URIs and semantic web standards such as RDF. For some web developers, the need to understand the RDF data model and associated serializations and query language (SPARQL) has proved a barrier to adopting LD. They face a steep learning curve before they can use the power provided by the underlying technologies [26]. There is scope for a standard way to provide simple RESTful APIs over RDF graphs to bridge the gap between Linked Data and SPARQL. Simple RESTful APIs are well-supported and understood by a large community of web developers. This specification aims to fill that gap by defining an easy-to-use and easy-to-deploy API layer that can act as a proxy for any SPARQL endpoint representing resources in the JSON, XML, or CSV format [26].

Metaweb Query Language (MQL) [27], rooted in Query by Example (QBE) [28], was proposed by Metaweb Technologies to query Freebase. Hence, MQL is also referred to as the Freebase query language. Freebase is a massive, collaboratively edited database of cross-linked data, which contains data harvested from sources such as Wikipedia, Notable Names Database (a.k.a. NNDB), Fashion Model Directory, and MusicBrainz, as well as data contributed by its users. The structured data was licensed under the Creative Commons Attribution License, and MQL, a JSON-based HTTP API, is provided to programmers for developing applications on any platform to utilize the Freebase data. Hence, MQL is a powerful API for making programmatic queries and incorporating knowledge from the Freebase database into any application and website where queries are served by a triplestore called Graphd, a C/Unix server, which processes commands in a simple template-based query language. Query and results are formatted as JSON as in GraphQL. On 16 December 2014, the Freebase team officially announced that the website and the application programming interface would be shut down by 30 June 2015 since Google would substitute Freebase API with the KG API and it would move Freebase data to Wikidata [29].

linked-data-api [26] seeks to develop APIs, data formats, and supporting tools to overcome technical barriers, including, but not limited to, accessing linked data via a developer-friendly JSON format. linked-data-api defines a vocabulary and processing model for a configurable API layer intended to support the creation of simple RESTful APIs over RDF triple stores. The API layer is intended to be deployed as a proxy in front of a SPARQL endpoint to support Linked Data publication, the provisioning of sophisticated querying and data extraction features without the need for end-users to write SPARQL queries, delivery of multiple output formats from these APIs, including a simple serialization of RDF in JSON syntax. Similarly, Linked Data Fragments [30] is a conceptual framework that provides a uniform view of all possible interfaces to RDF by observing that each interface partitions a dataset into its specific kind of fragments. It is based on the observation that while SPARQL endpoints can be easily exploited by clients but require high server costs, data dumps are expensive for clients and do not allow live querying on the Web. Linked Data fragments require minimal server effort and enable efficient client-side querying. Each Linked Data Fragment is characterized by a specific data selector (subject URI, SPARQL query), metadata (variable names, counts), and controls (links or URIs to other fragments). Users can execute (federated) SPARQL queries over datasets with a Triple Pattern Fragments client, available as standalone applications, libraries, and Web applications. Even if they represent still-alive solutions, the date of the last update back to 2020, giving way to workarounds like GraphQL.

4. Methodology

The leading research questions (RQs) on the basis of the literature review at the basis of this paper are as follows:

RQ1 What is the role of GraphQL in the Semantic Web?
RQ2 What are the potentialities and limitations in widely adopting GraphQL in the Semantic Web?

The literature review was conducted by in-depth reading, interpreting, and categorizing papers addressing the use of GraphQL in the Semantic Web. The aim was to develop a comprehensive understanding and a critical assessment of the knowledge relevant to this topic. The papers were searched using Scopus, ACM Digital Library, and IEEE Explore as sources. Used keywords included GraphQL, Semantic Web, and variations of those. To be less restrictive as possible, we look for keywords in all the available fields, both metadata, and the full-text. The search carried out on the databases is as follows: (ALL (graphql AND (“semantic web” OR “knowledge graph” OR “linked data” OR “linked open data” OR sparql))). We took into considerations English peer-reviewed articles published since the official and public release of GraphQL (2015–2022) and contributions published until the end of February 2023. By only considering Scopus, ACM Digital Library, and IEEE, a total of 254 papers were found fitting these criteria, 205 without duplicates. We considered relevant all the articles that describe, evaluate or discuss how GraphQL is used in the Semantic Web over Knowledge Graphs (KGs) or Linked (Open) Data (LD or LOD). Once removed out-of-topic articles, we reviewed and classified 16 articles.

Figure 1 summarises and schematically reports the criteria considered during the selection process on the basis of this literature review. We also verified the presence of GitHub projects concerning GraphQL and the Semantic Web by looking for the same keywords used in the systematic literature review described above. We identified 41 GitHub projects. Once removed duplicates (5 projects), GitHub projects attached to tools described in already reviewed articles (9 projects) and out-of-scope projects, we reviewed and classified 12 projects (see Figure 2).

Finally, we took track of enterprise solutions described in related work and in the bibliography of articles included in the proposed review. It results in the overview and classification of 32 projects. By taking into account GitHub projects and enterprise solutions, we identify contributions released by developers not belonging to academia, guaranteeing a more comprehensive vision of the role of GraphQL in the Semantic Web community.

5. GraphQL in the Semantic Web

This section replies to RQ1 by Overviewing the role of GraphQL in the Semantic Web. The contributions are first introduced and classified into clusters based on the way GraphQL has been integrated with semantic web solutions.
Clusters concern the exploitation of GraphQL as either the unique query mechanism or used as an alternative to SPARQL, and the use of GraphQL as a SPARQL proxy, and contributions that move from Semantic Web solutions to GraphQL components, such as contributions that starts from ontologies, SPARQL endpoint(s) content, or other formats to model Semantic Web data and authors GraphQL schemas and servers. Then, we discuss the proposed federation mechanisms and approaches to author a GraphQL schema.

**Contribution overviews.** Table 1 summarises all the described contributions pointing out GraphQL peculiarities described above supported by each solution. Contributions are split by the academic, industrial (referred to as commercial solutions), and developers’ communities. Table 2 reports developers’ or academic projects publicly available on GitHub. The following describes the four classes of contributions: solutions that only focus on authoring GraphQL components starting from Semantic Web-based solutions without caring about the exposure on a GraphQL interface over KG(s), GraphQL used as the unique access mechanism to KG(s), in a combination of SPARQL, or as a SPARQL proxy. While the last three clusters are disjoint, the first cluster reports contributions that can be used as starting point in proposing GraphQL servers over KG(s) as they support the creation of the GraphQL schema and, eventually, the resolvers without exposing the GraphQL APIs.

### 5.1. From Semantic Web-based solutions to GraphQL components

This cluster includes the solutions that target the definition of the GraphQL schema or the GraphQL server (i.e., GraphQL schema and GraphQL resolver(s)) as a preliminary step to expose RDF data with a GraphQL API. All the contributions proposing GraphQL APIs faced the GraphQL schema or server definition. However, the articles reported in this section only focus on the proposal of a draft or improved mechanism to move from available data to a GraphQL schema or server without detailing the GraphQL API exposure or providing access to it. Hence, these solutions are precursors to enable the proposal of a GraphQL interface on data modelled according to the Semantic Web technologies.

Some contributions only focus on the exposure of the GraphQL schema without caring about the GraphQL resolver(s) implementation. Hence, the GraphQL server developers have to define the GraphQL resolver explicitly, enabling the opportunity to implement a user-defined federation mechanism. This sub-class includes GraphQL metamodel [19], and Semantic GraphQL [48] that start from a set of ontologies to return a GraphQL schema.

While the GraphQL metamodel supports the semi-automatic generation of GraphQL schema from a given ontology, Semantic GraphQL [48] is an Open Source project that provides an API to convert any RDF, RDFS, and OWL-based ontologies into GraphQL objects.

Another sub-class in this cluster is represented by contributions that support developers in programming an entire GraphQL server by programming GraphQL resolver(s) besides its schema. GQLFTW [57] is an experimental project to, as verbatim reported in the GitHub project, have fun with GraphQL, SPARQL, and Wikidata. It is provided with a code generation tool to build strongly typed GraphQL servers in Go starting from a GraphQL schema.

API developers are in charge of defining resolvers to determine how to load data satisfying the GraphQL schema. Hence, they can directly access data sources or behave as a SPARQL proxy. To further support the developer in automatically developing a GraphQL server, Morph-GraphQL [9] generates GraphQL resolvers and schema from declarative OBDA mapping rules modeled in R2RML. This approach allows domain experts to use the generated schema and resolvers as the initial proof of concept that can be used to query datasets without the need for software engineers to develop a full-fledged GraphQL server. Software engineers may also benefit from our approach as they may also use Morph-GraphQL to generate the initial version of a GraphQL server instead of building it from scratch. Morph-GraphQL is available as CLI implementation and Docker image on GitHub under Apache License 2.0. However, the authors marked Morph-GraphQL as deprecated in its repository. Similarly, OBG-gen [54] is an Ontology-Based GraphQL Server Generation for Data Access and Integration. A GraphQL-based framework for data access and integration in which a global domain ontology drives the generation of a GraphQL server that answers requests by querying heterogeneous data sources. The core of this framework contains an algorithm to generate a GraphQL schema based on an ontology and a generic GraphQL resolver function as a library based on semantic mappings. The GraphQL schema generator takes an ontology as the input and then outputs a GraphQL schema, while resolvers are generated starting from the RML mappings file (in turtle format). Hence, the GraphQL server is set up using Ariadne and can serve databases or datasets queries posed by users.
Table 1
Overview of revised contributions, split by communities, sorted by year. ✓ stands for full support, ~ for partial support, - for not supported/reported.

<table>
<thead>
<tr>
<th>Contribution name</th>
<th>Auto schema extraction</th>
<th>Schema introspection</th>
<th>RDF-interpretable results</th>
<th>Filtering &amp; Sorting support</th>
<th>Mutation support</th>
<th>Federated queries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Academy</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphQL-LD [21]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>TROMPA [31]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mamoutova et al. [32]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GraphQL metamodel [19]</td>
<td>~</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Norton et al. [33]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UltraGraphQL [34]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Oakes et al. [11]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>COVID19-OBKG [35]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mohammed and Fiaidhi [36]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GraphChain [37]</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Punya [38]</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TALISMAN [39, 40]</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Open Research KG [41]</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>VESPUCCI v2 [42]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GraphSPARQL [43]</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Morph-GraphQL [9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Commercial solutions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TopBraid [44]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Stardog [45]</td>
<td>~</td>
<td>~</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GraphDB [46]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Developers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SemanticGraphQL [47]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Semantic GraphQL [48]</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>KG GraphQL [49]</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CubiQL [50]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
<td>-</td>
</tr>
<tr>
<td>HyperGraphQL [20]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>GraphQL enhancer [51]</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>KGGraph [52]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MANGQ [53]</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OBG-gen [54]</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Grappa [55]</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SPARQLess [56]</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GQLFTW [57]</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Grasp [58]</td>
<td>~</td>
<td>✓</td>
<td>-</td>
<td>~</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

An outlier in this cluster is GraphQL enhancer\(^1\) that is a publicly available GitHub project that enhances GraphQL responses with LD annotations. Hence, LD is used here during the reply formulation rather than during the GraphQL server authoring.

\(^1\)GraphQL enhancer: https://github.com/architolk/graphql-enhancer
5.2. GraphQL as unique KG query language

This cluster includes all the contributions that propose GraphQL as the unique access mechanism to the underlying data without providing them with a SPARQL endpoint, rarely focusing on the federation mechanism. In this case, GraphQL enables the opportunity to propose specialized and custom views on the underlying data, such as MANGQ [53] that configure a natural language interface to KGs via GraphQL. But a single outlier represented by KG GraphQL [49] that proposes an open-source GraphQL API for Google KG API, publicly available at https://graphqlbin.com/v2/nrj6FX, developers usually configure access to their own custom KGs. Hence, the proposed solutions can easily rely on curated and manual schema extraction and support mutations. These advanced functionalities are mainly implemented by exploited already available and well-known mechanisms such as Dgraph or Apollo Federation API. This cluster mainly includes thematic KGs related to domain-specific topics.

Healthcare is the most addressed topic spanning from the exposure of a KG, such as COVID19-OBKG [35], to healthcare systems, such as the one proposed by Mamoutova et al. [32] and Mohammed and Fiaidhi [36]. COVID19-OBKG [35] is an ontology-based KG and web service collecting data related to COVID-19. All the extracted entities, relationships, and attributes are stored in the Dgraph database, which supports GraphQL and returns the results in a dictionary. The continuous collection mechanism is also supported by the proposal of Mamoutova et al. [32], which is an ontology-based diagnostic model to perform continuous monitoring and diagnosing of a system state. Similarly to COVID19-OBKG, Mamoutova et al. rely on a Dgraph database which implements GraphQL+:

2GraphQL+: https://dgraph.io/docs
as a query language. Dgraph offers queries to fetch data, automatic schema introspection, which can be disabled according to the users’ needs, filtering and sorting features, and mutation support. In line with the topic addressed by the contributions described above, Mohammed and Fiaidhi [36] use the notion of mindmap to develop effective and standardized microservice-based care systems that utilize KGs and GraphQL graph-based technology. GraphQL behaves as a promising approach to implementing web-based services centered on high-level abstractions, such as schemas, queries, and mutations. It automatically builds a schema by detecting tables, columns, indexes, relationships, views, types, functions, and comments and exposes multiple legacy datasets and domain-specific knowledge by a unified view via the Apollo Federation API. Hence, it automatically deals with federated queries.

Changing domain-specific topic, TROMPA (Towards Richer Online Music Public-domain Archives)³ [31] defines a KG modeling the music library domain and a web application providing specialized views of the underlying KG using a graph-based data infrastructure in which searching is performed with the GraphQL language. Similarly, Haris et al. [41] propose Open Research KG⁴, a GraphQL-based federated query service that integrates multiple scholarly communication infrastructures (specifically, DataCite, ORCID, ROR, OpenAIRE, Semantic Scholar, Wikidata and Altmetric). They develop a novel web widget-based approach for presenting scholarly knowledge with rich contextual information. When users interact with the Open Research KG, the GraphQL Gateway poses sub-queries on each infrastructure by exploiting their REST APIs and returns a federated and integrated reply, with the possibility to filter the compared studies based on rich contextual metadata. Hence, it supports the federation mechanism, but it is user-defined manually. A similar approach to implement the federation mechanism can be found in Oakes et al. [11] that present an approach to digital twins construction which involves a layered KG architecture communicating with the organization’s data repositories, provided with a GraphQL interface.

While all the contributions cited above propose domain-specific approaches where the domain is fixed, Dorodnykh et al. propose the TALISMAN framework⁵ that implements an approach for the semi-automated formation of domain ontologies [39] or KGs [40] based on web tables. Hence, they address specific topics determined by data content, but it is a general-purpose approach that can be reused in different domains. With the use of the semantic annotation procedure, they extract ontology instances from table cells. The proposed approach is implemented in the form of a web-based tool, which supports user interaction with a target ontology schema for semantic annotation of tabular data. GraphQL is used to obtain sets of candidate classes, datatypes, and properties. Among general-purpose solutions, we can also report KGraph⁶, an Open Source project that lets users create a KG with hierarchical topic modeling provided with a GraphQL interface. The demo is publicly available at https://romitagl.github.io/kgraph/#/dashboard.

5.3. The synergy of GraphQL and SPARQL

This cluster includes academic and commercial solutions for data management making data accessible via a GraphQL API and a SPARQL endpoint. Proposed solutions span from the provision of an access interface to a single database, such as the VESPUCCI v2 compendium, to the proposal of a data management and data governance platform, such as TopBraid [44], GraphChain [37], the one proposed by Norton et al. [33] and Punya [38].

The VESPUCCI v2 compendium³ [42] is a comprehensive database of nearly all transcriptomic experiments performed on grapevines during the last 15 years (until December 2020). Raw data are collected and manually annotated, employing both domain-specific and general-purpose ontologies terms and RDF. Once normalized, data are stored in a database and made available by COMPASS, a front-end API based on GraphQL technology. COMPASS also offers several ways to retrieve information from metadata, such as the SPARQL query language. While other solutions provide data with both a SPARQL endpoint and a GraphQL interface, COMPASS makes data accessible via GraphQL, while SPARQL can query metadata.

³TROMPA: https://trompamusic.eu
⁴Open Research KG: https://orkg.org
⁵The TALISMAN framework: http://talisman.ispras.ru
⁶KGraph: https://github.com/romitagl/kgraph
⁷The VESPUCCI v2 compendium interface http://vespucci.fmach.it/compass and its documentation https://vespucci.readthedocs.io
Norton et al. [33] work in the same application domain, healthcare, by proposing a cloud-based solution for IoT to collect static and dynamic information from security devices within a hospital environment. Data are accessed by a cloud-based architecture built on Amazon Web Service (AWS) to support both SPARQL and GraphQL-based APIs. In particular, AWS AppSync is a managed service based on Apache Apollo that allows the definition of GraphQL-based API, supports automatic GraphQL schema extraction and introspection, and offers the possibility to query multiple databases, microservices, and APIs by a single GraphQL endpoint. Hence, Norton et al. rely on a commercial solution to automatically extract the GraphQL schema from available data and provide users with a data management platform where both GraphQL and SPARQL can be used to access data.

Similarly to AWS, TopBraid Enterprise Data Governance [44] is a commercial solution developed by TopQuadrant, designed as a data management and data governance platform. It automatically generates GraphQL schemas to expose a GraphQL interface over RDF sources. Even if it supports the federation mechanism, TopBraid produces an executable GraphQL schema starting from a SHACL shapes graph taken as input. Hence, the federation mechanism is manually managed by developers. One of the peculiarities and advanced usage of TopBraid is the possibility to define nested SPARQL queries within GraphQL queries, making the queries more explicit. Besides making access to the underlying data, TopBraid lets users also modify them, enabling mutation support via the GraphQL framework.

GraphChain [37] still focus on data management by dealing with the proposal of a blockchain-based framework. The framework forms the foundational technology for the Ontochain project, allowing the storage of data in the native semantic formats and implementing the blockchain mechanisms on top of semantic data. Data can be queried by a SPARQL endpoint, REST APIs, and GraphQL.

While all the solutions mentioned above only focus on data access or data management, Punya\(^8\) [38] let users accessing remote graph data via GraphQL and SPARQL while authoring LD-enabled mobile applications. Hence, data access via GraphQL API or a SPARQL endpoint is embedded within an open-source, web-based platform based on MIT App Inventor and lets end-users exploit LD in their mobile applications.

All the solutions mentioned in this cluster support schema introspection. Hence, by pointing a client at the GraphQL API, one can find out the available queries.

### 5.4. GraphQL as a SPARQL proxy

This cluster includes general-purpose approaches that propose a GraphQL API over one or multiple SPARQL endpoints. In all the contributions, GraphQL queries are converted into SPARQL queries and then run on the configured SPARQL endpoint(s). The first approaches to use GraphQL in the Semantic Web belong to this cluster, and the developers community proposed them as prototypes to make a SPARQL endpoint accessible via a GraphQL interface.

In this context, SemanticGraphQL [47] is a proof of concept introduced in 2016 to map the DBpedia SPARQL endpoint to a GraphQL API using the Sangria Framework. Also, Grappa [55] proposes a simple test application over DBpedia for exercising the GraphQL to SPARQL bridge relying on GraphQL-LD. GraphQL-LD [21] is a GraphQL-to-SPARQL approach that transforms queries written in GraphQL and enriched with a JSON-LD context to SPARQL queries and, vice versa, converts SPARQL results to a GraphQL query-compatible response. It requires a user-defined JSON-LD context which compensates for the lack of semantic support of GraphQL by storing the mapping between RDF terms of the database and GraphQL. It lets developers manually configure a federation mechanism. GraphQL-LD also supports custom features such as filtering and ordering through directives. GraphQL-LD has been implemented in the Comunica framework [67], and it is used for generating RDF and websites for disseminating semantic academic events [68].

Other solutions from academics, developers, and companies adopt a similar manual and user-defined schema extraction approach as the one implemented in GraphQL-LD. Among them, HyperGraphQL [20] is a Java-based open-source project maintained by Semantic Integration Ltd to provide users with a unique GraphQL interface over multiple RDF sources and returns results in JSON-LD or alternative RDF serializations. It requires a manually defined GraphQL schema with additional custom (i.e., tool-specific) annotations to link the exposed data with the

---

\(^8\)Punya: http://punya.mit.edu
underlying LD sources (SPARQL endpoints or RDF dumps). Each HyperGraphQL instance uses a configuration file and an annotated GraphQL schema. While the configuration file defines the RDF services to fetch data from, the annotated GraphQL schema indicates for each type and argument in which RDF service the type can be fetched and what their full URI is. Hence, the federation mechanism is manually defined. HyperGraphQL was introduced to expose access to RDF sources through GraphQL queries and emit results as JSON-LD. Hence, it implicitly transforms the GraphQL response enhanced with a JSON-LD context into RDF. Optionally, the server provides different content types to convert the reply into alternative RDF serializations. The same feature is also implemented in GraphQL-LD, providing users with RDF-interpretable results. Similarly, Grasp [58] has been proposed by the developers’ community as a bridge software that provides GraphQL endpoint wrapping SPARQL endpoints. It requires the definition of a GraphQL schema with some Grasp-specific notations. Once set up the GraphQL server, every GraphQL query is translated into SPARQL queries and sent to the corresponding SPARQL endpoints automatically. Then, SPARQL results are reformed to be returned to the client as a GraphQL reply.

In the direction of providing developers with a semi-automatic approach to perform the schema extraction, the commercial solution Stardog9 offers a graph database called the “Enterprise Knowledge Graph platform” provided both with a SPARQL and a GraphQL interface. It makes GraphQL schema optional. If no schema is used, the schema is automatically extracted from the available data.

The first approach to fully automatically extract the GraphQL schema from available data was proposed by the developers’ community in 2018 by proposing CubiQL10, a GraphQL service for querying multidimensional LD Cubes. It requires the configuration of a single SPARQL endpoint without supporting the federation mechanism. Similarly, SPARQLess [56] is a library that can discover the data schema within a SPARQL endpoint and subsequently construct an equivalent GraphQL schema. It then uses the generated GraphQL schema to host a GraphQL server, which processes incoming queries by translating them into SPARQL queries and querying the underlying SPARQL endpoint. SPARQLess provides a read-only view of the data without supporting mutations. Following the same philosophy to derive the GraphQL schema, Gleim et al. proposed UltraGraphQL [34], a fully automatic bootstrapping (i.e., setup and configuration) GraphQL endpoint for single or multiple existing RDF triple stores. Data are automatically extracted by actual data through configurable SPARQL 1.1 path queries and can then be used for the automatic generation of a corresponding GraphQL schema. GraphQL queries are translated to one or more SPARQL queries over distributed triple stores, and results are merged in a unified view formatted in JSON-LD. Also, GraphSPARQL [43] offers the possibility to automatically generate a GraphQL schema based on given ontologies in RDF/OWL format. GraphSPARQL is a middleware that allows accessing arbitrary SPARQL endpoints by using GraphQL. Even if the GraphQL schema is automatically generated, developers can manually extend the auto-generated schema by introducing additional types or fields. GraphSPARQL is the unique solution proposed by the academic and developers community to support the mutation mechanism, together with the commercial solution proposed by Ontotext, the “Ontotext Platform”[46]. Ontotext platform auto-generates a unified, fast, flexible, and scalable GraphQL API over multiple RDF KGs.

5.5. Overall picture

This section overviews the contributions of developers, academics, and industry to propose solutions integrating GraphQL and the Semantic Web. Table 1 demonstrates that all the communities are interested in proposing GraphQL-based solutions for the Semantic Web. Looking at the support of GraphQL peculiarities, most of the contributions support automatic schema extraction and schema introspection to simplify the generation of the GraphQL schema starting from RDF source(s) and support users in identifying the queries that can be performed on the GraphQL API. GraphQL is mainly exploited to retrieve results in JSON format to be directly consumed by web applications, while few solutions also return results in an RDF format, such as JSON-LD. It is worth noting that this feature might be masked by the direct consumption of results by web applications and user interfaces without explicitly reporting the used results’ format. While commercial solutions proposed by the industry fully support filtering and sorting features, it is rarely and only partially supported by the developers’ community. When partial

9Stardog: https://www.stardog.com

10CubiQL: https://github.com/Swirrl/cubiql
support is provided, developers support the filtering feature. The industry and academy exploit GraphQL not only as a querying mechanism but also to modify the server-side content, implementing mutation support. Thanks to the possibility of exposing multiple KGs with a unique interface, half of the contributions implement federated queries.

The majority of the academy and developers’ projects, 15 out of 26, are publicly available, mainly as Open Source projects (look at the License column of Table 2). There is no uniform project language, as made evident by the Lang column, demonstrating that GraphQL is not bounded to a specific programming language. However, web-based languages are the most common ones, demonstrating that GraphQL democratises the Semantic Web content to web developers without asking them to explicitly use SPARQL. Projects differ in size, spanning from proof-of-concept projects with few contributors and fewer dozens of commits, such as MANQG [53] and SemanticGraphQL [47], to wide and well-structured projects, with dozens of contributors and hundreds or thousands of commits, such as Open Research KG [64] and Punya [66]. By looking at the Last commit column, it is evident that half of the projects are continuously maintained and updated. This observation can be applied to both academy and developers’ projects.

GraphQL schema definition based on queried RDF data. The GraphQL schema is manually or (semi-) automatically reconciled with RDF data. Details for the automatism of the schema extraction is provided for each contribution in Table 1, column auto schema extraction. Hence, checkmarks are attached to fully automatic extraction mechanisms able to infer the GraphQL schema automatically by the configured data or ontologies, tildes are attached to semi-automatic approaches, while blanks are attached to approaches that manually perform the reconciliation. 17 out of 32 implements a semi-automatic approach, fully automatic in 14 out of 32 contributions.

Federation mechanism comparison. Details for the supported federation mechanism are provided for each contribution in Table 1, column federated queries. Federated mechanisms span from manual/user-defined to automatic.

– Manual federation mechanism. Semantic GraphQL [48] and GQLFTW [57] require API users to code the resolver themselves. Hence, developers can manually configure resolvers to query multiple data sources. Similarly, GraphQL-LD [61] transforms a given GraphQL query to a corresponding SPARQL query over arbitrary triple stores by employing a user-provided semantic JSON-LD context. TopBraid [44] takes a SHACL shapes graph as input and produces an executable GraphQL schema that supports queries against data in RDF graph databases. HyperGraphQL [20] is manually configured using a directive-annotated GraphQL schema, where each type and field in the schema is extended with a directive linking to the responsible service. OpenResearchKG [64] relies on a pre-configured GraphQL-based federated system that integrates multiple scholarly communication infrastructures.

– Semi-automatic federation mechanism. According to the GraphSPARQL [63] architecture, the query processor performs SPARQL queries over distinct RDF DB and merges results. However, GraphSPARQL requires as input distinct compatible ontologies. This approach is not fully automatic since developers have to guarantee that ontologies do not introduce ambiguities.

– Automatic federation mechanism. In the automatic federation mechanism, resolvers usually behave as a GraphQL gateway, such as in Grasp [58] or GraphDB [46]. To provide support for multiple RDF backend stores, UltraGraphQL [59] enriches the extraction query with service-specific information, and each service is queried separately. According to the open-source project documentation, UltraGraphQL realizes the federation mechanism through an internal service named ManifoldService. Mohammed and Fiaidhi [36] and Norton et al. [33] exploit the ApolloFederationAPI that relies on a gateway that receives GraphQL requests and routes them to the source of interest, intelligently unified in a unified schema.

The best approach is strictly related to specific needs. If we focus on the automatism level, UltraGraphQL [59], Mohammed and Fiaidhi [36] and Norton et al. [33] are ranked better. If we focus on the customization options, SemanticGraphQL [48], GQLFTW [57], GraphQL-LD [61], and TopBraid [44] are the best ones since they are based on a user-defined approach. If we focus on the exploitation of GraphQL peculiarities, contributions based on a GraphQL gateway are the best ones.
6. Discussion

This section summarizes the potentialities and limitations of GraphQL, replying to RQ2. Then, it discusses the promise of GraphQL to rewrite the Semantic Web by creating a parallel between limitations and potentialities. Potentialities and limitations are shortened by $L$ and $P$, respectively, followed by a progressive number to refer to them compactly. It is worth noting that some limitations are easily overcome by technological and technical solutions, while other limitations are intrinsic in the GraphQL framework and require a consistent effort to be mitigated or successfully overcome.

6.1. Potentialities of GraphQL

**P1 - Unified (virtual) data graph view.** In 1970s, Extract-Transform-Load workflows mainly adopted data warehousing solutions [69]. In the meantime, the wide adoption of web technologies, and the availability and heterogeneity of data, required a revision to the data warehousing techniques [69]. New data storage models and technologies are gradually adopted, including graph-based data models [23]. Moreover, Ontology Based Data Access (OBDA) approaches behave as a valuable alternative to materialized data warehouses. From this basic description of the GraphQL framework, the analogy with the OBDA architecture is clear. To setup a GraphQL server it is required i) a domain expert able to analyze the underlying datasets, propose a unified GraphQL schema, and describe how the source data sources should be mapped into it, and ii) a software developer that implements those mappings as GraphQL resolvers [69]. Hence, the GraphQL schema behaves as a mediator solution, providing heterogeneous sources with a unified access point.

**P2 - Get exactly what you need.** One of the slogans attached to the GraphQL specification is "ask for what you need, get exactly that". It solves one of the most common objections posed against SPARQL queries that are criticized for returning results containing unnecessary metadata and carrying duplicate information [12], often requiring additional parsing or transformation steps to allow for their usage in Web applications [70]. Furthermore, in the context of mobile application development, the size and structure of the results should be as compact and small as possible since network bandwidth and computational power are limited [71]. GraphQL has been designed for mobile and Web applications. It features a tree-like structure, allowing for the traversal of the underlying graph. The same syntax is mirrored by the result, reducing additional transformation, data redundancy, and size in comparison to SPARQL [34].

**P3 - Results in a JSON format.** The triple-based output of SPARQL endpoints can be a barrier for web developers who want to integrate LD in their applications [12]. In 2013, W3C recommendations suggested formatting SPARQL query results in JSON [?] to simplify the consumption of the data by Web and non-Web applications. The format consists of a set of all possible bindings (of the form $<\text{variable}, \text{value}>$) that satisfies the query [12]. It implies a graph-oriented data structure rather than nested objects that (web) applications can immediately consume. Taelman et al. [4] realized interoperability between the GraphQL language and RDF, performing in this way a conversion in JSON of the data in an endpoint. The GraphQL syntax allows the production of a JSON object with different levels of nested nodes.

**P4 - Web developers’ friendly solutions.** SPARQL has proven to be a very powerful querying language, but it is cumbersome and almost unknown for developers that are used to JSON and REST APIs [43, 72]. Due to the lack of proper ontology descriptions or example queries of SPARQL endpoints, retrieving data from a KG is time-consuming and requires a steep learning curve for beginners [8, 43]. Hence, Semantic Web technologies are inaccessible by front-end developers [7].

It justifies the need for substantially less powerful but far more developer-friendly technologies such as GraphQL [7]. The advantage of the initiatives at the basis of this survey is that no real LD knowledge is needed to develop a service that consumes KGs. In fact, GraphQL implements a layer of abstraction that hides the complexity of SPARQL while keeping as much as possible flexibility [43].
6.2. Limitations of GraphQL

L1 - GraphQl is less expressive than SPARQL as queries written in GraphQL represent trees [73], and not full graphs as in SPARQL [21].

L2 - The cost of the generation of a GraphQL schema. A GraphQL schema is strictly bound to the data source [9]. Hence, it cannot be defined once and reused on multiple data sources. Moreover, it requires a manual effort and domain experts able to analyze the underlying data source(s), design the corresponding GraphQL schema, and map the dataset(s) to the schema [9].

L3 - No native notion of semantics. Popular GraphQL implementations are not directly suited for LD applications since they typically implement neither RDF-compatible data serializations nor global identifiers by default [21], rendering them incompatible with fundamental Semantic Web principles [34]. In a nutshell, GraphQL has no native notion of semantics [21].

L4 - No native support of federated queries. Unlike RDF, which is designed to link many domains together with triples, GraphQL is built around the traditional idea of isolated servers providing domain-specific data [16]. To uniformly access heterogeneous data, some state-of-the-art methods proposed GraphQL extensions to support federated queries, such as federated GraphQL [74], or a KG federation, such as Ontario [75].

6.3. GraphQL and the promise of rewriting the Semantic Web

It is worth recalling that potentialities and limitations have been shortened by L and P, respectively, followed by a progressive number to refer to them compactly. In the following, we use the notation $\text{L}_x \leftrightarrow \text{P}_y$ to create a parallel between limitations and potentialities underlying that overcoming limitations $\text{L}_x$, researchers can benefit from $\text{P}_y$ potentialities.

$L_1 \leftrightarrow P_3$ and $P_4$. Semantic Web technologies play an increasingly important role in linking social, machine, and transactional data, which continues to grow rapidly. However, the RDF data model of LD and SPARQL are unfamiliar to most developers. GraphQL is increasingly popular among Web developers [34] as it is a query language designed to build client applications where query response are structured according to the widely used JSON-format [8]. Since it went open source in 2015, GraphQL has been adopted by a large community of developers, partly due to its elegance and conciseness as GraphQL lowers the entry barrier to LD for developers without prior experience with Semantic Web technologies [16, 21, 34]. In recent years GraphQL as an abstraction of SPARQL has evolved, welcoming contributions by the academic, commercial, and developers’ community, accessing RDF data directly or behaving as a SPARQL proxy. It is worth stressing that the use of GraphQL comes with a tradeoff; it is less expressive than alternative RDF query languages, such as SPARQL, but much easier to use and adopt [32, 72]. Still, it allows achieving many of the typical data retrieval tasks in applications [21].

Mitigations to $L_2$. The tool to be paid is the definition of the GraphQL schema, which is strictly bound to the data source and requires a domain expert able to analyze the underlying data source(s), design the corresponding GraphQL schema, and map the dataset(s) to the schema [9]. Fundamentally, classes in RDF may be mapped to object types in GraphQL, while RDF properties might become GraphQL fields of those object types, which may occur as part of their RDF domain. Hence, a GraphQL schema can be used to describe RDF structures. This effort can be mitigated by approaches to automatically generate a GraphQL schema from RDF data such as the one proposed in TopBraid, GraphSPARQL [43], UltraGraphQL [34], GraphQL metamodel [19].

$L_4 \leftrightarrow P_2$. Popular GraphQL implementations are not directly suited for LD applications since they typically implement neither RDF-compatible data serializations nor global identifiers by default, rendering them incompatible with fundamental Semantic Web principles [8, 34]. A principal challenge is the schemaless nature of RDF and users’ definition of schema encodings, which makes it difficult to create a universal approach to make LD queryable through GraphQL and its static data schema [34]. GraphQL has no native notion of semantics [21]. Its semantics is limited to the schema that is used [8]. However, when extended with a mapping that relates the necessary definitions to a universal context, it can be used to describe data in a linked and coherent way and to query, update and delete information [8]. Initiatives such as GraphQL-LD [21], and HyperGraphQL extends GraphQL with LD querying
functionality, so it is possible to fetch the desired and necessary data from the distributed RDF building model. The approach proposed by TopBraid is using GraphQL with RDF/SHACL to provide powerful access to RDF data, combining a JSON-centric query language and W3C standards for schema definition, data validation, inferencing and query [44].

\[ L_4 \iff P_1 \]

GraphQL is built around the traditional idea of isolated servers providing domain-specific data [16]. It makes combining multiple sources even more difficult [21]. The GraphQL middleware has been defined as a gateway interface. It defines a GraphQL schema that comprises entities from different data sources into a single entity graph, allowing the execution of the queries across the services behind the virtual graph implicitly defined by the GraphQL schema [23]. The GraphQL middleware is much slower than accessing KGs directly with SPARQL. However, it creates a unified view of multiple sources and exposes it by a unique endpoint. For instance, UltraGraphQL lets users extract data from multiple RDF backend stores, then, during the GraphQL query translation phase, each service is queried separately, and results are opportunistically merged to create a unified result across all supported triple stores [34]. GraphQL API resides on a server that acts as a proxy to heterogeneous backends, including databases, APIs, and other GraphQL implementations. Moreover, a GraphQL is that the schema is represented as nodes and edges of property graph [36]. Hence, independently from the original data structure, the GraphQL schema exposes them as a graph. GraphQL also offers subscribers real-time notifications from the server. This feature is of high importance in a distributed environment, where different nodes can be in a slightly different states due to the consensus mechanism. GraphQL subscriptions make it easy to notify users when a node is in a synchronized state and safe to consume data from it.

7. Conclusion

The semantic web is undoubtedly an invaluable source of data, but hardly consumed by the community of Web developers [12, 13]. GraphQL is perceived as a promising approach to lower the entry barrier to LD for developers without prior experience with Semantic Web technologies [16, 21, 34]. While entrepreneurs feltGraphQL would rewrite the Semantic Web, this review quantifies the effort invested by the academic, commercial, and developers’ community in proposing approaches to integrate GraphQL and the Semantic Web to verify to what extent they agree on the role of GraphQL in impacting how data are created and consumed on the Web.

Since its official release in 2015, the commercial, academic, and developers’ communities invested in GraphQL solutions spanning from proofs-of-concept to well-established projects. For instance, it is already supported by some of the major graph database providers, like GraphDB or Stardog. The use of GraphQL comes with a trade-off: it is less expressive than SPARQL but much easier to use and adopt [72]. Hence, it has the potential to democratize access to RDF data through a wider range of web and mobile applications.

GraphQL is an API to query knowledge, not data [36]. Hence, by paying the toll of defining a GraphQL schema, it is possible to have a unified (virtual) data graph view over RDF source(s). Furthermore, results can be easily formatted as a JSON object to be immediately consumed by Web applications. Moreover, GraphQL avoids unnecessary metadata and duplicates if compared with SPARQL.

All in all, it has the potential to welcome the exploitation of data authored according to the Semantic Web technologies by a wider community by removing the constraint to access RDF data only by SPARQL. The Semantic Web community is aware of this opportunity, as demonstrated by the interest in organizing a W3C working group [76]. However, some limitations still require to be addressed. While some limitations summarised in this contribution can be easily solved and a consistent effort has already been invested in mitigating them, others are still open directions. In particular, the cost of generating a GraphQL schema and the limited expressivity of GraphQL should be carefully considered to let GraphQL gain a more standard role in the Semantic Web community.

References


Semantic Web


D. Strebel, SemanticGraphQL, 2016. https://github.com/danistrebel/SemanticGraphQL.


R. Almar, GraphQL metamodel, 2017. https://github.com/genesis-upc/Ontology2GraphQL.


M. Meitinger, GraphSPARQL, 2020. https://github.com/meitinger/GraphSPARQL.


[80] F. Priyatna, D. Chaves-Fraga, A. Alobaid and O. Corcho, Morph-GraphQL: GraphQL servers generation from R2RML mappings (2019).


