The Highway to Queryable Linked Data: Self-Describing Web APIs with Varying Features

Laurens De Vocht, Miel Vander Sande, Joachim Van Herwegen,
Ruben Verborgh, Erik Mannens, and Rik Van de Walle

Multimedia Lab – Ghent University – iMinds
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium
firstname.lastname@ugent.be

Abstract. Making Linked Data queryable on the Web is not an easy task for publishers, for technical and logistical reasons. Can they afford to offer a SPARQL endpoint, or should they offer an API or data dump instead? And what technical knowledge is needed for that? This demo presents a user-friendly pipeline to compose APIs for Linked Datasets, consisting of a customizable set of reusable features, e.g., Triple Pattern Fragments, substring search, membership metadata, etc. These APIs indicate their supported features in hypermedia responses, so that clients can discover which server-provided functionality they understand, and divide the evaluation of SPARQL queries accordingly between client and server. That way, publishers can determine the complexity of the resulting API, and thus the maximal set of server tasks. This demo shows how publishers can easily set up an API with this pipeline, and demonstrates the client-side execution of federated SPARQL queries against such APIs.

Keywords: Linked Data, self-descriptive APIs, querying, SPARQL

1 Introduction

Querying live Linked Data on the Web used to be a story of two extremes, with *Linked Data documents* and the *SPARQL protocol* on opposite sides of the spectrum. Despite the advancements in SPARQL query techniques, current numbers[1] indicate a rather low presence of queryable Linked Data. The limitations of Linked Data traversal and the limited availability of existing public SPARQL endpoints call for exploring other client–server trade-offs. Recently, Linked Data Fragments [8,9] were introduced to analyze such trade-offs by proposing a uniform view on all interfaces to RDF data. This view reveals a complete spectrum between Linked Data documents and SPARQL endpoints to publish Linked Data. Each possible interface in this spectrum sends its own kind of responses, which are characterized by three parts:

Data, determined by which selectors a server allows (e.g., SPARQL query, triple pattern);

Metadata, extra triples about the data (e.g., statistics, provenance);

Controls, guidance on how to access the data (e.g., forms, links).

[1] *http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state*

For example, the Triple Pattern Fragments (TPF) interface [8] responds to a client request for a triple pattern with a paged document containing matching triples (*data*), the estimated total number of matches (*metadata*), and a form to retrieve other TPFs (*controls*). Complex SPARQL queries can be evaluated on the client side by requesting multiple TPFs, and using the data, metadata, and controls inside of TPF responses sent by the server. A higher query execution time and bandwidth usage compared to SPARQL endpoints are accepted in exchange for a minimal server load. This interface thereby strikes a more sustainable load balance between clients and servers.

By defining new *features* for such interfaces that vary among the *data* and *metadata* dimensions, we can realize new balances of client–server trade-offs. Allowing more complex data selectors makes individual requests more expensive for the server, but the client might be able to evaluate SPARQL queries faster and with less bandwidth. Additional metadata in responses might enable clients to plan query executions more efficiently, at the cost of a server-side preparation step. Setting up such interfaces, however, is currently a difficult task for Linked Data publishers, and it is not straightforward for them to decide which features they want to enable or disable, especially if features change over time.

To this end, this demonstration extends two of our ISWC2015 research track papers [6,7] by proposing a *pipeline* which can *i)* publish a dataset as a queryable Linked Data API fast and easily, and *ii)* customize the mix of supported features on demand. We will demonstrate the applicability of this pipeline through an *in-browser federated SPARQL client* that dynamically discovers supported features through self-describing controls in APIs created by this pipeline. Dynamic feature support is exemplified by the following features:

- a data feature for **substring search** from the corresponding research track paper [6];
- a metadata feature for **approximate membership functions** (e.g., Bloom Filters) from the corresponding research track paper [7];
- a metadata feature for **dataset summaries** [5].

The self-descriptiveness allows clients to dynamically discover and use features if they are offered by a server, and to ignore metadata or controls it does not recognize. This means that feature-rich clients can consume single-feature APIs, while single-feature clients can also consume feature-rich APIs—regardless of which features are supported on either side. This contrasts with hard-coded client–server contracts, in which servers sometimes unilaterally decide not to implement a certain part of a specification.

In the remainder of this paper, we first present the publication pipeline in more detail (Section 2). Then, we explain how clients use the self-descriptive features to execute SPARQL queries (Section 3), and conclude with the demonstration setup (Section 4).

2 Queryable Linked Data API feature pipeline

Many data dumps on the Web have quality issues, and a significant portion of them contain RDF syntax errors [1]. Quality issues can be a source of frustration, as they require restarting data ingestion when errors are found. The first pipeline step is therefore to repair data automatically to the extent possible. We achieve this using a component of the LOD Laundromat [1] that takes care of refining a dataset and removing invalid or bad triples. The Laundromat guarantees that data conforms to a specified set of best practices, thereby greatly improving the chance of data actually being (re)used.

Once data has been cleaned, it needs to be converted into one or more (indexed) formats that support the given features. The pipeline chooses formats based on performance and server impact, and can combine multiple formats if needed to satisfy a certain feature combination. For instance, triple-pattern searches might be supported by one format, whereas full-text searches are covered by a dedicated index. Below, we discuss some of the available features.

Triple Pattern Fragments The TPF interface allows clients to decompose basic graph patterns of SPARQL queries into more elementary triple patterns, providing metadata for planning. This interface could be realized by a triple store, which—being designed for more complex patterns—often involve too much overhead for such simple patterns. Therefore, the pipeline will likely opt for the compressed HDT format [2], which requires less server resources to look up triple patterns. The pipeline covers the entire HDT generation process, which currently consists of manual steps and choices that are rather difficult for data publishers.

Substring search There are multiple ways to support substring search on the server. Either the pipeline generates an HDT file with a dedicated literal index (an FM-index), or an Elasticsearch index is populated [6]. This choice can be influenced by the desired performance and the availability of an Elasticsearch instance on the server. The cumbersome process of setting up either option is handled by the pipeline and can be switched on demand. When activated, this feature introduces a new kind of supported requests, allowing clients to request a list of all literal objects that contain a given substring. This can greatly reduce the cost of queries that use textual features such as regular expressions, at the cost of more expensive requests.

Membership metadata Membership metadata provides a client with the possibility of locally checking whether certain triple patterns are present on the server [7]. This allows for smarter decision making during query execution since the membership of a lot of triples can be verified without actually having to contact the server, thereby reducing the number of required HTTP calls. For example, servers can send approximate membership metadata with Bloom Filters or Golomb codes to compactly indicate whether a triple pattern has potential matches [7]. Another possibility are dataset summaries [5], which contain URI authorities of subject and object, partitioned by predicate. The trade-off is that metadata needs to be prepared on the server side and that individual responses can become larger. The pipeline can provide these features by pre-generating summaries. This requires extra storage space and processing time, so the pipeline lets a Linked Data publisher decide whether this is acceptable and/or desired.

3 Self-descriptive API publication and consumption

The premise of the feature-based publication approach is that data publishers decide which features they offer on the server. Therefore, clients need a means to discover what features a server supports in order to split a SPARQL query appropriately into HTTP requests. Our multi-feature federated SPARQL client takes as parameters a SPARQL query and a list of URLs to RDF interfaces. To evaluate the SPARQL query, it *i)* requests each of the interface URLs through HTTP `GET`; *ii)* looks inside of the responses for *control* triples which describe the features of the interface; *iii)* splits the SPARQL query based on the features of the

interface that both the client and the server support. This highlights the necessity of explicit feature description on the server side, which the pipeline automatically sets up.

As is the case for the TPF interface, each of the features are described in API responses through the Hydra Core Vocabulary [4]. This vocabulary can be regarded as the RDF equivalent of HTML's `<a>` and `<form>` tags, which similarly inform human users of the "features" a website supports. In contrast to implicit static contracts, such in-band hypermedia controls dynamically inform clients about how to use the interface [3]. For example, when an interface supports substring search through Elasticsearch, but the index is rebuilding, it can make substring search temporarily unavailable. Also, publishers can switch off certain features at peak moments, or try new features if they have spare server resources. In either case, clients can adapt by interpreting the controls in the response. With the same mechanism, more simple clients can safely ignore unrecognized features.

4 Demonstration

The demonstration at ISWC2015 will allow people to set up a custom API through a user-friendly pipeline that allows them to select the features they want to offer. Visitors will be able to bring their dataset, which will then be automatically transformed and loaded into a dedicated server during their visit at the booth.

Afterwards, they can execute federated SPARQL queries on their and others' datasets using the in-browser client. This multi-feature federated client is available online[2] with example SPARQL queries. For instance, the federated query at *http://bit.ly/cubist-works* is evaluated using DBpedia and VIAF interfaces set up by the pipeline. We will show how the client can use these new features to improve SPARQL queries, and how it handles multiple endpoints with a heterogeneous feature set.

References

1. Beek, W., Rietveld, L., Bazoobandi, H.R., Wielemaker, J., Schlobach, S.: LOD laundromat: a uniform way of publishing other people's dirty data. In: ISWC (2014)
2. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). Journal of Web Semantics 19, 22–41 (2013)
3. Kjernsmo, K.: The necessity of hypermedia RDF and an approach to achieve it. In: Proceedings of the Workshop on Linked APIs for the Semantic Web (May 2012)
4. Lanthaler, M., Gütl, C.: Hydra: A vocabulary for hypermedia-driven Web APIs. In: Proceedings of the 6th Workshop on Linked Data on the Web (May 2013)
5. Saleem, M., Ngomo Ngonga, A.C.: HiBISCuS: Hypergraph-based source selection for SPARQL endpoint federation. In: ESWC (2014)
6. Van Herwegen, J., De Vocht, L., Verborgh, R., Mannens, E., Van de Walle, R.: Substring filtering for low-cost Linked Data interfaces. In: ISWC (2015)
7. Vander Sande, M., Verborgh, R., Van Herwegen, J., Mannens, E., Van de Walle, R.: Opportunistic Linked Data querying through approximate membership metadata. In: ISWC (2015)
8. Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., Van de Walle, R.: Querying datasets on the Web with high availability. In: 13th International Semantic Web Conference (Oct 2014)
9. Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., Van de Walle, R.: Web-scale querying through Linked Data Fragments. In: Linked Data on the Web (Apr 2014)

[2] *http://client.linkeddatafragments.org/* (source code: *http://github.com/LinkedDataFragments/*)