# Towards Cost-model-based Query Execution over Hybrid Linked Data Fragments Interfaces

Amr Azzam[1], Ruben Taelman[2], Axel Polleres[1]

1 Vienna University of Economics and Business, Vienna, Austria, amr.azzam@wu.ac.at

[2]IDLab, ELIS, Ghent University – imec, ruben.taelman@ugent.be

**Abstract.** A multitude of Linked Data Fragments (LDF) server interfaces have been proposed to expose Knowledge Graphs (KGs) on the Web. Each interface leads to different trade-offs when clients execute queries over them, such as how query execution effort is distributed between server and client. There is however no single *silver bullet* that works best everywhere. Each of these interfaces has diverse characteristics that vary the performance based on server load, client resources, and network bandwidth. Currently, publishers can only pick one of these interfaces to expose their KGs on the Web. However, in some cases, multiple interfaces may be suitable for the publisher, and these may even vary over time based on the aforementioned factors. As such, we propose a hybrid LDF interface that can expose multiple interfaces based on a server-side cost model. Additionally, we sketch a negotiation protocol through which clients can determine desirable interfaces during query planning using a client-side cost model. In this paper, we lay out the high-level ideas behind this hybrid framework, and we explain our future steps regarding implementation and evaluation. As such, our work provides a basis for exploiting the trade-offs that exist between different LDF interfaces for optimally exposing KGs on the Web.

## 1. Introduction

The rapid growth of open and decentralized Knowledge Graphs over the Web has created an immense demand for public Knowledge Graph query services. However, serving live queryable Knowledge Graphs on the Web is difficult due to the low availability [1] and expensive hosting of SPARQL endpoints. As an alternative, publishing data dumps moves query effort to the client, but this may not always be desirable. Recently, the Linked Data Fragments (LDF) [1] framework was introduced to explore the range of Web query interfaces that exist between SPARQL endpoints and data dumps that distribute query execution load between clients and servers.

Several approaches have emerged following this framework such as Triple Pattern Fragments (TPF) [1] and Bindings-Restricted TPF (brTPF) [2], SaGe [3] and smart-KG [4], each offering their own trade-offs. For instance, TPF and brTPF increase server availability at the cost of increased network load. SaGe enhances average query performance at the cost of increased server load for concurrent complex queries. smart-KG increases server availability at the cost of higher client effort. Research shows that no single optimal approach exists, but they all have their advantages and

disadvantages. As such, there is a need for a hybrid LDF approach that determines one or more efficient query approaches based on changing circumstances.

A preliminary hybrid LDF approach [5] investigated the diversity of LDF characteristics [6] that can influence query execution plans. Another proposal [7] provides a different interface based on the current server workload. None of the aforementioned hybrid approaches allow clients and the server to negotiate query interfaces depending on factors such as the executed query, server load, client capabilities, and network bandwidth. In this paper, we propose a negotiation-based hybrid. Using a server-side cost model, the server can expose one or more query interfaces based on its current load, and the query that the client aims to execute. Using a client-side cost model, an efficient query plan over the available interfaces can be determined. This combination of server and client cost model ensure efficient usage of server and client resources to aim for the best possible query performance over the available LDF approaches.

## 2. Hybrid Framework

The goal of our framework is to expose different server interfaces based on the server load and the queries. Instead of exposing *just one interface* per query, we expose a *collection of interfaces* per query. This allows clients to select a combination of interfaces based on their capabilities, query plans, and other circumstances.

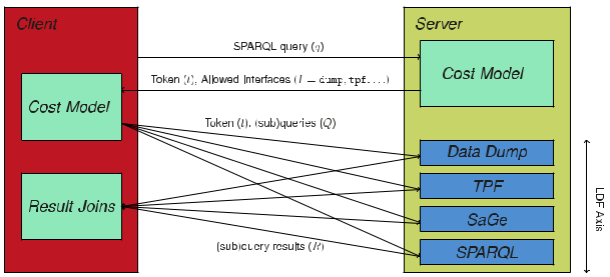


**Fig. 1:** Overview of client-server communication for a cost-model-based query execution over a hybrid of Linked Data Fragments interfaces.

To achieve such a server interface hybrid, a server cost model selects a set of interfaces based on a given query and the server current load. While a client cost model determines a query plan based on the granted interfaces. Fig. 1 shows an overview of this framework where client-side query engines start by sending a query q to the server, and receive an answer that contains a token t and a set of allowed interfaces I. Based on the returned interfaces, the client can determine a query plan over these interfaces. These (sub)queries can then be resolved by requesting the appropriate interfaces using the given token.

### 2.1. Server Component

The server component of our framework consists of a cost model for calculating a set of allowed interfaces, and a token-based wrapper over a set of interfaces.
**Cost Model** The goal of this server-side cost model is to ensure the server availability, and to allow queries to be executed as fast as possible. Since the latter goal can sometimes be detrimental to the server availability, for example when many concurrent users are sending highly complex queries, availability must always have priority. Based on these goals, the model should be able to make a suggestion for a set of interfaces based on a given query and a set of internal metrics. For this, we propose a set of internal metrics such as the current CPU usage, memory usage and network I/O. The threshold for these metrics can be configured so that the cost model can estimates the set of interfaces that optimize both goals.
Listing 1 shows the pseudocode of an algorithm that can be used to calculate a set of allowed interfaces. GetValueIncrease would still need a concrete implementation. For this, different possibilities exist, such as heuristics to predict query execution effort based on the number of triple patterns and query operators.

```
GetInterfaces(q, metrics, interfaces, GetValue, GetThreshold)
  allowedInterfaces = []
  FOREACH interface IN interfaces
    validInterface = true
    FOREACH metric IN metrics
      increase = GetValueIncrease(metric, q, interface)
      IF GetValue(metric) + increase > GetThreshold(metric)
        validInterface = false
    IF validInterface
      allowedInterfaces.push(validInterface)
  RETURN allowedInterfaces
```

**Listing 1:** Algorithm for calculating the allowed interfaces for a given query.

**Interface Wrapper** Based on the server-side cost model, the server can wrap over a number of LDF interfaces that the publisher wants to expose. This wrapper is a proxy that accepts SPARQL queries, and replies with a token and a set of granted interfaces that have been estimated for the given query. The token is *required* for performing any requests to any of the wrapped LDF interfaces. This token should be seen as temporary *permission* to make use of a specific set of query capabilities from the data publisher. The server must validates this token upon every request to an LDF interface to prevent the clients from ignoring the set of allowed interfaces and execute queries using the most expressive interface (e.g. SPARQL endpoint).

### 2.2. Client Component

Usually, the goal of clients is to execute queries as fast as possible. There could however be a number of metrics that can soften this need for fast query execution such as reducing CPU, bandwidth usage or optimizing for early results [8]. Using our server-side hybrid of LDF interfaces, clients will retrieve a set of allowed interfaces based on

given query. With respect to the client resources, the client should determine an efficient query plan based on the granted interfaces capabilities. While most client-side query algorithms focus on decomposing queries for execution against a single type of interface, additional algorithms are needed for *intelligently combining interfaces* for certain subqueries [5]. Another metric that influences the selection is the location of dataset fragments, locally [4] or within a network of peers [9].

## 3. Conclusions

This article outlines our high-level framework. We plan to implement the server and client components, and evaluate different cost models. The client component will be implemented using the Comunica platform [10] as a *Mediator* that can determine optimal interfaces based on the current metrics. This enable us to focus on the cost model of the client, as Comunica supports the majority of the LDF interfaces and SPARQL query operators. Our envisioned cost-model-based framework for enabling query execution over hybrid LDFs is a key element in achieving the vision of a Web where any client can query data over any combination of heterogeneous interfaces.

## References


1. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web. Journal of Web Semantics. (2016).
2. Hartig, O., Buil-Aranda, C.: Bindings-Restricted Triple Pattern Fragments. In: Proc. of ODBASE. pp. 762–779 (2016).
3. Minier, T., Skaf-Molli, H., Molli, P.: SaGe: Web Preemption for Public SPARQL Query Services. The World Wide Web Conference. 1268–1278 (2019).
4. Azzam, A., Fernández, J.D., Acosta, M., Beno, M., Polleres, A.: SMART-KG: Hybrid Shipping for SPARQL Querying on the Web. In: Proceedings of The Web Conference 2020. pp. 984–994. Association for Computing Machinery, New York, NY, USA (2020).
5. Montoya, G., Aebeloe, C., Hose, K.: Towards Efficient Query Processing over Heterogeneous RDF Interfaces. In: DeSemWeb@ISWC (2018).
6. Montoya, G., Keles, I., Hose, K.: Analysis of the Effect of Query Shapes on Performance over LDF Interfaces. In: QuWeDa@ISWC (2019).
7. Khan, H.: Towards More Intelligent SPARQL Querying Interfaces. In: International Semantic Web Conference (2019).
8. Acosta, M., Vidal, M.-E., Sure-Vetter, Y.: Diefficiency metrics: measuring the continuous efficiency of query processing approaches. In: International Semantic Web Conference. Springer (2017).
9. Grall, A., Skaf-Molli, H., Molli, P.: SPARQL Query Execution in Networks of Web Browsers. In: DeSemWeb (2018).
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. pp. 239–255 (2018).