Proceedings of the 10th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'24)
Barcelona, Spain - August 19 - 21, 2024
Paper No. CIST 143
DOI: 10.11159/cist24.143

Using Linked Data to Build Semantic Web Applications: A Case Study

Neli Zlatareva, Vincent Capra, Sharayu Khedekar, Ramya Sree Satyavarapu
Central Connecticut State University
1615 Stanley Street, New Britain, CT 06050, USA
zlatareva@ccsu.edu; caprav@my.ccsu.edu; sd.khedekar@my.ccsu.edu; ramyasrees@my.ccsu.edu

Abstract – Linked data applications offer a promising avenue to harness the power of the Semantic Web. However, realizing this power involves addressing a variety of technical, data-centric, and user experience challenges. This paper presents *MusicFans*, a real-world application developed as part of a case study intended to assess the practicality and effectiveness of using the *Linked Open Data* (LOD) cloud within a specific domain while deliberately avoiding the need for developing local ontologies and data repositories. Two primary challenges that we have encountered in the development of MusicFans were: i) establishing the application's domain scope due to the limited LOD cloud resources providing usable SPARQL endpoints, and ii) crafting a user interface that enables users not familiar with the SPARQL query language to place queries. The later challenge is compounded by the fact that SPARQL queries must align closely with the structure and the semantics of the underlying dataset schemas. We advocate that currently the most viable approach to tackle this challenge is to rely on a pre-defined library of dynamically generated SPARQL query templates. The paper delves into the design of these templates and discusses how they facilitate user interaction with the application in diverse real-world scenarios.

***Keywords*:** Semantic Web, SPARQL, RDF datasets, Linked Open Data cloud, DBPedia, Wikidata

1. Introduction

The World Wide Web is an integral part of our daily life. The incorporation of Large Language Models (LLMs) like ChatGPT into web browsers marks a considerable advancement in enhancing user experiences on the internet and broadens the scope and quality of services offered online. However, the effectiveness of these models in web browsing largely depends on the specific browsing tools they are integrated with and the defined parameters for web interaction. Currently, this equates to traditional browsing within the web of documents, where information is predominantly unstructured and tailored for human readers.

In contrast, the Semantic Web (SW), an idea proposed by Tim Berners-Lee [1], was developed to extend access to web content to computers by providing a universal language for information exchange, called the *Resource Description Framework* (RDF) [http://www.w3.org/RDF]. Furthermore, RDF also serves as a model for data representation on the SW allowing seamless integration of various datasets through different types of semantic relationships. This interlinking done using Uniform Resource Identifiers (URIs) allows for more effective data discovery across multiple datasets. These datasets comprise the *Linked Open Data* (LOD) cloud. Many of them are accessible via SPARQL endpoints, which allow for complex queries promoting data discovery, integration, and reuse across different applications and services. In addition, this new representation paradigm supports automated reasoning based on the relationships and rules defined within the data. All this, as originally envisioned by Tim Berners-Lee "[would] bring structure to the meaningful content of web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users" [2].

The bottleneck of this technology is that users must be familiar with the *SPARQL Protocol and RDF Query Language* (SPARQL) [http://www.w3.org/TR/rdf-sparql-query/] to place a query. Even the most user-friendly SPARQL endpoints, such as *DBPedia* [https://dbpedia.org/sparql], suggest some basic knowledge on writing SPARQL queries. More sophisticated queries, or federated queries over multiple SPARQL endpoints, require in-depth familiarity with SPARQL. This difficulty is well recognized by the Semantic Web community and various techniques were suggested to address it. These techniques can be divided into two categories: information extraction techniques and semantic parsing techniques. The former aim to identify the main entities of the user's query and map them to ontology relations, most commonly by using pre-defined or automatically generated templates [3, 4, 5]. Semantic parsing techniques extract the meaning of the query by converting it into a syntactic structure [6, 7, 8]. The main difficulty in this process is recognizing user intention expressed in a natural language query so that it can adequately be translated into a SPARQL query. LLMs could potentially offer a new way to address this difficulty by assisting users in formulating SPARQL queries by interpreting their natural language questions and suggesting corresponding SPARQL syntax. This would require the LLM to have knowledge of SPARQL syntax but also to have a deep understanding of the structure and schema of the RDF dataset being queried. At this point, we are not aware of any attempt to use LLMs in SPARQL query generation.

Our previous experience with semantic parsing suggests that it works well if the underlying RDF schema (classes, properties, and relationships within the data) is well understood and static which is achievable if the parser works with a custom-built ontology [8]. Complex queries, involving multiple datasets, nested queries, or aggregations, are challenging for semantic parsers.

In this paper, we present a case study intended to assess the accessibility and usability of LOD cloud in building SW applications intentionally avoiding the creation of local ontologies and data repositories. This suggests that the only practical approach to support user interface is to use pre-defined SPARQL templates for query generation. Since SPARQL queries are intricately linked to the structure of the underlying dataset schema, the templates must be carefully designed to reflect the semantics of the specific problem domain they are intended for. This ensures that the queries are both relevant and effective in retrieving the desired data from the LOD cloud. For our case study we chose the music domain and created an application, called MusicFans, to support targeted data retrieval and aggregation of data from multiple LOD datasets.

The paper is organized as follows. In Section 2, we discuss available music resources on LOD cloud and the scope of MusicFans application. Section 3 introduces the SPARQL templates library, offering an in-depth examination of select query templates to showcase the nature of user interaction with the application. Section 4 summarizes some of the challenges encountered in the development of Semantic Web applications relying on the power of the LOD cloud. We conclude with brief statement of our future plans related to this project.

2. Music Domain on LOD Cloud and an Overview of the MusicFans application

The music domain seemed especially promising for our case study due to the substantial amount of music related information on the LOD cloud. Unfortunately, some of the datasets such as *MusicBrainz* and *BBC Music* which we hoped to utilize turned out to be not viable for our purposes. The access to the *MusicBrainz* dataset, for example, is provided through a *LinkedBrainz* project [https://wiki.musicbrainz.org/LinkedBrainz], which is currently unsupported and out of date. Still, *DBTune* SPARQL endpoint [dbtune.org/musicbrainz/snorql] provides an interface to the old *MusicBrainz* pre-NGS dataset schema which is hard to navigate due to substantial changes in how data was structured and stored. SPARQL endpoint of *BBC Music* is not available at present. This left us with the two core LOD repositories, *DBPedia* [9] and *Wikidata* [https://guides.library.ucla.edu/semantic-web/wikidata]. Both repositories have SPARQL endpoints and support federated queries. DBPedia which is one of the most well-known and used linked datasets, is a semantic version of Wikipedea and also includes links to other LOD cloud datasets. Wikidata, unlike DBPedia, allows users to directly input and edit data, similar to how Wikipedia operates. This suggests that Wikidata can include structured data not just from Wikipedia but also from other sources.

The general overview of MusicFans architecture is shown on Figure 1.

The front-end of the application includes a comprehensive suite of modules designed to facilitate user interaction such as a *review* module that enables users to provide feedback on the application's responses, a *history* module that tracks users' interactions with the application, and *a library of dynamically generated SPARQL query templates* that facilitate user interaction with the application.

The interface between the application's front-end and backend is implemented using Python libraries such as SPARQLWrapper and RDFLib. These tools are used to process and execute user queries. The user's query, based on its nature, is first transformed into a SELECT or CONSTRUCT query by the chosen query template and aligned with the specific requirements of the RDF data being accessed. The execution of the query is handled by SPARQLWrapper, which enables the application to send the formatted query to DBPedia and/or Wikidata SPARQL endpoints. The results from the SPARQL query are then collected and returned to the user via an HTML template.

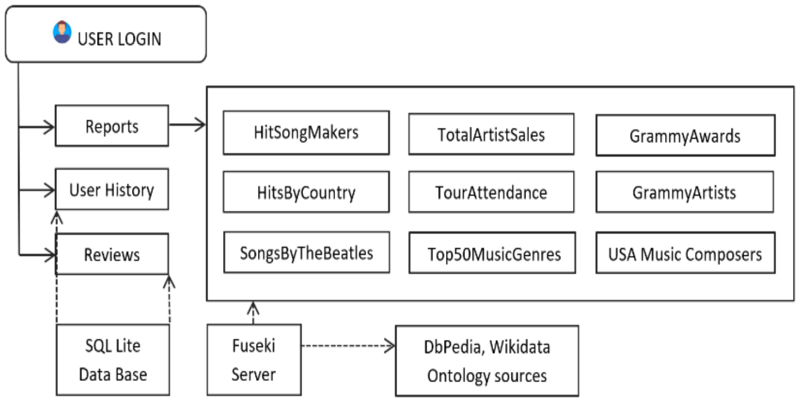


Figure 1. MusicFans Architecture

The true power of Semantic Web applications comes from the ability to carry out federated queries. Unfortunately, it turned out that implementing federated queries on either Wikidata or DBPedia SPARQL endpoints was not attainable. DBPedia been hosted as an OpenLink Virtuoso universal server seems to require special permission to support SERVICE keyword, while Wikidata endpoint timed out on relatively simple federated queries. The way around this difficulty was to use a local Fuseki server as an intermediate repository where partial results retrieved from one dataset are temporarily stored before a SERVICE call is placed to the other dataset.

In this paper, we focus on the application's capability to generate relevant and effective responses to a pre-defined set of user queries. Our primary interest was to test how well the application leverages its SPARQL query templates to access and utilize data from the selected repositories, thereby providing insightful and accurate information in response to user queries. We hope to use this as a measure of the application's utility and effectiveness in harnessing LOD resources in our chosen domain.

3. SPARQL Query Library

SPARQL query library of the MusicFans application currently houses a collection of nine distinct templates. These templates have been crafted to cater to a variety of user queries such as searches for hit song creators, popular songs by country, Grammy award recipients, American music composers, artists with significant total sales, and more. This diversity allows users to explore various aspects of music data, from commercial successes and geographical popularity to awards and artist-specific information. The current set of templates, while comprehensive, is designed to be flexible and adaptable. It can be expanded based on user feedback or the emergence of new and relevant SPARQL endpoints within the LOD cloud.

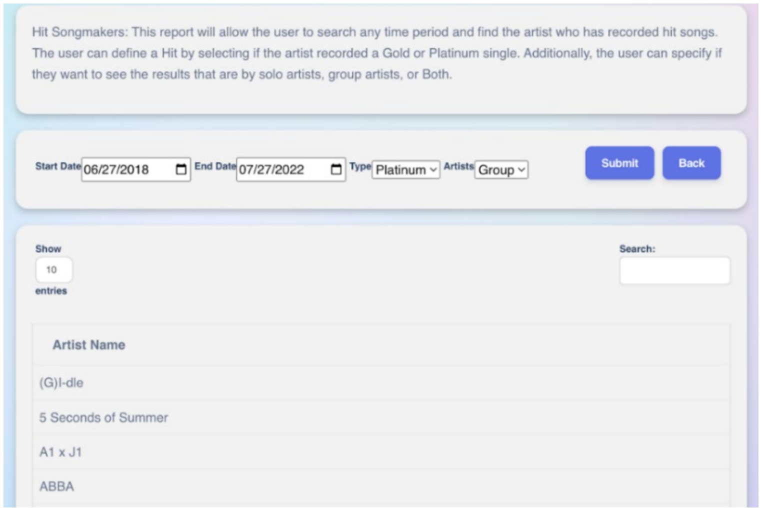
Next, we describe and illustrate the functionality of some of these templates. For those interested to explore the SPARQL query library in more detail, the implementation of MusicFans is available on GitHub at [https://github.com/caprav/Semantic-Web-Project] .

3.1 The *Hit Song Makers* template

This template (see Figure 2) is designed to generate a SPARQL query that enables users to search for hit songs recorded by artists over a chosen time frame. Users have the flexibility to categorize hits as either *Gold* or *Platinum* and can refine their search to include solo artists, group artists, or both. The search results are presented in an interactive table format, allowing users to easily navigate and analyse the data.
The embedded SPARQL query constructed according to the input parameters defined by the user is the following:

```
self.query_dbpedia_isHitSongOf = (
            "CONSTRUCT {?works <http://example.org/isHitSongOf> ?artist. "
            "?works dbo:releaseDate ?releaseDate. "
            "?works dbp:award " + self.hit_threshold + " . " +
            self.group_or_solo_query_string + "}" +
            "WHERE {?works a dbo:Song; "
            "dbp:award " + self.hit_threshold + "; " +
            "dbo:artist ?artist; "  # user defines "hits" as either gold or platinum
            "dbo:releaseDate ?releaseDate. "
            + self.group_or_solo_query_string
            + 'FILTER( ?releaseDate > "' + self.start_date + '"^^xsd:date '
              '&& ?releaseDate < "' + self.end_date + '"^^xsd:date)}' )
```

The data obtained from the CONSTRUCT query is also stored in the application's local Fuseki triplestore for two primary reasons. First, by keeping the triples locally, it opens the possibility of augmenting the data using reasoning tools like *Pellet* [10]. This enhancement can lead to more advanced and enriched data interpretations. Second, considering the potential time required to execute certain queries, caching these results locally enables quicker future retrieval. This feature is supported by the history module which is part of the front-end functionality of the application. This module not only records the user's search history but also utilizes the cached results from these searches, stored in the local triplestore, to provide quicker access in subsequent sessions. This approach significantly enhances user experience by reducing wait times for repeat queries and allowing for efficient revisitation of previous searches.


Hit Songmakers: This report will allow the user to search any time period and find the artist who has recorded hit songs. The user can define a Hit by selecting if the artist recorded a Gold or Platinum single. Additionally, the user can specify if they want to see the results that are by solo artists, group artists, or Both.

Start Date 06/27/2018 End Date 07/27/2022 Type Platinum Artists Group

Show 10 entries

Search:

Artist Name
(G)I-dle
5 Seconds of Summer
A1 x J1
ABBA

Figure 2. *Hit Song Makers* Template

3.2 The *Total Artist Sales* template

This template allows the user to search for an artist or a group and obtain their single sales. The template is shown on Figure 3.

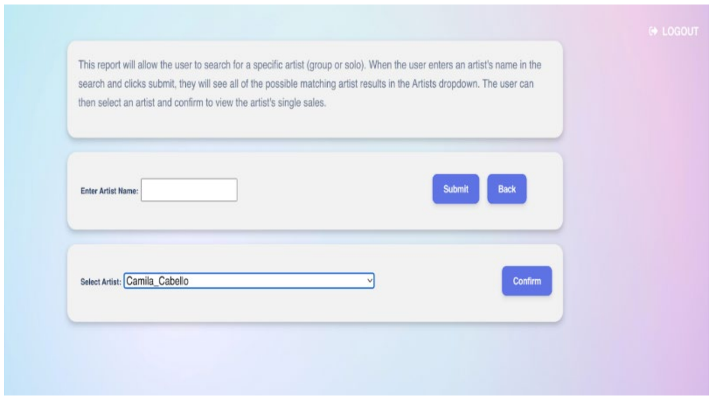
To implement this query, the search string is first run against all musical recording artists in DBpedia:

```
SELECT DISTINCT ?artist ?artist_name WHERE {{
        ?artist a dbo:MusicalArtist ;
             foaf:name ?artist_name .
        FILTER(CONTAINS(LCASE(str(?artist_name)), LCASE("{pattern}"))) }}
```

After the results are returned and the user selects the artist from the list of found artists the following query retrieves all sales from *singles* attributed to the artist.

```
SELECT DISTINCT ?work ?sales WHERE {{
        ?work dbo:artist dbr:{artist_name} ;
            a dbo:Song ;
            dbp:salesamount ?sales. }}
```

Lastly, before being presented to the user, the results are extracted from the returned information and summed up across all of the records (see Figure 4).

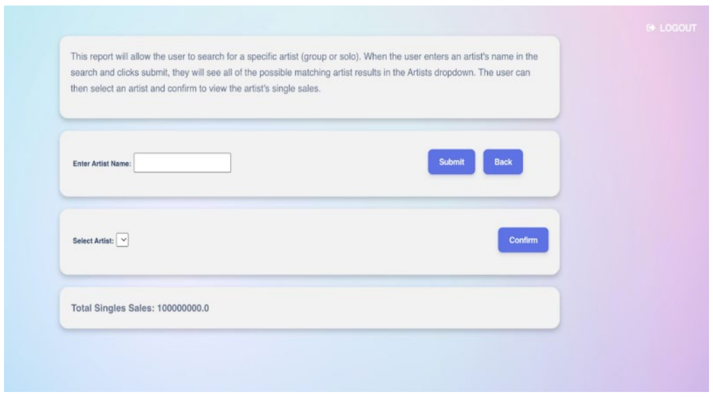


This report will allow the user to search for a specific artist (group or solo). When the user enters an artist's name in the search and clicks submit, they will see all of the possible matching artist results in the Artists dropdown. The user can then select an artist and confirm to view the artist's single sales.

Enter Artist Name:

Select Artist: Camila_Cabello

Figure 3. Artist Search



This report will allow the user to search for a specific artist (group or solo). When the user enters an artist's name in the search and clicks submit, they will see all of the possible matching artist results in the Artists dropdown. The user can then select an artist and confirm to view the artist's single sales.

Enter Artist Name:

Select Artist:

Total Singles Sales: 100000000.0

Figure 4. Total sales query result

3.3 The *Hits by Country* template

This template enables users to search for artists who recorded hit songs within any specified time frame. It then generates a comprehensive report detailing hit songs by country for that time frame. This query is an example of a federated query that allows for integration of data from multiple LOD cloud datasets.

As noted earlier, executing federated queries directly against Wikidata and DBpedia proved to be unviable. To address this challenge, the local Fuseki server is used as a SPARQL endpoint. This approach allows for the efficient gathering and integration of data from these distinct sources, facilitating the successful execution of federated queries within the application.

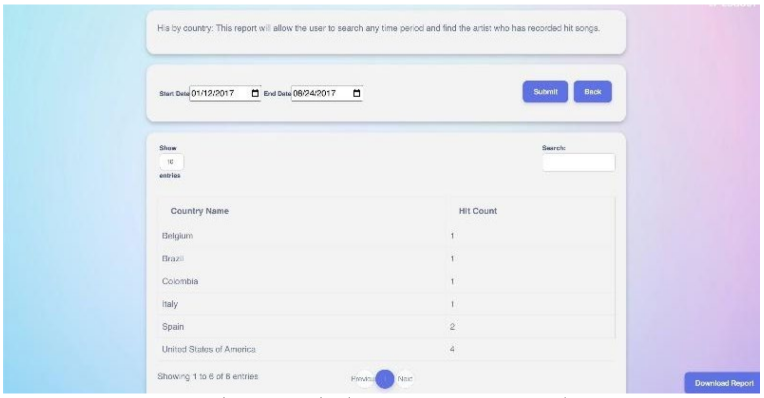
The execution of the query starts with posting the following CONSTRUCT query to DBPedia:

```
query_dbpedia_get_sameAs = (self.prefixes +
        "CONSTRUCT {"
        "?dbpedia_works owl:sameAs ?external_work_URI. "
        "?dbpedia_works dbo:releaseDate ?releaseDate. "
        "} "
        "WHERE { "
        "?dbpedia_works owl:sameAs ?external_work_URI; "
        'rdf:type dbo:Song; '
        'dbp:award "Platinum"@en; '  # user defines "hits" as either gold or platinum
        'dbo:artist ?artist; '
        'dbo:releaseDate ?releaseDate. '
        '?artist a owl:Thing. '
        'FILTER( ?releaseDate > "' + str(self.start_date) + '"^^xsd:date '
        '&& ?releaseDate < "' + str(self.end_date) + '"^^xsd:date '
        '&& (isURI(?external_work_URI) || isIRI(?external_work_URI)) '
        '&& STRSTARTS(STR(?external_work_URI), STR(wd:)))'
        "} " )
```

The extracted from DBPedia set of triples is stored in the local Fuseki triplestore and the subsequent federated query to Wikidata is sent:

```
results_federated_query = (self.prefixes +
        "SELECT DISTINCT ?external_work_URI ?label "
        "WHERE { "
        "?dbpedia_work owl:sameAs ?external_work_URI; "
        "dbo:releaseDate ?releaseDate. "
        "SERVICE <https://query.wikidata.org/sparql> { "
        "?external_work_URI p:P495 ?statement. "
        "?statement ps:P495 ?country. "
        "?country rdfs:label ?label "
        "FILTER(lang(?label) = 'en') "
        "} "
        'FILTER( ?releaseDate > "' + str(self.start_date) +
        '"^^xsd:date && ?releaseDate < "' + str(self.end_date) + '"^^xsd:date )'
        "} "   )
```

The combined result (see Figure 5) of these queries is a list of hit songs by the country which is returned to the user in a dictionary format which can be further searched by the user for a specific award/country.


His by country: This report will allow the user to search any time period and find the artist who has recorded hit songs.

Start Date 01/12/2017 End Date 08/24/2017

Show 10 entries

Search:

Country Name	Hit Count
Belgium	1
Brazil	1
Colombia	1
Italy	1
Spain	2
United States of America	4

Showing 1 to 6 of 6 entries

Figure 5. Hits by Country query result

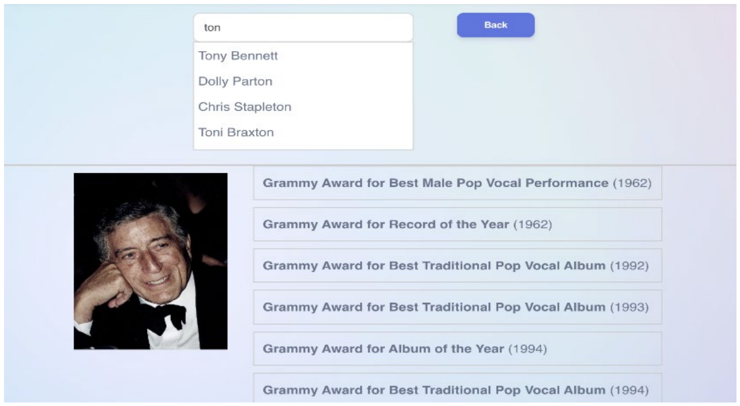
3.4 The *Grammy awards* and *Grammy winners by city* templates

These templates are designed to facilitate user searches for information regarding Grammy awards, applicable to both individual artists and bands. In addition, they offer functionality to compile and present a summary of Grammy awards categorized by city. These templates enhance user interaction by providing search suggestions and displaying content dynamically, thereby improving the overall user experience in accessing and understanding Grammy-related data. An example result of the Grammy awards search is shown on Figure 6.

Searching for *Grammy winners by city* template is shown next:

```
results_grammy_city_query_both = (self.prefixes +
        "SELECT ?hometown_name (COUNT(?artist) AS ?total) "
        "WHERE { "
        "{?artist dbo:wikiPageWikiLink dbc:Grammy_Award_winners;  "
        "rdf:type dbo:Person; "
        "dbo:birthPlace ?hometown_name .}"
        "UNION "
        "{?artist dbo:wikiPageWikiLink dbc:Grammy_Award_winners; "
        "rdf:type dbo:Group; "
        "dbo:hometown ?hometown_name .}"
        "}"
        "GROUP BY ?hometown_name " )
```

It should be noted that this query is one of the most time-consuming ones. It takes several minutes to obtain the result.



Grammy Award for Best Male Pop Vocal Performance (1962)
Grammy Award for Record of the Year (1962)
Grammy Award for Best Traditional Pop Vocal Album (1992)
Grammy Award for Best Traditional Pop Vocal Album (1993)
Grammy Award for Album of the Year (1994)
Grammy Award for Best Traditional Pop Vocal Album (1994)

Figure 6. Grammy awards query result

The SPARQL query library for MusicFans includes several additional templates that are not discussed in this paper for the sake of brevity. However, these templates are available for access and review. They can be found on the project's GitHub repository at [https://github.com/caprav/Semantic-Web-Project].

4. Challenges in Building Linked Data Applications

The MusicFans application was developed as part of a case study intended to assess the accessibility and practicality of utilizing the Linked Open Data (LOD) cloud for developing Semantic Web applications. A key objective of the project was to explore the design choices that bypass the need for creating local ontologies and data repositories. After an extensive evaluation of available music datasets on the LOD cloud, DBpedia and Wikidata were chosen as the primary sources of music-related data. While the LOD cloud hosts a variety of music-related datasets, most of them do not offer SPARQL endpoints which limits their utility for use by Semantic Web applications.
A fundamental aspect of the Semantic Web is its capability to interlink data from different sources via the OWL *sameAs* property [http://www.w3.org/2001/sw/WebOnt/]. This property is instrumental in linking a URI from one dataset to its equivalent in another, thereby establishing a connection between them. Once such a link is established, the two datasets are

interlinked which allows for implementation of federated queries. An example of a federated query is the one described in Section 3.3 and implemented by the *Hits By Country* query template.

The implementation of federated queries outlined several challenges that had to be addressed to operationalize this functionality successfully. These challenges included ensuring effective communication between different data sources and optimizing query performance. Even relatively simple federated queries like *Hits By Country* if directly posted on Wikidata SPARQL endpoint led to performance issues resulting in query timing out. The solution which we adopted was to utilize the local Fuseki server as a SPARQL endpoint where data returned from one resource is collected and then a federated query to the other resource is placed. This solution allowed for the efficient implementation of the federated query by the application.

SPARQL queries heavily rely on the specific data schema of the RDF dataset being queried. DBpedia and Wikidata have their own unique data schemas. A thorough understanding of these distinct schemas is essential for crafting accurate and effective SPARQL queries. This requirement represents a significant challenge in leveraging the LOD cloud for Semantic Web application development.

The complexity of SPARQL syntax, combined with the inherent ambiguity of natural language, poses additional difficulties. These challenges compounded with the necessity of having a comprehensive knowledge of the particular data schema result in substantial difficulties in employing semantic parsing techniques for automated SPARQL query generation. This makes the prospect of using advanced language models like ChatGPT for simplifying user interactions in Semantic Web applications a distant goal.

Given these complexities, a more pragmatic and immediate solution for applications like MusicFans is the use of a well-thought-out library of SPARQL query templates. Such a library can streamline the query-building process and provides a practical pathway for building user interfaces in Semantic Web applications, mitigating the challenges posed by the technical intricacies of SPARQL and the nuances of RDF data schemas.

5. Conclusion

Building linked data applications unlocks substantial potential for harnessing the power of the Semantic Web. Yet, this requires overcoming a range of technical, data-centric, and user experience challenges. This paper discussed MusicFans, a real-world application developed as part of a case study intended to assess the feasibility and efficiency of utilizing the LOD cloud within a specific domain deliberately circumventing the need to create local ontologies and data repositories.

A well-recognized challenge in utilizing Semantic Web technologies is the requirement for users to have familiarity with the SPARQL query language. Addressing this, the paper detailed a pragmatic solution: the utilization of a pre-defined library of dynamically generated SPARQL query templates. We advocated that, as of now, this approach represents the most viable method to facilitate user interaction within the Semantic Web domain. The paper delves into how these templates are designed to simplify user engagement with the application, making it more accessible to those without knowledge of SPARQL, and thereby broadening the application's usability in real-world scenarios.

References

[1] Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. *Scientific American*. pp. 96-101, May 2001

[2] Shadbolt N., T. Berners-Lee and W. Hall, "The Semantic Web Revisited," in *IEEE Intelligent Systems*, vol. 21, no. 3, pp. 96-101, Jan.-Feb. 2006, doi: 10.1109/MIS.2006.62.

[3] Dimitrakis E., K. Sgontzos, M. Mountantonakis, and Y. Tzitzikas - Enabling Efficient Question Answering over Hundreds of Linked Datasets. *Post-proceedings of the 13th International Workshop on Information Search, Integration, and Personalization (ISIP'2019)*, 2019.

[4] Abujabal A., M. Yahya, M. Riedewald, and G. Weikum. - Automated template generation for question answering over knowledge graphs. *Proceedings of the 26th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 2017.

[5] Diefenbach D., K. Singh, and P. Maret. -WDAqua-core1: A Question Answering service for RDF Knowledge Bases. *WWW'18: Companion Proceedings of the The Web Conference*, 2018.
[6] Shaik S., P. Kanakam, S. Hussain,and D. Suryanarayana - Transforming Natural Language Query to SPARQL for Semantic Information Retrieval, *International Journal of Engineering Trends and Technology (IJETT)*, v. 41, no. 7, 2016.
[7] [Online] Available Semantic Parsing Natural Language into SPARQL: Improving Target Language Representation with Neural Attention 1803.04329.pdf (arxiv.org)
[8] Zlatareva N., D. Amin - Natural Language to SPARQL Query Builder for Semantic Web Application, *Journal of Machine Intelligence and Data Science (JMIDS)*, vol. 2, 2021, Avestia Publ.
[9] Lehmann J., Isele R., Jackob M., Jentzsch A., Kontokostas D., Mendes P., Hellmann S., Morsey M., van Kleef P., Auer S., and C. Bizer C – DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *SemanticWeb* 6, vol. 2, 2015.
[10] [Online] Sirin E., Parsia B., Cuenca Grau B., Kalyanpur A., Katz Y. Pellet: A Practical OWL-DL Reasoner. http://www.cs.ox.ac.uk/people/bernardo.cuencagrau/publications/PelletDemo.pdf