# Orchestrating Music Queries via the Semantic Web

Milos Vukicevic, John Galletly
American University in Bulgaria
Blagoevgrad 2700
Bulgaria
+359 73 888 466
milossmi@gmail.com, jgalletly@aubg.bg

*Abstract* - **This paper describes the design and implementation of a Semantic Web application that allows queries and inferences to be made on a music knowledge base using Semantic Web technologies such as RDF, OWL and SPARQL. Additionally, the paper explains how these technologies were blended together to develop the application that illustrates the principles of the Semantic Web.**

## I. INTRODUCTION

The Semantic Web has been heralded by the W3C as the future web – a web that relies heavily on the software implementation of knowledge bases and inference mechanisms [1]. The Semantic Web has several standards recommended by the W3C with, currently, varying levels of functionality and usability [2]. It is still heavily under development and evolution, with the latest standard coming out in 2014. The Semantic Web software stack [3] is illustrated in Figure 1

The application described in this paper is a Semantic Web application that allows music queries and inferences to be made on a music knowledge base. The word "knowledge" is important – a traditional database approach would not give the breadth and scope for queries and inferences that a knowledge base (expressed as an RDF ontology) would.

The design and implementation of a fully-fledged music ontology was beyond the scope of this work. Rather than rely on a large, ready-built music ontology (e.g. mucicontology.com), the application described here was developed with a much narrower ontology, namely one for rock music and bands. But, given enough time and effort, the design described here could be extended to cover different types of music and artists. Information about artists, tracks, etc. in this ontology, is represented as RDF statements.

The use of the Semantic Web technologies in the music industry is not new. For example, the BBC's Music Project is an effort by the BBC to build semantically-linked and annotated web pages about artists and singers whose songs are played on BBC radio stations [4].

## II. DEVELOPMENT ENVIRONMENT

Apache Jena [5], in conjunction with the Eclipse IDE, was used as the basic programming environment. Jena is a Java-based API for Semantic Web development. It provides extensive Java libraries for handling RDF, OWL and SPARQL in line with the published W3C recommendations. Jena includes a rule-based inference engine to perform reasoning based on OWL and RDFS ontologies, and a variety of storage strategies to store RDF triples.

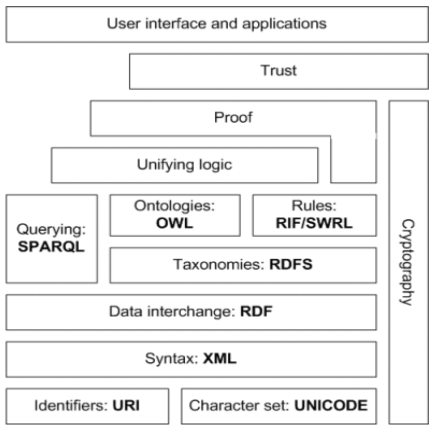


Figure 1 – Semantic Web software stack

The Stanford Protégé ontology editor [6] was used to build the application's ontology, as this editor provides an easy-to-use environment for developing ontologies. The ontology was developed using OWL Description Logic (OWL DL). Once built, the ontology was then loaded into Jena. Jena's generic inference mechanism was used to make inferences between the ontology classes. Additionally, the Jena SPARQL query engine allows for expressive SPARQL queries. However, it does not contain the full implementation of SPARQL as it is envisioned by the W3C. It is impossible for the user to create resources or add properties, only to search for already existing graph patterns.

## III. DESIGN

### *A. Design Overview*

From the outset, the design of the software was made to be scalable, and is essentially developed with an MVC pattern, where the GUI is the View, the ontology is the Model, and the Jena inference engine and SPARQL query engine are the Controller [7, 8].

Basically, the ontology is used as the basis for executing SPARQL queries, and making inferences using the Jena inference mechanism. The ontology had to be extensible in terms of having the ability of adding new ontologies to it and expanding the ontology itself, while also providing a scalable ground for adding new instances of ontology classes, etc.

With the limits imposed by the current standards, and by the architecture of Jena, the SPARQL queries had to be created programmatically to fit the ontology. The queries had to be designed in such a way that they would operate with the architecture of the ontology in question, making full use of the data and logic provided by it. Moreover, the application's SPARQL interface had to be designed in such a way that it would handle additions to the ontology, and ensure that the program would still work correctly, even with these additions.

The GUI is the front end, and is able to accept four types of queries: queries for a track, album, artist or band. The results are shown in three screen text panels, one containing basic data inferences, the other containing basic relationship inferences (such as Artist X plays in Band Y), and advanced inferences linking independent nodes together semantically (Figure 2).

As the application was designed with scalability in mind, adding more query types to the list would not be too difficult, but the ontology, as is designed currently, requires no further subtypes.

The ontology consists of several top-level classes. These classes all have instances of themselves, in some case multiple instances. The semantic web allows for a dynamic addition of other instances of these classes, even of other classes. The ontology class design can be seen in the Figure 2.

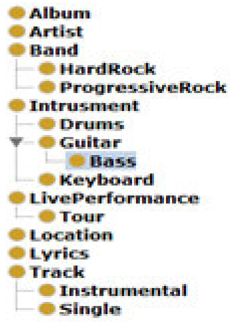


Figure 2 – Ontology top-level classes

The relationships between classes are defined using object properties. Figure 3 is a list of all object properties in the ontology.

Object properties act as predicates between individuals but no literals. Predicates for literals are data object properties and they are illustrated in Figure 4.

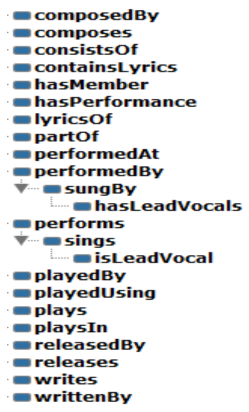


Figure 3 – Ontology object properties

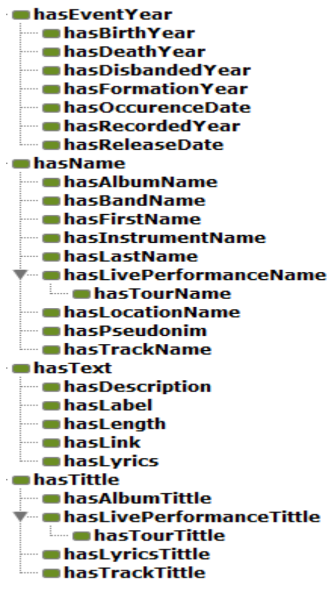


Figure 4 – Ontology data object properties

Figure 5 shows the GUI, the SPARQL engine and ontology packages with their dependencies. The GUI relies on the SPARQL engine to populate it with data. The GUI package has various elements and functions that allow it to display the data properly and also capture button click events. The SPARQL Engine has all the necessary data structures and functions to run queries, process them, and perform advanced inference.

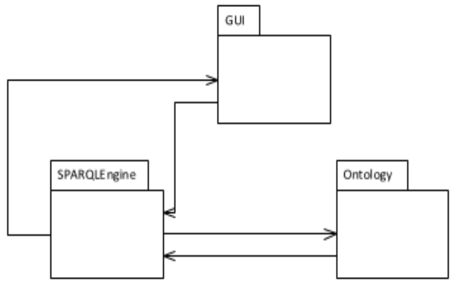


Figure 5 – UML package diagram

Figure 6 illustrates the deployment of the packages.

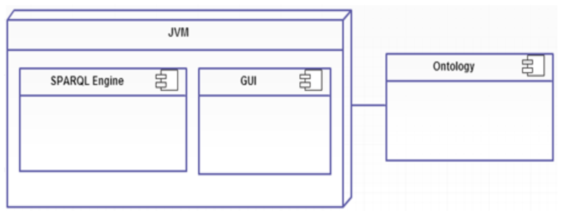


Figure 6 – UML deployment diagram

## *B. Design Details*

The ontology is stored as a separate file using the RDF/XML standard, and it is then imported into the Java application using the Jena ModelFactory pattern. The ontology contains the ontological specifications, i.e. all the classes, object properties and data properties available for the specific ontology, along with all the individual instances of the classes and their predicates. It is designed so it is extensible, i.e. new classes can be added, as well as new ontologies, and it is also scalable, i.e. new individuals can be added without negatively impacting the execution of the entire software solution.

The Jena inference and SPARQL query engines operate on the ontology. After being loaded into the application, the inference engine is run on the ontology and an "inferred" model is created. This is an extended, in-memory version of the ontology, providing advanced inferences about the classes and properties. This inferred model is then used as a basis for various SPARQL queries.

The Semantic Web can essentially allow for an extremely large amount of semantic queries (such as "Who played the guitar at concert X?") and therefore needs some kind of query parsing or translation mechanism to allow the application to "understand" what exactly it is that the user is looking for. This is programmatically a challenge in its own right, and there was not enough time to implement such an input parser. However, having the ontology in mind, the interface to the SPARQL engine was constructed in such a way that it is able to return complex inferences from the ontology itself for a particular set of search strings.

While the user is able to perform basic semantic querying, the SPARQL interface takes the particular query of the user and retrieves additional advanced semantic inferences about the particular object the user is looking for. This was accomplished by generating inferred assertions using Protégé's inference engine operating on the ontology, as the ontology was built.

### *C. The SPARQL Interface*

This is the "heart" of the application. This part revolves around reading the user input, and then trying to match it to a list of all albums, artists, bands, or tracks, depending on what the user has selected. If there is a match, this is then processed and a query is created that can be run against the ontology. This module is also responsible for loading the ontology, creating an inferred model using the Jena inference mechanism and then running queries on the inferred (in-memory) model.

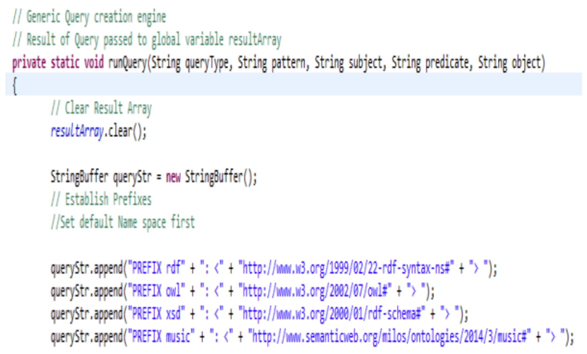
There are several operations that need to be performed before doing so. The most straightforward function is the URI dereferencing. All entities in the ontology have a URI namespace prefix. For example, the Pink_Floyd instance of the class Band always has the entire namespace prefixed to it, so it would be:

http://www.semanticweb.org/milos/ontologies/2014/3/music #Pink_Floyd

Before an entity can be searched for, the namespace must be removed.

Similarly, there is an algorithm that prepares an entity for output based on its type, i.e. Artist, Band, Album, Track, etc. This turns Pink_Floyd into "Pink Floyd," for example.

The SPARQL query is built functionally. The example below demonstrates the SPARQL query interface. It takes in the query type, which would be SELECT in most cases, the string pattern which is the subject of selection, the subject of the WHERE clause, the predicate of the WHERE clause, and the object of the WHERE clause. This returns a distinct result set which is passed onto a globally declared variable called resultArray. The resultArray is an ArrayList of type string that stores all the information a SELECT query returns. The main application then deals with the returned data in some way.

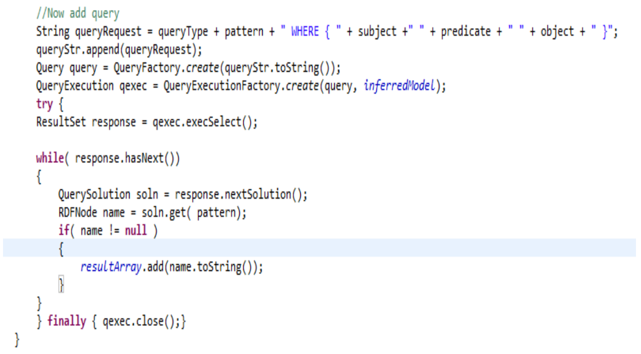


```
// Generic Query creation engine
// Result of Query passed to global variable resultArray
private static void runQuery(String queryType, String pattern, String subject, String predicate, String object)
{
        // Clear Result Array
        resultArray.clear();

        StringBuffer queryStr = new StringBuffer();
        // Establish Prefixes
        //Set default Name space first

        queryStr.append("PREFIX rdf" + ": <" + "http://www.w3.org/1999/02/22-rdf-syntax-ns#" + "> ");
        queryStr.append("PREFIX owl" + ": <" + "http://www.w3.org/2002/07/owl#" + "> ");
        queryStr.append("PREFIX xsd" + ": <" + "http://www.w3.org/2000/01/rdf-schema#" + "> ");
        queryStr.append("PREFIX music" + ": <" + "http://www.semanticweb.org/milos/ontologies/2014/3/music#" + "> ");
```

The code below illustrates the second part of query execution, where the query is formulated and executed using the functions provided by the Jena ModelFactory. Both resources and literals are retrieved in this fashion with proper formulation of queries. However, the advanced inference relies on a programmatically use of several query calls, relating individuals that are not usually directly related - more on this in the implementation section.

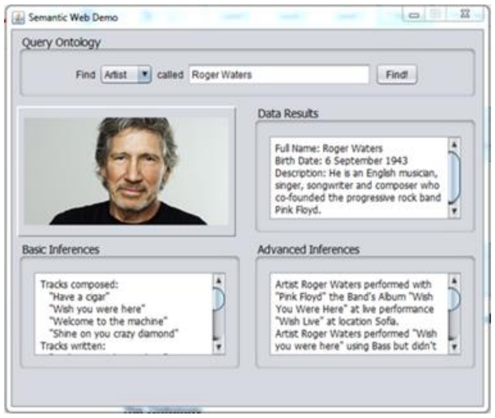


```
    //Now add query
    String queryRequest = queryType + pattern + " WHERE { " + subject +" " + predicate + " " + object + " }";
    queryStr.append(queryRequest);
    Query query = QueryFactory.create(queryStr.toString());
    QueryExecution qexec = QueryExecutionFactory.create(query, inferredModel);
    try {
    ResultSet response = qexec.execSelect();

    while( response.hasNext())
    {
        QuerySolution soln = response.nextSolution();
        RDFNode name = soln.get( pattern);
        if( name != null )
        {
            resultArray.add(name.toString());
        }
    }
    } finally { qexec.close();}
}
```

### D. The GUI

The GUI accepts and parses user input data, and displays the results of the query and the relevant basic and advanced inferences on the screen. The front end was simplified to provide scope-limited queries, in the sense that the user could query for specific information while the inferences were prebuilt into the ontology itself. That is to say, the user could query to find an artist, and the artist would be found, while advanced inferences about the artist are displayed in the information boxes. The screen itself is split into five panels (Figure 7). The top panel is the search box and it does not change. It contains a combo box allowing the user to select the type of query he/she wants to perform (Find artist, album, track or band), a textbox for the actual query string, and a button to initiate the query. The other four panels are used to display the data retrieved. The first and top left panel of the four displays the relevant image associated with the query. The second panel contains basic data inferences, such as data properties. The third panel contains subject assertions, i.e. the correlation of the searched subject with all the other subjects that the searched subject is immediately connected to in terms of the semantic graph. The fourth panel contains the advanced queries, linking multiple nodes that are not directly correlated, or performing operations on existing data.



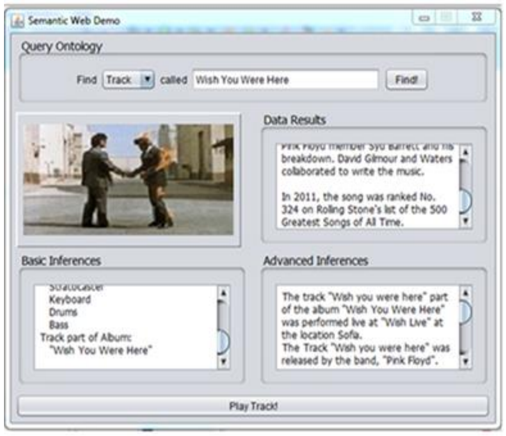
Full Name: Roger Waters
Birth Date: 6 September 1943
Description: He is an English musician, singer, songwriter and composer who co-founded the progressive rock band Pink Floyd.

Tracks composed:
"Have a cigar"
"Wish you were here"
"Welcome to the machine"
"Shine on you crazy diamond"
Tracks written:

Artist Roger Waters performed with "Pink Floyd" the Band's Album "Wish You Were Here" at live performance "Wish Live" at location Sofia.
Artist Roger Waters performed "Wish you were here" using Bass but didn't

Figure 7 – Finding an artist

Figure 7 and Figure 8 illustrate the functionality for finding a particular named artist and a particular named track.



Pink Floyd member Syd Barrett and his breakdown. David Gilmour and Waters collaborated to write the music.

In 2011, the song was ranked No. 324 on Rolling Stone's list of the 500 Greatest Songs of All Time.

Stratocaster
Keyboard
Drums
Bass
Track part of Album:
"Wish You Were Here"

The track "Wish you were here" part of the album "Wish You Were Here" was performed live at "Wish Live" at the location Sofia.
The Track "Wish you were here" was released by the band, "Pink Floyd".

Figure 8 – Finding a track

The last element of the GUI is the "Play Track" button which is hidden at the bottom of the page and is only displayed once a track is searched for. If clicked, it will open a new frame which opens a relevant YouTube link to the track in question (Figure 9).

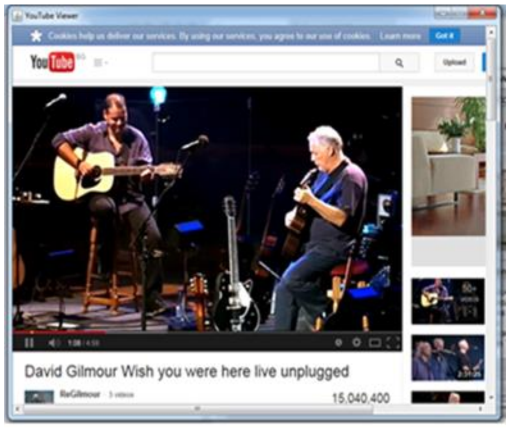


Figure 9 – YouTube link frame

## E. Semantic Web Implementation

RDF is the Semantic Web notation for modelling application domain information. The information is actually represented in the form of a graph database. "Pieces" of information are represented in the form of assertions called statements and each statement is made out of three parts (or triples): a subject, a predicate and an object.

Each subject and object represents either a resource or literal, while the predicate illustrates the relationship between subjects, objects, and literals.

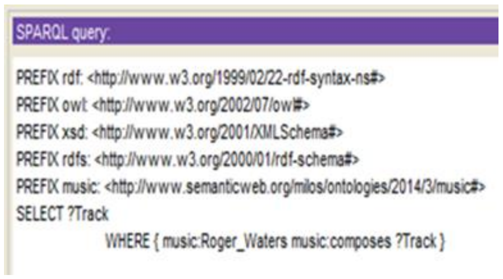
While RDF allows the description of domain resources and relationships using domain vocabularies, it does not support semantics. RDF Schema (RDFS) is an extension to RDF that allows the description of semantics in terms of classes, instances of classes, hierarchies, etc. RDFS allows the creation of disjoint properties, the specifications of types, domains and ranges, as well as indicating the type of properties in terms of their being functional, reflexive, and transitive, etc. This permits a simple implementation of semantics that is used as the basis for the powerful Web Ontology Language, OWL.

W3C developed OWL as a standardized way of expressing higher-level data semantics in Semantic Web applications. Like RDF and RDFS, OWL has an XML-based syntax. It comprises several sections. The first section is a header section where appropriate OWL namespaces are referenced. This section is followed by class declarations, object properties and data properties. After these declarations comes the individual specifications section, which declares instances of the classes and uses the aforementioned object properties to link different individuals together, while the data properties are used to link individuals with literals.

The following is an example of an OWL expression used to declare an individual of class Artist.

SPARQL is in many ways similar to SQL but it is for the Semantic Web. For example, the SELECT command specifies the result set and its name, while FROM clause indicates which file or SPARQL endpoint will be queried for the result. A WHERE statement specifies the graph pattern to be searched for, while ORDER can be used for data result formatting.

```
<NamedIndividual rdf:about="&music;Nick_Mason">
    <rdf:type rdf:resource="&music;Artist"/>
    <music:hasBirthYear>27 January 1944</music:hasBirthYear>
    <music:hasLastName>Mason</music:hasLastName>
    <music:hasDescription>He is an English musician and composer, best known as the
drummer of Pink Floyd. </music:hasDescription>
    <music:hasFirstName>Nick</music:hasFirstName>
    <music:plays rdf:resource="&music;Drums"/>
    <music:playsIn rdf:resource="&music;Pink_Floyd"/>
    <music:performs rdf:resource="&music;have_a_cigar"/>
    <music:performs rdf:resource="&music;shine_on_you_crazy_diamond"/>
    <music:performs rdf:resource="&music;welcome_to_the_machine"/>
    <music:performs rdf:resource="&music;wish_you_were_here"/>
</NamedIndividual>
```



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX music: <http://www.semanticweb.org/milos/ontologies/2014/3/music#>
SELECT ?Track
        WHERE { music:Roger_Waters music:composes ?Track }
```

The above diagram illustrates a simple SPARQL query. The PREFIX statements define various namespaces, with "music" being the namespace for the ontology. The query SELECTs a subject (?Track) that fits into the graph pattern subject-predicate-object (music:Roger_Waters - music:composes - ?Track). This query will return a list of all Tracks composed by artist Roger_Waters. The program iterates through all top-level entity object properties and appends them to the basic inferences text area.

The advanced inferences rely on multiple levels of connection and making further inferences. For example, the inference "Roger Waters has played with Pink Floyd, the album "Wish You Were Here" at the concert "Wish Live" in Sofia", is deduced in this way. A number of special algorithms were developed to make these inferences for artists, tracks, bands, etc. For example, the algorithm for the above inference is

1. An artist is selected.
2. A list of band(s) is found through basic inference.
3. A list of all matched bands is looped through, and a list of all albums is found through basic inference through the Band, connecting the Artist with the album (no direct connection).
4. A list of all matched albums is looped through, and a list of all live performances is found through basic inference through the Album, connecting the Artist with the live performance and the band with the live performance. (no direct connection)
5. A list of all matched live performances is looped through, and a list of all locations where the performances were held is found, connecting the album, artist, and band with the locations.
6. A list of all locations where the live performance was held is looped through, connecting the Album with the location, the Band with the location, and the Artist with the location, and the results are printed recursively at this point from the location back to the Artist. (no direct connection)

## IV. CONCLUSION

The design and implementation of a Semantic Web application, that handles music queries for a limited domain, has been described. In principle, the application could be further extended as a comprehensive, semantically-organized music knowledge base with support for all types and genres of music, ranging from modern rock and roll and pop, to classical music and classical pieces of music.

## REFERENCES

[1] W3C Semantic Web: http://www.w3.org/standards/semanticweb/

[2] W3C Semantic Web Standards: http://www.w3.org/standards/semanticweb/

[3] Wikipedia: Semantic Web: http://en.wikipedia.org/wiki/Semantic_Web

[4] BBC Music Project

http://readwrite.com/2009/01/21/bbcs_semantic_music_project

[5] Apache Jena Documentation: https://jena.apache.org/documentation/

[6] Protégé Wiki: http://protegewiki.stanford.edu/wiki/Main_Page

[7] Strategies for Building Semantic Web Applications: http://notes.3kbo.com/sparql

[8] Semantic Web Programming: https://code.google.com/p/ia1213/downloads/detail?name=semantic-web-programming.9780470418017.47881.pdf