Report
Knowledge Engineering
Linked Pop Music

Jorge Arranz Simón, Irene Herrero Carranza, María Sánchez Villanueva

June 09, 2022

1. Introduction

This project aims at creating an ontology and a linked data knowledge graph capturing information about **pop music compositions,** from difference data sources as Genius, Musicbrainz, Second hand songs, Songfacts and Spotify. This data sources may use different formats and refer to common items, but without having explicit links between other. We aim to perform an **entity linking** task to make sure that the resulting knowledge graph will link almost all the musical entities they have in common.

We followed the **eXtreme Design Methodology**, hence after having analyzed the different data files, we established several competency questions (CQs) that we will then use to model our ontology. Afterwards, in the testing phase, we checked the consistency of our ontology and the correctness of the inferences by verifying the CQs. Finally, according to the developed ontology, we mapped the available data into RDF triples using SPARQL-Anything.

2. Data

Our data is related to the Isophonic Datasets used for the Poliphonia Project. It contains data from the original dataset (Reference Annotations: The Beatles; Reference Annotations: Queen) as well as metadata for each song downloaded from different services (Genius, Musicbrainz, Songfacts, Spotify and Secondhandsongs).

The access to the data is public via a repository of Github (https://github.com/polifonia-project/datasets/tree/main/dataset/isophonics).

The repository is structured as follows:

2.1. Chord annotations

Contains the chord annotations (and their relative durations) of the original datasets.

2.2. Metadata

Contains the metadata files of the original datasets.

2.3. Cornucopia data

Contains additional metadata files downloaded using the cornucopia tool. This tool allows to retrieve and store metadata about each track of a music dataset from different web services, using web APIs. The format of the data stored in this folder follows the structure of the service from which the data was obtained.

This folder involves five more folders:

2.3.1. Genius: contains two files per each song of The Beatles and Queen: one with its lyrics and another with annotations of the respective song.

2.3.2. Musicbrainz: for Queen and The Beatles, four different folders are available: for the artists, recordings, releases and works. Each of these folders has one file per each element.

2.3.3. Spotify: is composed of a metafile for each song of the groups.

2.3.4. Secondhandsongs: shows information about each group (its members, country…).

2.3.5. Songfacts: consists of a file with facts (curiosities, annotations from the writer) of each song.

2.4. data_map.csv

Contains a mapping between each track of the original dataset and the metadata stored in the folders contained in Cornucopia Data. Any references to the data from the services Spotify and Songfacts appear in this file because they haven't been aligned to the dataset.

We finally have taken into account mostly the data placed in the folder “Musicbrainz”, because it contains multiple files in which mostly all the information is included. Finally, we have decided to do the **linking** between the four different sources, as it was the most complete dataset but in separate files. The linking of the Artist – Work – Recording – Release folders is modelled in the graph as shown in the graph

3. Competency questions

Competency questions (CQs) are the natural language counterpart of structured queries that we want to enable against the knowledge graph. These interrogatives allow us to scope our ontology: they are questions that our users would want to gain answers for, through exploring and querying the ontology and its associated knowledge base.

In this report section we specify our CQs and its respective SPARQL query implementation.

1. Who publishes a certain song?

```
SELECT ?namePublisher
WHERE{
	?Song pm:isPublishedBy ?Publisher .
	?Publisher pm:hasName ?namePublisher
}
```

2. What is the begin date of a Band?

```
SELECT DISTINCT ?date
WHERE{
    ?Band pm:hasBeginDate ?date .
}
```

3. What is the original language of a certain song?

```
SELECT ?language
WHERE{
	?Song pm:hasLanguage ?language
}
```

4. When was a certain Record released?

```
SELECT ?date
WHERE{
	?Recording pm:hasRelease ?Release .
	?Release pm:hasDate ?date
}
```

5. What is the title of a version of a certain song?

```
SELECT DISTINCT ?title
WHERE{
    ?song pm:hasVersion ?version .
    ?version pm:hasTitle ?title
}
```

6. What is the language of a version of a certain song?

```
SELECT DISTINCT ?language
WHERE{
    ?song pm:hasVersion ?version .
    ?version pm:hasLanguage ?language
}
```

7. Which is the length of a Recording?

```
SELECT DISTINCT ?length
WHERE{
    ?recording pm:hasLength ?length .
}
```

8. What is the rate of a Recording?

```
SELECT DISTINCT ?rate
WHERE{
    ?recording pm:hasRate ?rate .
}
```

9. Who is the Artist that plays a certain song?

```
SELECT DISTINCT ?name
WHERE{
      ?song pm:isPlayedBy ?musician .
      ?musician pm:hasName ?name
}
```

10. When did a Artist start his career?

```
SELECT DISTINCT ?date
WHERE{
    ?Musician pm:hasBeginDate ?date .
}
```

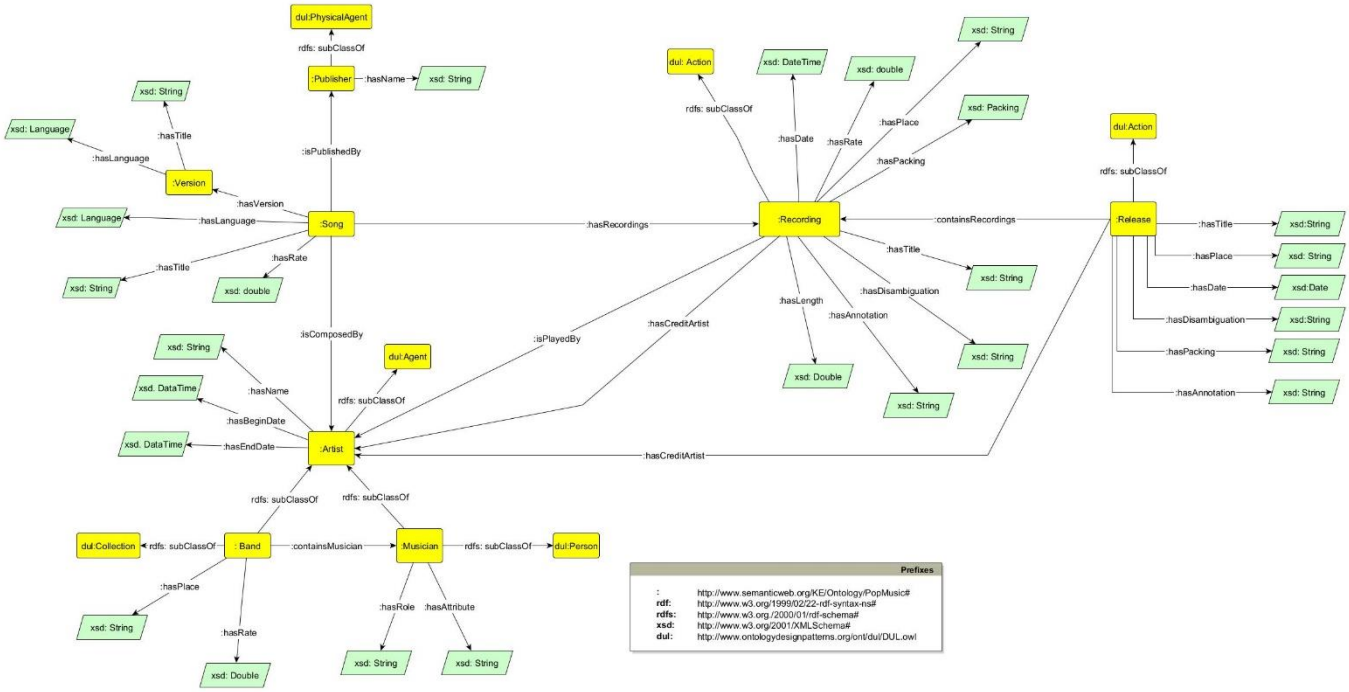
11. Where was a Band born?

```
SELECT ?place
WHERE{
```

```
    ?Band pm:hasPlace ?place .
}
```

4. Ontology

In this section we present our ontology and explain our design choices: the classes, patterns, and relations we have implemented.



The graph represents all the classes and relations of our ontology. Everything is centered around the **:Song,** as we have considered that is the most important aspect in the Music. The class **:Song** makes reference to literal definition of a song: “a short poem or other set of words set to music or meant to be sung”. It has as main characteristics its *language* and *original title.* It could have different **:Version**, which are the possible translations of the poem, and it has different kind of agents involved in its creation: Its composer, the **:Artist** who wrote the letter, and the **:Publisher**, the person/entity in charge of the publication.

Respect to the **:Artist**, we have reuse the definition of **dul:Agent** (Any agentive Object , either physical, or social). We have considered two types of :Artist, on one hand could be a solist **:Musician**, considered a **dul:Person**, and the other hand could be a :Band, a set of musicians modelled under the definition of **dul:colletion** (any container for entities that share one or more common properties). We have considered as interesting features

for a :**Musician** its *role* while referring to the :**Song** and other optional *attributes*. While referring to the **:Band**, we have considered as interesting its *rate* and *place*.

In the musical aspects, we have considered as a **:Recording** all the different times a :Song has been sung. This **:Recording** is associated with **dul:Action** as it is an Event with one :**Agent** involved. Some of the characteristics that we have considered interesting are shown in the graph, such as *spatiotemporal* aspects and *musical* facts.

The **:Release** identifies each release of a product on a specific date with specific release information such as the place, date, and packaging. It can refer to an Album: a set of :**Song** or to a Single, a unique song released alone.

4.1. Patterns

Band	*Collection*	A band is a collection of Musicians
Publisher	*PhysicalAgent*	Physical person or company that can plan an action (the publishment of songs).
Artist	*Agent*	The figure of the person related to the music world. This makes a difference between the Artist and the real-life person is behind.
Release/Recording	*Action*	An Event with at least one Agent (in our case Artist) that is participant in it.
Musician	*Person*	Persons in commonsense intuition, which does not apparently distinguish between either natural or social persons.

5. Testing

We have determined that our testing procedure will be **Competency Question Verification**. This testing procedure allows to verify if the ontology is able to answer to the competency questions that have been collected.

For carrying out this approach we first created a few test cases as defined by the testing framework OWL unit. We created Toy datasets as input testing data. OWLunit makes sure that the IRIs used within the SPARQL query are defined either in the tested ontology or in the input test data, the result of the SPARQL unit test query evaluated over the input data is isomorphic to the expected result.

We specified a test case as follows:

```
@prefix owlunit: <https://w3id.org/OWLunit/ontology/> .
@prefix tc: <https://raw.githubusercontent.com/jorge-arranz/Pop-Music-KE22/main/test/competency-question/test-case/> .
@prefix td: <https://raw.githubusercontent.com/jorge-arranz/Pop-Music-KE22/main/test/competency-question/toy-dataset/> .
@prefix pm: <https://raw.githubusercontent.com/jorge-arranz/Pop-Music-KE22/main/Ontology/PopMusic_New.owl> .


tc:CQ6.ttl a owlunit:CompetencyQuestionVerification ;
    owlunit:hasCompetencyQuestion "What is the language of a version of a certain song?" ;
    owlunit:hasSPARQLUnitTest "PREFIX pm:<https://raw.githubusercontent.com/jorge-arranz/Pop-Music-KE22/main/Ontology/PopMusic_New.owl> SELECT DISTINCT ?language WHERE{ ?song pm:hasVersion ?version . ?version pm:hasLanguage ?language} " ;
    owlunit:hasInputData td:TD6.ttl ;
    owlunit:hasInputTestDataCategory owlunit:ToyDataset ;
    owlunit:hasExpectedResult "{  \"head\": {  \"vars\": [  \"language\" ] } ,  \"results\": {  \"bindings\": [ {  \"language\": {  \"datatype\":  \"http://www.w3.org/2001/XMLSchema#language\" , \"type\":  \"literal\", \"value\": \"en\" } } ] } }";
    owlunit:testsOntology pm: .
```

6. Data mapping

SPARQL-Anything is a system for Semantic Web re-engineering that allows users to query anything with SPARQL. For this reason, we used SPARQL-Anything to assemble RDF triples that will be stored in a triple store by querying our pre-processed data with SPARQL. As we used only the data provide by MusicBrainz, all the file we proceed were of the type of *JSON*. The data was already divided into 4 different categories: *Work, Release, Recording and Artist*. In pursuance of having a clear and clean environmental that might be useful for further creation of triples from other files with the same structure, we designed one mapping file for each type. (*Work.sparql*, *Release.sparql*, *Recording.sparl* and *Artistis.sparql*). By using the PowerShell and taking advantage of the different output file formats supported by SPARQL Anything, we obtained one Turtle file for each input JSON file. These files contain the data aligned with our ontology i.e., in such a way the triples can be stored and queried in our ontology.

7. SPARQL Endpoint and Triple Store

Once we obtained the Turtle file with the RDF triples, we want to store in our Ontology we use the tool Blazergraph. Blazegraph DB is an ultra-high-performance graph database supporting blueprints and RDF/SPARQL. It covers all application needs starting from small applications with embedded storage, to larger standalone ggapplications, and up to 50B statements stored in NanoSparqlServer.

For our project, we chose to work locally and not to publish public URIs, so we installed a executable jar artifacts work with Java 9+. Once starting it from the PowerShell, the default workbench location is http://localhost:9999/blazegraph/ . Blaze is an instance segregated by namespace, so we first created our name space. Then from the "UPDATE" tab, we loaded our data by indicating they are RDF triples in Turtle format. Finally, from the "QUERY" tab, we were able to query our data.

Attached, some examples of queries:

1. Which are the titles of the songs composed by "*John Lennon*"?
    - Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org./2000/01/rdf-schema#>
PREFIX pm: <https://raw.githubusercontent.com/jorge-arranz/Pop-Music-KE22/main/Ontology/popMusic.owl>

SELECT ?SongName
WHERE{
  ?Song a pm:Song;
          pm:hasTitle ?SongName;
          pm:isComposedBy ?Composer .
  ?Composer pm:hasName "John Lennon" .


}
```

    - Output

SongName
There's a Place
P.S. I Love You
Please Please Me
Misery
Love Me Do
I Saw Her Standing There
Do You Want to Know a Secret
Ask Me Why

2. Who are the members of the band "*The Queen*" and which work do they develop in the band?
    - Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org./2000/01/rdf-schema#>
PREFIX pm: <https://raw.githubusercontent.com/jorge-arranz/Pop-Music-KE22/main/Ontology/popMusic.owl>

SELECT ?musicianName ?musicianAttribute
WHERE{
  ?Band a pm:Band ;
          pm:hasName "Queen";
          pm:containsMusician ?Musician.
  ?Musician pm:hasName ?musicianName;
            pm:hasAttribute ?musicianAttribute



}
```

- Output

musicianName	musicianAttribute
Freddie Mercury	lead vocals
Freddie Mercury	original
Roger Taylor	drums (drum set)
Roger Taylor	original
Roger Taylor	background vocals
Brian May	guitar
Brian May	original
Brian May	background vocals
Doug Bogie	bass guitar
Mike Grose	bass guitar
John Deacon	bass guitar
Barry Mitchell	bass guitar

3. Which songs were recorded in the year *1963*?
    - Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org./2000/01/rdf-schema#>
PREFIX pm: <https://raw.githubusercontent.com/jorge-arranz/Pop-Music-KE22/main/Ontology/popMusic.owl>

SELECT ?titleSong ?dateRecording
WHERE{
  ?Song a pm:Song;
          pm:hasTitle ?titleSong;
          pm:hasRecording ?Recording.
  ?Recording a pm:Recording ;
          pm:hasDate ?dateRecording;
  FILTER (?dateRecording > '1962-12-31') .
  FILTER (?dateRecording < '1964-01-01') .



}
```

- Output

titleSong	dateRecording
Twist and Shout	1963-02-11
Misery	1963-02-11
Misery	1963-02-20
There's a Place	1963-02-11
I Saw Her Standing There	1963-02-11
Baby It's You	1963-02-11
Baby It's You	1963-02-20
Do You Want to Know a Secret	1963-02-11
Anna (Go to Him)	1963-02-11
Chains	1963-02-11
Boys	1963-02-11
A Taste of Honey	1963-02-11

4. When was the song "*Bohemian Rhapsody*" released?
    - Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org./2000/01/rdf-schema#>
PREFIX pm: <https://raw.githubusercontent.com/jorge-arranz/Pop-Music-KE22/main/Ontology/popMusic.owl>

SELECT ?date
WHERE{
  ?Song a pm:Song;
        pm:hasTitle "Bohemian Rhapsody";
        pm:hasRecording ?Recording .
 ?Release a pm:Release;
          pm:containsRecordings ?Recording;
          pm:hasDate ?date

}
```

    - Output

date
1985
1985-11-28
1994-05-01

5. When are where did the band "The Beatles" born?
    - Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org./2000/01/rdf-schema#>
PREFIX pm: <https://raw.githubusercontent.com/jorge-arranz/Pop-Music-KE22/main/Ontology/popMusic.owl>

SELECT ?born ?place
WHERE{
  ?Band a pm:Band;
        pm:hasName "The Beatles";
        pm:hasBeginDate ?born;
        pm:hasPlace ?place


}
```

- Output

born	place
1957-03	GB

8. References

Blazegraph: https://github.com/blazegraph/database/releases

Getting started with RDF using Blazegraph: https://thejeshgn.com/2020/12/11/getting-started-with-rdf-using-blazegraph/#Import-some-data

OWLUnit: https://github.com/luigi-asprino/owl-unit/releases

Protégé Tutorial: http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf

SPARQL Anything Readthedocs:https://sparql-anything.readthedocs.io/en/latest/#command-line-interface

SPARQLAnything: https://github.com/SPARQL-Anything/sparql.anything/releases