Linked Jazz Music

Simone Persiani - simone.persiani2@studio.unibo.it
Marco Buiani - marco.buiani@studio.unibo.it
Nicolas Lazzari - nicolas.lazzari2@studio.unibo.it

October 18, 2022

Contents

1 Introduction

In the following document we will describe the design and implementation of an ontology for Jazz music tunes. The main objective of the ontology is that of linking content-based information - musical annotations - with their respective recording metadata. Content-based information will be extracted from the *Audio-Aligned Jazz Harmony* dataset (JAAH) [6] while metadata will be extracted from the MusicBrainz project (https://musicbrainz.org/).

The newly obtained information will help us get additional insights on how the content of a musical piece relates to the artist that collaborated within it. For instance, it will allow us to get answers for questions on the line of:

> *Within all the available tunes composed by Thelonious Monk, which artists played as drummers in tunes that contain sections in which the drum plays a central role?*

We can easily see how being able to answer this kind of questions, from a domain expert point of view, requires a deep knowledge on the artist's activity and works. In addition we would also be able to ask more domain-specific questions, such as:

> *Which pieces of Louis Armstrong contain the* A minor*,* C major *and* D minor *chords?*

Moreover, we will extend the information extracted from the MusicBrainz database by aligning our ontology with the Linked Jazz project developed by the Pratt Institute [18]. In this way we will be able to provide more meaningful information on artists relationships (for instance linking two different artists not only by their actual collaboration but also because of their possible friendship). This might give us the tools to explain and uncover influence networks that were previously difficult to infer.

Throughout the whole document we will follow the eXtreme Design methodology [2] as strictly as possible, motivating and illustrating our main decisions and giving an outline of the possible alternatives that we could've explored.

2 Requirement collection and formalization

In the following sections we analyze and collect all the relevant information from JAAH, MusicBrainz and Linked Jazz. For each dataset, we give a brief description of its contents, and provide an in-depth description of the data model on which we will be working on. We then extract exemplary stories based on the data and formalize a list of competency questions that will be later converted into test cases.

2.1 MusicBrainz

MusicBrainz is an open music encyclopedia that collects music metadata and makes it available to the public in a crowd-sourcing fashion, which is similar to the one employed by Wikipedia. Data in MusicBrainz is neatly categorized in hierarchical structures containing structured data. In the following sections we will only consider a subset of the available MusicBrainz classes, namely *artists* (section 2.1.1), *works* (section 2.1.2), *recordings* (section 2.1.3), *releases* (section 2.1.4) and *relations* between them (section 2.1.5). The data we will be working on has already been extracted as part of the Polifonia Project[1] and is available on GitHub[2].

2.1.1 Artist

According to MusicBrainz's terminology [13], an *artist* is generally a musician (or musician persona), a group of musicians, or another kind of music professional (like a producer or engineer). Occasionally, it can also be a non-musical person (like a photographer, an illustrator, or a poet whose writings are set to music), or even a fictional character.

Data model
Each individual of type *artist* is persisted in a data structure containing many properties. We'll list only the information which we considered relevant for our project.

- `id` = MusicBrainz ID
- `name` = official name
- `sort-name` = sorted name, for example "Beatles, The"
- `IPI` = Interested Party Information code
- `ISNI` = International Standard Name Identifier
- `alias-list` = a list of the alternative names or mispellings by which the artist can be referred to, each of them with the corresponding sorted version
- `type` = the type of artist: `Person`, `Group`, `Orchestra`, `Choir`, `Character` or `Other`
- `gender` = the gender of the artist expressed as a string, only present if `type` is equal to `Person`

Area and Time properties:
Begin/end dates (and areas) can be used to specify when (and where) an artist *"started/finished existing"*. They correspond to the date (or place) of birth/death (for people) or of formation/dissolution (for collectives).

Areas are individuals of the MusicBrainz class named `Area`, whose data structure may contain various properties. We are only interested in the name and the MusicBrainz ID.

- `area` = indicates the area with which an artist is primarily identified (it is often, but not always, its birth/formation country)
- `begin-area/end-area` = starting/finishing area
- `life-span` = starting/finishing dates

[1] https://polifonia-project.eu/
[2] https://github.com/polifonia-project/datasets/tree/main/dataset/jaah/cornucopia_data/musicbrainz

Example story
Ella Fitzgerald (MusicBrainz page) is a female artist born in Newport News on April 25 1917. Her main area of activity is the United States, where she went by many aliases: some of them cover for her apparently easy to misspell surname ("Fitzgerard", "Fiztgerald", "Fiztgerarld"), while others relate to her relevance in the Jazz landscape ("First Lady of Song", "Lady Ella", "Queen of Jazz"). Ella died in Beverly Hills on the 15th of June 1996 at the age of 79.

Reference: artist_ de6cd145-7a74-4e4b-86ac-4094433b849f_ 1.json.

Competency questions
We can formulate several different competency questions from the data of an artist:

- What is the *name* of the artist?
- By which *aliases* is the artist known for?
- Is the artist a *person*?
- When is the artist person born, or the group formed?
- What is the *main/begin/end area* of the artist?
- Is the artist a *group*? *When did it form*? *Did it dissolve*?

2.1.2 Work

According to MusicBrainz's terminology [17], a *work* is **a distinct intellectual or artistic creation**, which *can be expressed* in the form of one or more audio recordings. While a work in MusicBrainz is generally musical in nature, it isnot required to be so. For example, a work could be a novel, play, poem or essay, later recorded as an oratory or audio-book.

A work's **distinctiveness** is based on the artists who contributed to its final output and on whether a work is derived from another original work. Examples of works that are distinct:

- a work that is written by an individual songwriter
- a work that is the result of a collaboration between composer and lyricist
- an instrumental work where an artist later adds lyrics
- translation of an original work into a different language
- a parody, medley or mashup of multiple original works

Works are typically of two *types*:

- a **discrete work** is an individual song, musical number or movement. This includes recitatives, arias, choruses, duos, trios, etc. In many cases, discrete works are a part of larger, aggregate works.
- an **aggregate work** is an ordered sequence of one or more songs, numbers or movements, such as: symphony, opera, theatre work, concerto, and concept album. A popular music album is not considered a distinct aggregate work unless it is evident that such an album was written with intent to have a specifically ordered sequence of related songs (i.e. a "concept album").

Data model
Each individual of type *work* is persisted in a data structure containing many properties. We'll list only the information which we considered relevant for our project.

- `id` = MusicBrainz ID
- `type` = type of work, for example "song"
- `title` = title, expressed in the original language

- `language` = language of the lyrics, if they are present
- `ISWC` = International Standard Musical Work Code
- `alias-list` = a list of the alternative names or mispellings by which the work can be referred to, each of them with the corresponding sorted version

Example story
"Wrap Your Troubles in Dreams" (MusicBrainz page), also known as "Wrap Your Troubles in Dreams (and Dream Your Troubles Away)", is a song composed in 1931 by Harry Barris with lyrics by Ted Koehler and lesser known Billy Moll. It has been published by both Shapiro Bernstein & Co. Ltd. and Campbell Connelly Ltd. We can see that the first recording session happened on march 2nd 1931 by Bing Crosby, while McKinney's Cotton Pickers, Mildred Bailey and Louis Armstrong & His Orchestra all recorded their own version later that year. Recordings continued to take place until the year 2000.
Reference: work_177be3d6-14bf-3e12-87af-c1d7457b5c59.json.

Competency questions
We can formulate several different competency questions from the data of a work:

- What's the *title*, *lyrics language* and *ISWC code* of the work?
- Is the work known by some *alias*?
- Who are the *artists related* to this work? *Under which role* are they involved?
- Which are the *reported recordings* of this work? Recorded by whom? Recorded when?
- Which *artists* took part in a *recording* of this work?

2.1.3 Recording

According to MusicBrainz's terminology [15], a *recording* is the act of capturing an audio *track* for later processing such as **copying**, **mastering** or **editing**. To be able to produce a *release*, an *artist* must first of all go into a recording studio to record a performance of its *work*: the recorded *track* could then be used to create a *release* (for example by selling a CD inside which the track is copied as-it-is). Figure 1 shows an example of relations between original audio tracks (red boxes), recordings (orange boxes) and released tracks (green boxes).

Figure 1: An image taken from [15]

Data model
Each individual of type *recording* is persisted in a data structure containing many properties. We'll list only the information which we considered relevant for our project.

- `id` = MusicBrainz ID
- `title` = title given to the recording
- `length` = length of the recorded audio track, in milliseconds
- `ISRC` = International Standard Recording Code
- `artist-credit` = a list of artists that are credited for the recording
- `release-list` = a list of releases that make use of the recording

As far as the `length` is concerned, citing [15]:

> «It's only entered manually for standalone recordings. For recordings that are being used on releases, the recording length is the median length of all tracks (that have a track length) associated with that recording. If there is an even number of track lengths, the smaller median candidate is used.»

Example story
The jazz guitarist and composer John Leslie "Wes" Montgomery (rated 4 stars on MusicBrainz) is the author of an instrumental jazz piece called "West Coast Blues", which was produced under the "Taggie Music Co." label.

The recording titled "West Coast Blues (abridged)" (MusicBrainz page), which is credited to Montgomery himself, lasts around 5 minutes and 17 seconds and it was included (not exclusively) in the releases "The Smithsonian Collection of Classic Jazz, Volume 5" and "The Smithsonian Collection of Classic Jazz". The recording took place in the days between 26/01/1960 and 28/01/1960 (we don't know *where*) and it involved the following performers: Tommy Flanagan at the piano, Albert "Tootie" Heath at the drums, Percy Heath at the double bass and Wes Montgomery at the guitar.

Reference: recording_ 02ffe965-bd93-45bd-ae9f-5671261871ae.json.

Competency questions
We can formulate several different competency questions from the data of a recording:

- What's the *title/length/ISRC code* for a recording?
- Who is (are) the *credited artist(s)*?
- How many releases contains a track from a particular recording?
- What are the works that are concretized by a recording?

2.1.4 Release

According to MusicBrainz's terminology [16], a *release* is intended as a real-world object that can be found in stores. It has publishing date and country, a list of catalog number and label pairs, packaging type and release status [14].

Each release can be part of a release group, which represents an important distinction from a simple release. While a release represents the material concept (i.e. the actual CDs, vinyls, ...), a release group refers to the reification of that concept. Usually each release group has only one associated release but it might happen, for instance for highly popular albums, that several different edition are realized across many years and they all differ for some particular detail. The double CDs edition of The Fragile by Nine Inch Nails, for instance, is part of the homonymous release group, which is composed in a total of 18 different releases.

Each release is also composed of a medium and a track-list. A medium corresponds to the physical thing one would obtain when getting in possess of a release. For instance the before-mentioned Fragile release is composed of 2 CDs. Those releases whose distribution is only digital are classified as mediums if the creators divide the whole release in different disks. A medium is characterized by its format (refer to Release / Format on MusicBrainz) for an exhaustive list.

Data model
Each individual of type *release* is persisted in a data structure containing many properties. We'll list only the information which we considered relevant for our project.

- `id` = MusicBrainz ID
- `title` = title given to the release
- `status` = how "official" the release is, can take values such as `official`, `promotion`, `bootleg`, `pseudo-release`
- `date` = date in which a release was made available for the public. For live performances it doesn't refer to the performance's date but rather to the date the release was made available as a consumable medium
- `artist-credit` = artist(s) that the recording is primarily accredited to
- `barcode` = barcode
- `asin` = Amazon Standard Identification Number

Example story
The official 1994 US release of "Portrait of the Artist as a Young Man" (MusicBrainz page) by Louis Armstrong is made up of 4 CDs. Each CD has a track-list of at least 20 tracks and at most 21 tracks named in English. The release is issued by the label "Legacy". The product is available on Amazon with ASIN `B000002984`. The release is of normal quality with a non-ordinary packaging (*packaging field is "Other"*). The release is part of the 1994 album compilation "Portrait of the Artist as a Young Man". 98 artworks are included in the release.
Reference: release_ 00210ce8-2f59-4a9e-ab97-30c7334e67e7_ 10.json.

Competency questions
We can formulate several different competency questions from the data of a release:

- What's the *title/status/barcode/ASIN code/publishing date* of a release?
- Who is (are) the *credited artist(s)*?

2.1.5 Relations between entities

Each entity in the MusicBrainz dataset can be part of a binary relation with another entity. Some relations are characterized by additional information to allow complex n-ary relation to be modeled. For instance, the relation between *artists* and *recording* might contain information on the role of the artist in that particular recording and so on.

Data model
An exhaustive list of all the possible relations is available on the MusicBrainz documentation, along with the additional information each relation can have. In particular, we will make extensive use of the relations in the following list:

- Artist-Url, Work-Url, Recording-Url, Release-Url
- Recording-Release, Recording-Work

Competency questions
Many competency questions arise from the combinations of relation lists, for example:

- In what recording did this artist participate and playing what?
- With which artist did our artist collaborate for recordings?
- Is this artist *married with/sibling of* some other artist?

- In what works did this artist participate?
- With which artist did our artist collaborate for works?
- Which roles did the artist cover in the realization of artistic works?

2.2 JAAH

Introduction
JAAH is a dataset of time-aligned jazz harmony transcriptions [6] provided in a JSON format similar to JAMS [10]. The author of the dataset used a custom format instead of JAMS to annotate tunes, as it is easier to manually inspect by a human. For the very same reason we will make use of the same format.

The whole dataset is composed of 113 tunes extracted from two famous collections of jazz tunes, compiled by experts. Each tune has been manually annotated with a mixture of automatic (i.e. beat detection, tuning detection, length detection) and human annotation.

Data model
Each file, as we already mentioned, is a JSON file representing the whole annotation of the tune. Each file contains some metadata such as:

- track title
- artist name and its MusicBrainz id
- start and duration of the track
- tuning frequency of the track

The array `parts` contains the beats and chords annotations, which can recursively contain other array of the same type. Each element represents a hierarchical part of the track and is characterized by a name, the form of the part, if present, (e.g. AABA, ABAC) and the predominant instrument. The arrays `beats` and `chords` contains respectively the beat localization and the chord annotation of the whole track. Each element of the `beats` array represent the on-set start of the beat. Each element of the `chords` array represents a sequence of measure bars, delimited with pipes ($''|''$), in a similar fashion to lead sheet style. Each measure (composed of 4 beats for $\frac{4}{4}$ meter) is annotated. Chords are labeled using Harte notation [8].

Example story
Airegin, a track by the artist Tito Puente, is a $\frac{4}{4}$ tune in $A\flat$, $255.59s$ long, whose `A4` note is tuned to 439.23 in the analyzed track. The track is composed of 8 parts: *Intro*, *Head*, *Trumpet Solo*, *Trombone Solo*, *Tenor Sax Solo*, *Ensemble, Percussion Solo*, *Head* and *Coda* with respective forms, when applicable (here omitted for the sake of brevity). Each section has a list of annotated chords.

Reference: Airegin by Tito Puente.

Competency questions
We can formulate several different competency questions from the data from a JAAH file:

- What's the *key/time signature* of the track?
- Which are the *chords* that composes the track?
- Which parts composes the track?
- Which are the *main instruments* of the track?
- Which is the *opening chord*?

Dataset manipulation
In order to allow for an easier extraction of knowledge from the dataset, some preprocessing and data manipulation had to be performed. In particular, we transformed each `chords` element from a string, in the style of lead-sheet notation to a list of chords as in equation 1. We also extracted, from each `part` name (when applicable), the form (which we'll call pattern) and the main instrument as in equation 2. Extracting the form is easy, since we can just split it based on the presence of the character `'-'`. On the contrary, isolating the instrument from the section is trickier: in order to accomplish that, we relied on Ratcliff/Obershelp Pattern Recognition [1], taking the most similar string (at least 75% match). We compared each string with the instruments contained in MusicBrainz (see the related documentation). Whenever an instrument is found, the corresponding MusicBrainz ID is also added.

$$\text{“|F:min|F:7|Bb:min7|F:7|"} \longrightarrow [\text“}F : min\text{”}, \text“}F : 7\text{”}, \text“}Bb : min7\text{”}, \text“}F : 7\text{”}] \tag{1}$$

$$\begin{aligned} \text{"Trumpet Solo - ABAC"} \longrightarrow & \texttt{name} = \texttt{trumpet} \\ & \texttt{pattern} = \texttt{ABAC} \\ & \texttt{mbid} = \texttt{1c8f9780-2f16-4891-b66d-bb7aa0820dbd} \end{aligned} \tag{2}$$

2.3 Pratt

Linked Jazz [18] is a research project at the Pratt Institute's Semantic Lab. It investigates the application of Linked Open Data technologies to digital cultural heritage materials. The project draws on jazz history materials in a digital format to expose relationships between musicians and reveal their community network. The dataset we use is hosted on triplydb at https://triplydb.com/pratt/linked-jazz/.

Data model
The dataset consists of 3.635 different people and a total of 5.831 personal and musical relations among them. Individual IRIs mainly come from DBpedia [11], while a minor part is from linkedjazz.org. Every individual is given a name, while most of them also have a related image from linkedjazz.

The following are some examples of triples from this dataset:

- <http://dbpedia.org/resource/Ella_Fitzgerald>
  <http://dbpedia.org/ontology/thumbnail>
  <http://linkedjazz.org/image/square/Ella_Fitzgerald.png>
- <http://dbpedia.org/resource/Ella_Fitzgerald>
  <http://xmlns.com/foaf/0.1/name>
  "Ella Fitzgerald"@en
- <http://dbpedia.org/resource/Oscar_Peterson>
  <http://purl.org/vocab/relationship/friendOf>
  <http://dbpedia.org/resource/Ella_Fitzgerald>

Relations between artists are represented using the RDF predicates from the following list:

- <http://purl.org/vocab/relationship/mentorOf>
- <http://linkedjazz.org/ontology/bandmember>
- <http://purl.org/vocab/relationship/friendOf>
- <http://linkedjazz.org/ontology/playedTogether>
- <http://purl.org/vocab/relationship/hasMet>
- <http://purl.org/vocab/relationship/knowsOf>
- <http://purl.org/vocab/relationship/acquaintanceOf>

- <http://purl.org/ontology/mo/collaborated_with>
- <http://purl.org/vocab/relationship/influencedBy>
- <http://linkedjazz.org/ontology/touredWith>
- <http://linkedjazz.org/ontology/bandLeaderOf>
- <http://linkedjazz.org/ontology/inBandTogether>

3 Ontology

In the following we will describe the approach we pursued to develop an ontology that would model the information described in section 2. We will try to reuse other ontologies as much as possible. This is done mainly to ensure a good quality of the conceptualization, both by integrating existing resources that we deem as good and by avoiding reimplementation of already modeled concepts. In section 3.1 we will describe the resources that we ended up using.

At the same time, we will try to abstract from the actual data presented in the previous section in order to build an ontology that will hopefully be able to better generalize the description of real-world musical entities. Thus, we will try as much as possible not to blindly replicate the data-structures employed by MusicBrainz and we will even discard some bits of information that is only relevant in the context of the MusicBrainz ecosystem (for example, the internal annotations and disambiguation comments left by the crowdsourced contributors).

3.1 Dependencies

DUL

Throughout the whole ontology development we relied and committed to the DUL [9] ontological framework. DUL is based, mainly, on DOLCE [3] and D&S [7] formal theories. Hence, it builds on six main concepts: *Object* (*Endurant*), *Event* (*Perdurant*), *Quality* and *Abstracts* from DOLCE and *Situation* and *Description* from D&S.

By means of an over-simplification, the *Object* class can be thought of as the class of those entities (including social entities) that exists independently of the time in which they are considered, for instance people. On the contrary, the *Event* class represents those entities that only exist within a finite time interval, as the intuitive definition of event suggests.

Quality entities are characterizations of entities that are inherently part of them, for instance the color of an object is one of its qualities. The values of a quality are represented by *Region* entities. Hence, when modeling, it's important to distinguish whether to model an information as a quality for a certain entity or as an abstract entity by itself.

Description and *Situation* are trickier to summarize in few words. We can essentially consider those aspects as a way to model those information that are ambiguous when used in different contexts. Generally, while a *Situation* satisfies a *Description*, the latter can define a *Concept* which might be used to classify an entity.

For example, in the musical domain, we might consider three artists recording a track together as a *Situation*. The fact that they are indeed playing together to ultimately produce a track is the *Description* of that *Situation*. This description defines the *Concept* of recording, which can ultimately be used to classify an entity: the track that will be produced by the musicians.

The *Description* reifies the intentional aspect that describes the overall situation. It could happen, for instance, that the same three artists gather to play together without recording any track. Even though this is a similar *Situation*, it doesn't satisfy the *Description* that defines the concept of recording. Hence the music that they played won't be classified as a recording, even though it is the result of a similar happening.

Music Ontology

The information that we described on section 2.1 can be partially described by means of Music Ontology (MO) [19]. Citing the official website[3]:

> The Music Ontology provides a vocabulary for publishing and linking a wide range of music-related data on the Web.

Whenever possible we aligned our ontology to MO. Note that, since we committed to DUL, we can't express strict class equalities. That is because MO doesn't explicitly commit to DUL and, by expressing equivalence with our formalization, we would have ended up classifying MO entities possibly in an unexpected way. We made use of the OWL2 `equivalentClass`[4] axiom, which states that two different classes have the same set of individuals.

[3] http://musicontology.com/docs/faq.html

[4] https://www.w3.org/TR/owl-ref/#equivalentClass\protect\discretionary{\char\hyphenchar\font}{}{}def

Chord Ontology

Information described in 2.2 needs an ontology built specifically to model the musical domain, in particular the one that deals with annotations. We tried to tackle this problem within our own ontology but we had to rely on the Chord Ontology to describe chords. Chord ontology builds specifically on Harte notation [8] and encodes the rules that forms a chord.

In addition to the plain Chord Ontology, we also made use of the Music Theory Ontology[5], as it contains formal specifications useful to describe musical notation such as intervals, time signatures and chord progressions.

Vocab, FOAF and Schema

When using the data described in section 2.3 we are forced to indirectly make use of the Vocab ontology[6], which contains common predicates to express relationships between agents. In addition, for a similar purpose, we made use of FOAF [4] to describe relationships between agents. We also made use of the vocabulary defined by schema.org[7] whenever applicable to our ontology.

[5] https://github.com/tetherless-world/MusicTheory
[6] https://vocab.org/
[7] https://schema.org/

3.2 Design of the main ontology

In this section we will provide some generalities about the development choices we adopted while working on our main ontology. We chose to name this ontology as "KEJazz", which stands for "Knowledge Engineering Jazz". In order for it to be able to describe musical annotations made over a `:Recording`, it has to import the other ontology developed by us (namely, "chord-annotations"). A rough schematisation of the import relations between ontologies is provided in Figure 2.

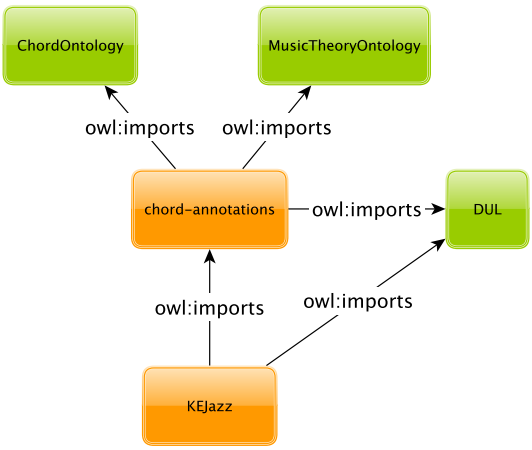

Figure 2: Imports Schema

The KEJazz ontology aims at describing the main aspects of musical artists activities, mainly focusing on the production of works and on the participation in recording sessions. Therefore, we created the following four classes that have more or less the same meaning as the corresponding homonymous MusicBrainz classes:

- `:Work`;
- `:Recording`;
- `:Release`;
- `:Artist`.

The `:Artist` class is further categorized into the two `:ArtistPerson` and `:ArtistGroup` subclasses, respectively thought for individuals (e.g. Louis Armstrong) and groups of artists which go by a single denomination (e.g. John Coltrane Quartet).

Some additional classes were put in place with the goal of representing n-ary relations between the previously-listed classes:

- `:ArtistContributionToWork`, which links a work to the artists that contributed to it, with the artist's role (encoded as an individual of the `:RoleInWork` class) and the `dul:TimeInterval` during which the contribution happened;
- `:ArtistPerformanceInRecording`, which links a recording to the artists that participated in it, with the instrument played by each artist (encoded as an individual of the `:MusicalInstrument` class) and the `dul:TimeInterval` during which each artist's participation to the recording happened;
- `:RelationBetweenArtists`, which links two artists with the type of `dul:SocialRelation` that existed between them (encoded as an individual of the `:RelationBetweenArtistsType` class) and the `dul:TimeInterval` during which such relation was valid.

3.2.1 Differences with respect to the MusicBrainz datamodel

Many data properties that we initially included in the ontology were later removed, since we realized that they were either too tied to the MusicBrainz data model or they felt irrelevant for the scope of our project. Examples of those properties are the following:

- the sorted versions of names (`sort-name`);
- the internal annotations and disambiguation comments left by the crowdsourced contributors;
- the lyrics language code for the works that include lyrics;
- the country for releases and artists (since we didn't find enough documentation for it on the MusicBrainz's website);
- the type of each work;
- the packaging format of a release;
- the artist credit phrase for releases and recordings.

Other data properties included in our ontology consist in various forms of identifiers for the main types of entities. For example, an `:Artist` can be identified through a MusicBrainz ID (`:hasMusicBrainzID`), an IPI (`:hasIPI`) or an ISNI (`:hasISNI`). Moreover, inside the data coming from MusicBrainz, for each entity it is possible to find a list of related URLs.

In addition to the creation of `owl:sameAs` triples that link an entity with its related URLs, we chose to convert the identifiers into URLs whenever it was possible (for example, it was possible to construct a URL by concatenating the "https://isni.org/isni/" string with the ISNI of the artist). Finally, we added `rdfs:label` properties to classes that already have a data property which is amenable for this objective (for example, we used the value of the `:hasTitle` property of works in order to define a label for them).

In the following part of this section we will briefly review the main Ontology Design Patterns that we employed for the design of our ontology, which is an opportunity for going deeper in the explanation of its details.

3.2.2 Work - Recording: Information Realization

We model the relation between `:Work` and `:Recording` entities with the Information Realization content pattern, as shown in Figure 3. The pattern clearly identifies `:Work` as a `dul:InformationObject`, a piece of information independent from its concrete realization. `:Recording` is the concrete `dul:InformationRealization` of `:Work`, as it consists in the `dul:Event` of recording a musical performance into a digital or analog medium.

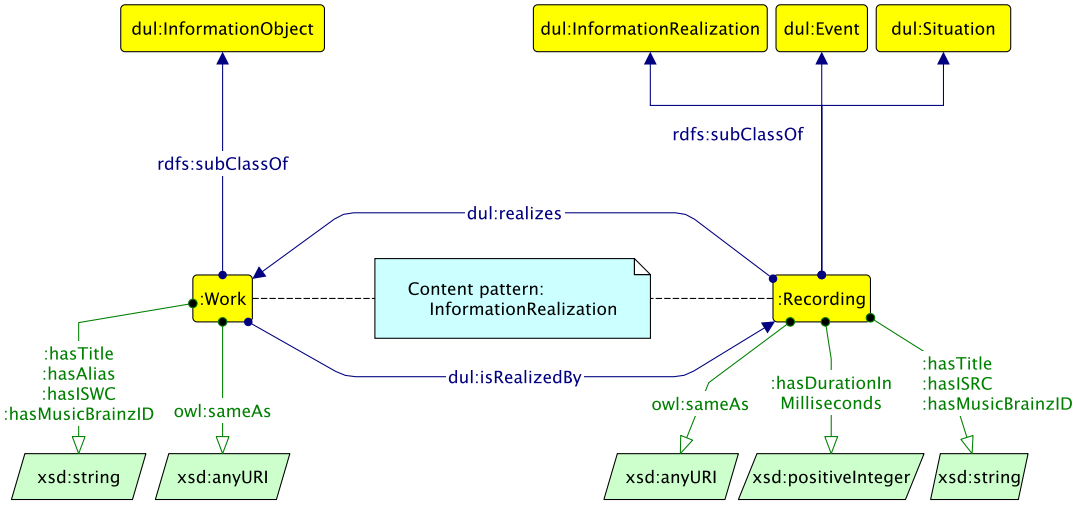

Figure 3: **Work - Recording**: Information Realisation

3.2.3 ArtistContributionToWork: TimeIndexedParticipation With Role

An `:ArtistContributionToWork` is the act of a single `:ArtistPerson` to contribute to the content of a musical `:Work` with its specific role, being it "Composer", "Lyricist", etc. This entity is modeled as a Time Indexed Participation Situation with the additional `:involvesRole` object property, thus being a `dul:Classification` of the `:ArtistPerson` over time. It involves the `:ArtistPerson`, its `:RoleInWork`, the target `:Work` and a `dul:TimeInterval`. A co-participation relation exists between the artist and its role, and can be inferred with the property chain shown in Figure 4.

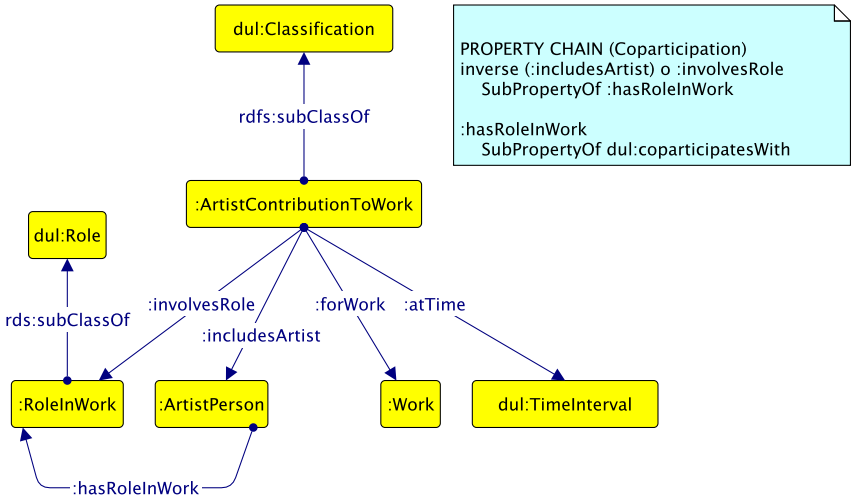

Figure 4: **ArtistContributionToWork**: TimeIndexedParticipationWithRole

3.2.4 ArtistPerformanceInRecording: TimeIndexedParticipation With Object

Similarly to the previous case, `:ArtistPerformanceInRecording` (see Figure 5) models the participation of an `:ArtistPerson` in a recording `dul:Event`, also specifying the played instrument (which can also be the "vocal apparatus", in case of a singer).

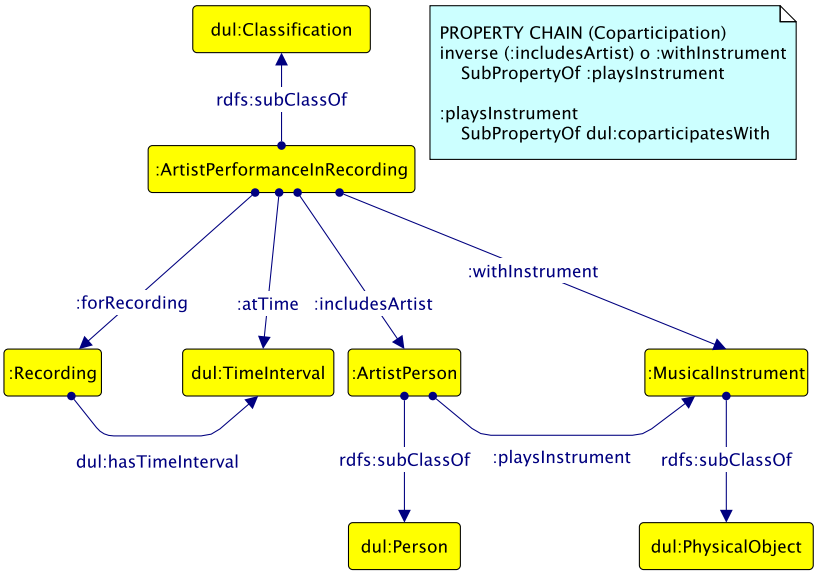

Figure 5: **ArtistPerformanceInRecording**: TimeIndexedParticipationWithObject

3.2.5 ArtistGroup: ExhaustivePartition

We can model the `:ArtistGroup` class as an Exhaustive Partition, as shown in Figure 6. This means that instances of the class must belong to one and only one of its subclasses, namely `:Orchestra`, `:Choir` and `:Band`, which are all disjoint.

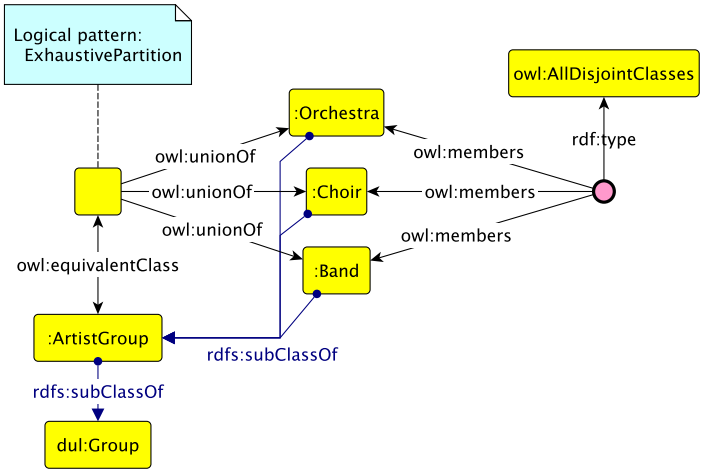

Figure 6: **ArtistGroup**: ExhaustivePartition

3.2.6 RelationBetweenArtists: TimeIndexedRelation

`:RelationBetweenArtists` is a `dul:TimeIndexedRelation`, a `dul:Situation` which includes a `dul:TimeInterval` for its setting. It is non-symmetric, thus identifies two different artists (persons or groups) as being the source and the target of the relation. A symmetric co-participation relation (`:inRelationWith`) occurs between the artists involved, as shown in Figure 7.

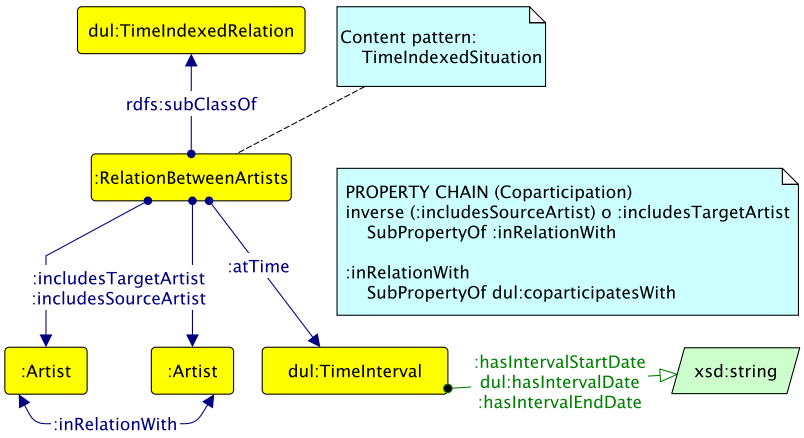

Figure 7: **RelationBetweenArtists**: TimeIndexedRelation

3.2.7 RelationBetweenArtists - RelationBetweenArtistsType: ClassesAsValues

The `:RelationBetweenArtists` *Situation* satisfies the `:RelationBetweenArtistsType` *Description*, which is a specific type of `dul:SocialRelation` between artists. With the Classes As Values pattern we can express relations as individuals, but also talk about individual relation types, as shown in Figure 8.

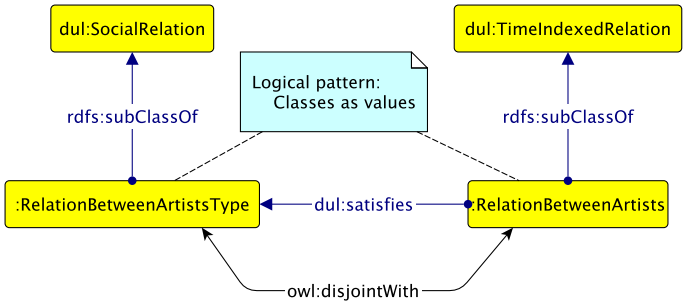

Figure 8: **RelationBetweenArtists - RelationBetweenArtistsType**: ClassesAsValues

3.3 Chord Annotations Ontology

3.3.1 Design

The design of the ontology describing annotations revolves mainly on the concept of musical sheet. Here we talk about a musical sheet whenever we have a description of the harmonic structure of a recording. Each musical sheet – as it can be seen in Figure 9 – classifies the recording with a key and an overall time signature. Note that here, for simplification purposes, we only take into account musical pieces with a constant time signature.

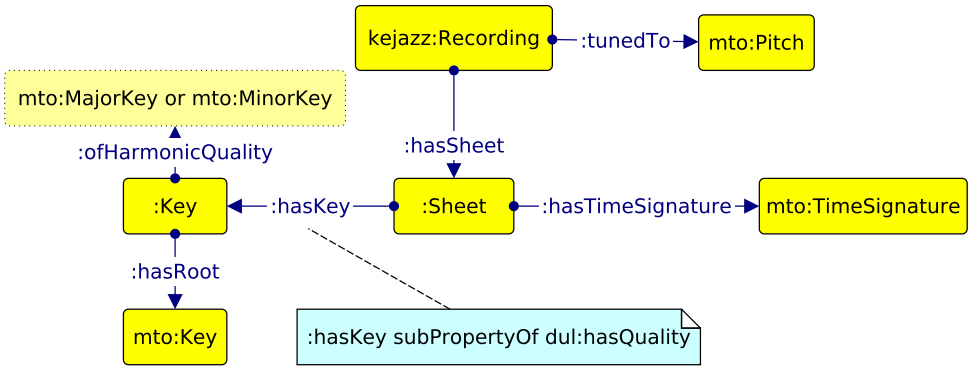

Figure 9: Closeup of the relations involving a recording

Each musical sheet is composed of different sections, such as *intro* or *verse*. A section is classified by the main instruments that partakes in it and the pattern of that section. Each section is composed of a progression, whose elements are bars. Finally, each bar may contain several different chords. The relevant part of the ontology is shown in Figure 10.

`:Sheet`, `:Progression` and `:Bar` are modeled as `dul:Collection`. This results in the instantiation of a containment design pattern. `:Section` is further defined to be an `dul:InformationObject` which is characterized by the abstract information entity `:Pattern`.

Even though the proposed formalization doesn't allow to model sections composed by other subsections (as in the case of a verse made up of two main parts), it's trivial to "*linearize*" those hierarchies to be used on the proposed ontology.

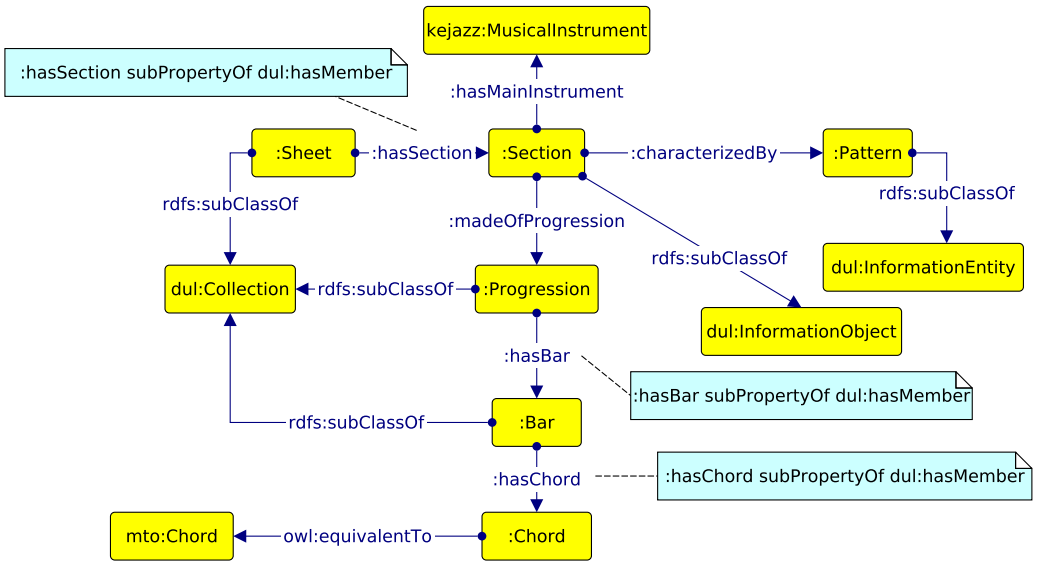

Figure 10: Entities and relations between sections, progressions and bars

`:Section`, `:Progression` and `:Bar` are sub-classes of the `:Annotation` class (see Figure 11).

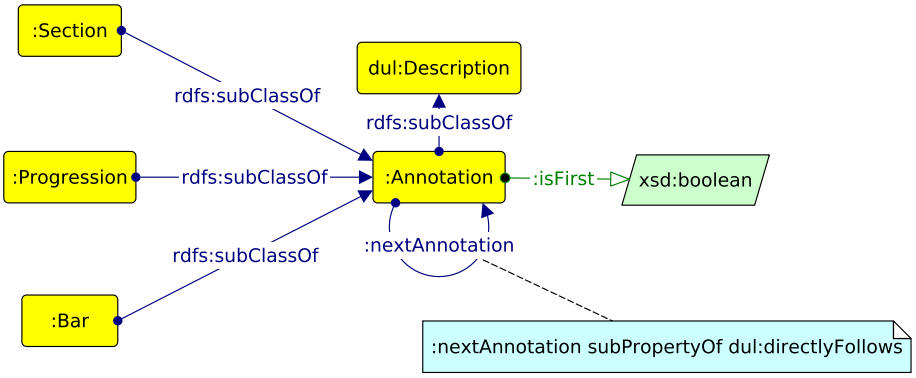

Figure 11: Annotation entity

An `:Annotation` is the primitive building block in a `:Sheet`. It is produced either by a human or an algorithm and it represents the annotation of a particular part of a recording. For this reason, it represents a *description* in the context of the DUL framework.

Chaining multiple annotations together builds up the complete musical sheet. Hence, each annotation has a reference to its following annotation (`:nextAnnotation`) and maintains a binary property which encodes whether it represents the first annotation of a chain (`:isFirst`) or not.

3.4 Mapping

After the definition of our ontology we focussed on populating it with resources and properties from the collected data sources. In the Knowledge Engineering field this process is referred to as Mapping. For this purpose we make use of SPARQL-Anything [5], a system that allows to execute SPARQL queries over data extracted from several different file formats. Our work consisted in defining different mapping queries to adapt to the various types of inputs.

The tool is very flexible: we managed to fill the ontology without sacrificing any property present in the data. Input files are processed one-by-one to consume all necessary information. We subdivide input files into three categories, based on similarities of the mapping strategy: "main entities", "relations", and "chord annotations".

3.4.1 Mapping Main Entities

Our data source has a file entry for each single Artist, Work, Recording and Release. The mapping for these main entities with SPARQL-Anything shares many similarities and proceeds as follows:

1. match the root element of the JSON file – which corresponds to the main entity (Artist, Work, etc.) – to a temporary blank node;
2. match the mandatory MusicBrainzID data property, which is essential for the identification of the entity and is used for the construction of a unique resource IRI;
3. match every other individual data property of the entity, whose presence could be optional;
4. produce an IRI from the unique MusicBrainzID, following RDF specifications;
5. rename the entity's temporary blank node with the produced IRI using the BIND keyword;
6. finally populate an ontology individual by SPARQL CONSTRUCT, with retrieved input information mapped to ontology relations.

This procedure allows us to retrieve the maximum amount of data from the available files, since optional matching of non essential properties makes it robust to missing or incorrectly labelled values in the JSON source file.

3.4.2 Mapping Relations

Our ontology defines relations among three pair of entities that are populated by the respective mapping queries:

- `:RelationBetweenArtists` ⟷ `artist-artist-relation.sparql`
- `:ArtistContributionToWork` ⟷ `artist-work-relation.sparql`
- `:ArtistPerformanceInRecording` ⟷ `artist-recording-relation.sparql`

Information on relations can be extracted from the same "main entities" files as before, however the focus is now on retrieving items from the relation-lists part. The procedure works as follows:

1. match the root element of the JSON file – which corresponds to the source or target entity of the relation (Artist, Work, etc.) – to a blank node;
2. match the mandatory MusicBrainzID data property of the node;
3. match all items of the relation-list to temporary blank nodes, by using the `fx:anySlot` Facade-X primitive;
4. match every other data property of source and target entities of the relation;
5. finally populate the data properties of the relation itself, such as the time interval at which the relation is valid and the type of relation;

6. the direction value of each relation ("forward"/"backward") is used to select whether the main entity is source or target of the matched relation (the UNION keyword allows the matching of either case);

7. BIND IRIs for source and target entities;

8. BIND the IRI for the type of relation of each matched relation;

9. finally populate the relations and participants entities with the SPARQL CONSTRUCT query.

As we did for "main entities", OPTIONAL matching makes the query robust to mislabelled or missing values, allowing to extract the maximum amount of relations from the data. We chose to match all expressed relations, even if not completely described by all properties.

3.4.3 Mapping Chord Annotations

Each file from the set of annotations provided by the JAAH dataset is mapped to the proposed chord ontology by making use of the file `chord-annotations.sparql` as follows:

1. information related to the recording are first extracted, together with music-content metadata on the recording such as the piece tuning, and linked to the recording

2. a `Sheet` entity is instantiated and linked to the main recording. Each information related to it is linked with the appropriate relation.

3. `Section`, `Progression` and `Bar` entities are then extracted with the relevant information. Particular attention is payed to the relations from `Annotation` entity: in order to build the appropriate chain of annotations, the relation `nextAnnotation` and `isFirst` are carefully handled. In particular, the Facade-X `next` primitive is used to obtain the next annotation in the original list of elements, its value is then used to craft the required relation, mantaining consistency between the data model and the produced triples.

3.4.4 Considerations

Producing coherent mapping queries helped us refine some details of the ontology itself. When a query would require too much complexity to work, often the reason was that we were trying to comply too strictly with the MusicBrainz datamodel, thus failing to generalize with respect to the modelled domain.

For instance, MusicBrainz expresses aliases with an entity of its own, which has its name, sort-name, locale, and possibly time interval to specify the name change over time of an artist or venue. To match all this would result in a very complex query that is exceptionally data specific. This suggested us that a simplification was possible, and so we decided to include aliases as a simple string data property of our entities. With this trade-off we can be more general, while still including in our ontology the flexibility offered by aliases: searching for misspelled, renamed or foreign language entities.

In a linked open data philosophy, we decided to align our entities with the IRIs of the other ontologies that are present in the data. The data includes many links (URLs) to external sources that are not expressed in RDF. We produce `owl:sameAs` relations with all of them.

Files are processed one by one by replacing the input path in the mapping query: this is performed by binding the `?_name` variable in the query (`fx:properties fx:location ?_name`). We ran all the different queries with the help of a Python script called `mapping.py`.

3.5 Testing

Along-side the development process of the ontology and the mapping of the data on it we continuously tested our ontology both manually and automatically. For the automation of tests we made use of the owl unit suite[8]. OWL Unit builds upone the OWL Unit ontology, which allows developers to define different type of tests on an ontology such as competency question verification, annotation verification, inference verification [12]. Our testing suite contains 25 tests: 23 competency question verification tests and 2 annotation verification tests. Competency question verification spans most of the competency questions defined on previous sections.

pyowlunit During the development of automatic tests we stumbled upon some configuration and runtime issues when using the official OWLUnit tool. For this very reason we decided to build our own OWLUnit client: `pyowlunit`[9]. `pyowlunit` is written in Python and makes use of the same libraries used by the original OWLUnit tool, through the java bridging library JPype[10]. Differently from OWLUnit, pyowlunit outputs more detailed error messages on why and how errors failed, in particular when modeling competency question verification tests. Additionally, it allows developers to test ontology on the fly, without publishing the sources on an active endpoint. This enables the user to write tests in an easier and faster way. Even though we initially planned on supporting only competency questions, we eventually decided to give support to the full set of tests than can be formalized using OWLUnit's ontology.

[8] https://github.com/luigi-asprino/owl-unit
[9] https://github.com/KEGP/pyowlunit
[10] https://github.com/jpype-project/jpype

4 Entity linking

Once all the data from the json files has been loaded into our graph we performed linking with the Pratt graph, described in section 2.3.

In order to do so we had to perform some entity linking based on the attributes we have at our disposale.

We tried different linking techniques:

- exact match between artists names
- partial match using edit distance

Exact match between artists names doesn't yield good results. The problem is that sometimes an artist name is joined with its band name, for instance *Charlie Parker* might be represented ad *Charlie Parker All-Starts*. This prevents the method from exactly matching both entities, even though they represent the same artist. Out of 2005 artist on Pratt graph and 510 only 231 can be perfectly aligned. To improve on the previous result the most simple thing to do is to rely on a distance measure between two different names. We employed the edit distance algorithm to measure the similarity between two artist names. This gives us some additional artists match, giving us a total of 241 matches.

The correctness of match however isn't easy to assess. Homonyms might lead us to a wrong path and performing a reliable entity linking phase by making use of only names is not a sound approach. To validate the correctness of our linking we used additional information at our disposal. In particular, MusicBrainz data contains date of birth, place of birth and, if present, date and place of death for each artist. The same is present in the Pratt graph since entities are directly linked to DBPedia entities and we can thus query the DBPedia graph to obtain such information. We considered as valid only those links whose described information matched.

We finally ended up with 189 matches between entities in our graph.

5 Demo

We developed a simple web page for our ontology in order to test its content and provide some example of useful queries. The service is called *Playing Together Brings Together* and, as the name suggests, its aim is that of finding links between different artists that played together in the recording of a track. In particular we make use of the `friendOf` predicate from the relationship[11] ontology to find friendship bonds between different artists. On listing 1 the main query used by the website is written. The query makes extensive use of `owl:sameAs` predicate to retrieve linked artists. The query first retrieve all recording performances from an artist and its linked entities. It then does the same for all the artist that has been part of the same recording. Friendship relationship between artists are then selected. Overall the query takes some time to execute, given the fact that some reasoning needs to be performed, in particular to compute the sameAs semantic. To increase performances the result is cached, as it is not going to change unless new entities are inserted in the graph.

Listing 1: Relationship query

```
PREFIX kejazz: <http://localhost:8080/ontology/kejazz/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX relationship: <http://purl.org/vocab/relationship/>
PREFIX dul: <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#>
SELECT DISTINCT ?recordingIRI ?recordingName ?from ?to
                ?artistIRI ?artistName ?artistInstrument
                ?otherIRI ?otherName ?otherInstrument
WHERE {
    ?recordingIRI kejazz:hasTitle ?recordingName .
    ?recordingPerformance kejazz:forRecording ?recordingIRI ;
                          kejazz:includesArtist ?artistIRI ;
                          kejazz:withInstrument [
                            kejazz:hasName ?artistInstrument
                          ];
                          kejazz:atTime [
                            kejazz:hasIntervalStartDate ?from ;
                            kejazz:hasIntervalEndDate ?to
                          ] .

    ?artistIRI kejazz:hasName ?artistName ;
               owl:sameAs ?sameAs .

    ?otherIRI kejazz:hasName ?otherName ;
              owl:sameAs ?sameAsOther .
    ?otherRecordingPerformance kejazz:forRecording ?recordingIRI ;
                               kejazz:includesArtist ?otherIRI ;
                               kejazz:withInstrument [
                                 kejazz:hasName ?otherInstrument
                               ] .

    ?sameAs (relationship:friendOf|^relationship:friendOf) ?sameAsOther .
}
```

To extract an image thumbnail of an artist we perform a federated query toward DBPedia. The corresponding query is shown in listing 2 where `ARTIST_IRI` is replaced with the artist we want the image of.

Listing 2: Image retrieval query

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?image WHERE {
BIND(<ARTIST_IRI> AS ?artist) .
?artist owl:sameAs ?dbpedia .
SERVICE <https://dbpedia.org/sparql/> {
  ?dbpedia dbo:thumbnail ?imageCropped .
  BIND(REPLACE(STR(?imageCropped), "width=300", "") AS ?image) .
}
FILTER(CONTAINS(STR(?dbpedia), "dbpedia")) .
}
```

[11] http://purl.org/vocab/relationship/

References

[1] Black, P. E. *Ratcliff/Obershelp pattern recognition*, vol. 8. 2021.

[2] Blomqvist, E., Hammar, K., and Presutti, V. Engineering ontologies with patterns-the extreme design methodology. *Ontology Engineering with Ontology Design Patterns*, 25 (2016), 23–50.

[3] Bottazzi, E., and Ferrario, R. Preliminaries to a dolce ontology of organisations. *International Journal of Business Process Integration and Management 4*, 4 (2009), 225–238.

[4] Brickley, D., and Miller, L. Foaf vocabulary specification 0.99. *From http://xmlns. com/-foaf/spec* (2014).

[5] Daga, E., Asprino, L., Mulholland, P., and Gangemi, A. Facade-x: An opinionated approach to sparql anything. In *Volume 53: Further with Knowledge Graphs*, M. Alam, P. Groth, V. de Boer, T. Pellegrini, and H. J. Pandit, Eds., vol. 53. IOS Press, August 2021, pp. 58–73.

[6] Eremenko, V., Demirel, E., Bozkurt, B., and Serra, X. Jaah: Audio-aligned jazz harmony dataset, June 2018.

[7] Gangemi, A., and Mika, P. Understanding the semantic web through descriptions and situations. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (2003), Springer, pp. 689–706.

[8] Harte, C., Sandler, M. B., Abdallah, S. A., and Gómez, E. Symbolic representation of musical chords: A proposed syntax for text annotations. In *ISMIR* (2005), vol. 5, pp. 66–71.

[9] Hitzler, P., Gangemi, A., and Janowicz, K. *Ontology engineering with ontology design patterns: foundations and applications*, vol. 25. IOS Press, 2016.

[10] Humphrey, E. J., Salamon, J., Nieto, O., Forsyth, J., Bittner, R. M., and Bello, J. P. Jams: A json annotated music specification for reproducible mir research. In *ISMIR* (2014), pp. 591–596.

[11] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web 6*, 2 (2015), 167–195.

[12] luigi asprino. owl-unit, Aug. 2022. [Online; accessed 2. Aug. 2022].

[13] MusicBrainz. https://musicbrainz.org/doc/Artist, Feb 2022. [Online; accessed 07. Feb. 2022].

[14] MusicBrainz. https://musicbrainz.org/doc/MusicBrainz_Entity, Jan 2022. [Online; accessed 25. Jan. 2022].

[15] MusicBrainz. https://musicbrainz.org/doc/Recording, Feb 2022. [Online; accessed 05. Feb. 2022].

[16] MusicBrainz. https://musicbrainz.org/doc/Release, Feb 2022. [Online; accessed 07. Feb. 2022].

[17] MusicBrainz. https://musicbrainz.org/doc/Work, Feb 2022. [Online; accessed 07. Feb. 2022].

[18] Pattuelli, M. C., Weller, C., and Szablya, G. Linked jazz: an exploratory pilot. In *International Conference on Dublin Core and Metadata Applications* (2011), pp. 158–164.

[19] Raimond, Y., Abdallah, S. A., Sandler, M. B., and Giasson, F. The music ontology. In *ISMIR* (2007), vol. 2007, Citeseer, p. 8th.