Knowledge Graph Construction From MusicXML: An Empirical Investigation With SPARQL Anything

Conference or Workshop Item

Knowledge Graph Construction from MusicXML: an empirical investigation with SPARQL Anything*

Marco Ratta[1,*], Enrico Daga[1]

[1]The Open University, Walton Hall, Kents Hill, Milton Keynes (UK)

Abstract

Multimodal knowledge graphs are gaining momentum because of their ability to integrate multiple types of representations. In particular, Musical Heritage Knowledge Graphs combine rich contextual information – metadata from encyclopedic KGs, with symbolic content – the scores encoded in a music ontology. In this paper, we explore the application of SPARQL Anything – a tool for façade-based knowledge graph construction (KGC) – for integrating musical content encoded in MusicXML. Specifically, we investigate the hypothesis that SPARQL is flexible enough to handle relevant tasks for musical knowledge graph construction such as (a) extracting melodic information, (b) extracting N-grams of musical information, (c) supporting the analysis of those N-grams and (d) populate a musical note ontology. We contribute a collection of reusable queries for extracting musical features from MusicXML files to construct Musical Knowledge Graphs. Crucially, we discuss friction points in using the façade-based approach (either in querying the façade or transforming the data) and provide recommendations on how to improve the usability of SPARQL for musical KGC tasks.

Keywords

SPARQL, Knoweldge Graph, Façade-X, MusicXML

1. Introduction

Multimodal knowledge graphs (KGs) are gaining momentum because of their ability to incorporate multiple types of representations, thus enabling complex feature engineering tasks that support machine and/or deep learning methods [1]. Multi-modality is recognised to be crucial for music information processing applications [2]. A paradigmatic case of multi-modal KGs are Musical Heritage Knowledge Graphs, which combine rich contextual information – metadata from encyclopedic KGs, with symbolic content – the scores encoded in a music ontology. We place our research in the context of the EU H2020 project Polifonia, aiming at developing a portal to support different cohorts of musical stakeholders (scholars, citizens, musicians) supported by a unified, multimodal knowledge graph [3, 4] populated with crowdsourcing [5], textual documentary evidence [6, 7], and knowledge graph construction (KGC) pipelines. Here we focus on KGC from structured symbolic music.

However, building such KGs require complex processing tasks, typically performed with a combination of libraries and intermediate formats. A recent approach to knowledge graph construction (KGC) proposes a façade for querying a multitude of formats through SPARQL, allowing KG practitioners to directly use their querying expertise for integrating non-RDF content [8, 9]. In this paper, we therefore explore the possibility of applying SPARQL Anything – a tool for façade-based KGC – for querying musical content encoded in the MusicXML format. Our motivation for this work stems from the fact that encoded music is an excellent case study because of (a) the multiplicity of representations required to solve computational musicology tasks, (b) the heterogeneity of existing formats and representations, mixing musical symbolic content and textual content (e.g. metadata), and because (c) sequential information is important but is typically neglected by KGC frameworks. Specifically, we wish to investigate the hypothesis that SPARQL Anything is flexible enough to handle relevant tasks for musical knowledge graph construction such as (a) extracting melodic information, (b) extracting N-grams of musical information, (c) supporting the analysis of those N-grams and (d) populate a musical note ontology.

We contribute a collection of reusable queries for extracting musical features from MusicXML files to construct musical KGs and to function as a basis for achieving more complex musicology tasks. Further, we explore issues in query design, investigating the hypothesis that the extended set of functions for working with container membership properties and sequential information of SPARQL Anything can help in designing more compact queries and aid overcoming specific technical challenges. Finally, we discuss friction points in using the façade-

Music Heritage Knowledge Graphs (MHKG). This workshop is part of the 21st International Semantic Web Conference, 23-27 October 2022, Hangzhou, China

* Conceptualization, Software, Investigation and Writing, M.R. and E.D.; Validation and Formal Analysis, M.R.; Supervision, Project Administration and Funding Acquisition, E. D.

*Corresponding author.

marco1791@protonmail.com (M. Ratta);
enrico.daga@open.ac.uk (E. Daga)
https://github.com/MarcoR1791 (M. Ratta);
http://www.enridaga.net (E. Daga)
0000-0003-3788-6442 (M. Ratta); 0000-0002-3184-5407 (E. Daga)

CEUR Workshop Proceedings (CEUR-WS.org)

based approach (either in querying the façade or transforming the data) and provide recommendations on how to improve the usability of SPARQL for KGC tasks.

The rest of the paper is structured as follows. We begin with an exposition of the related work, followed by a description of both the façade structure, Façade-X, at the heart of SPARQL Anything, and the XML and MusicXML format. Afterwards, we will continue by defining our tasks and we will show how we have implemented them. We then propose an evaluation of our results, comparing them to the results of selected tasks completed via the software Music21. After discussing what we have learnt from our work, we will proceed to draw our conclusions and propose further developments.

2. Related work

We consider related work on symbolic music encoding efforts, the application of semantic web technologies for treating music symbolic content, and façade-based querying of XML resources in SPARQL.

Symbolic music encoding Symbolic music formats varies according to the tasks and communities they refer to. Here, we limit to mention XML based representations of score sheets. An analysis of representation formats for symbolic music encodings can be found in [10]. MEI [11] is designed as a framework for encoding arbitrary musical documents, with particular focus to the needs of academia, for example, covering historical music notations. MusicXML is the industrial standard for encoding and exchanging musical sheets. Its most recent offical release (4.0) is curated by the W3C Music Notation Community Group. The community is currently working on a new format, MNX, which will expand MusicXML to fully support the Common Western Musical Notation (CWMN), with a particular focus on improving the semantics of the representation. The MusicXML format was designed to be a universal translator for programs that understand common Western musical notation [12]. Therefore, we make the assumption that any digital music sheet can be converted into MusicXML before applying our approach to KGC.

Music21 is a popular python library for symbolic music processing, supporting a large variety of formats. Its internal representation is based on nested containers (called Streams) modelling the hierarchical temporal structure of the score in measures, voices, and parts. We note how such internal representation nicely resembles the design principles behind Façade-X. In this paper, we use Music21 as a baseline method for verifying the correctness of our SPARQL-based approach. We rely on 'AltDeu10' book of the Essen Folksong Collection from the Music21 corpus, after converting it from its original ABC format to MusicXML.

Music and the Semantic Web A survey of musical data publishing practices on the (semantic) web can be found in [13]. Music has been a focus of the Semantic Web community from the very early stages. Music ontologies exist to support exchanging metadata [14], semantically enriching musical scores [15] and events [16], audio features [17], and enabling the interaction with MIDI content via RDF-based technologies [18]. Recent efforts include the Doremus [19] and Wasabi [20] knowledge graphs.

Crucially, the encoding of musical events in RDF raises important concerns in relation to how sequential information can be treated in RDF/SPARQL [21, 22]. Our work contributes to evaluate the impact of adding specialised functions for working with container membership properties, specifically.

Façade-based data access with SPARQL and XML In [8], a method based on the notion of *façade* is proposed to enable direct querying to an open-ended set of formats with plain SPARQL. Façade-X is based on a subset of RDF(S) and it is proposed as a generic meta-model through which reengineer potentially any format into RDF, including the popular CSV, JSON, and XML. The approach, implemented by the SPARQL Anything[1] software, was further validated theoretically in [9], demonstrating that it can indeed be applied to any format expressible into a BNF grammar. In a nutshell, Façade-X data objects are wrapped into a root *container*. Data is expressed using plain RDF properties for *key-value* structures, RDF container membership properties (CMP) for *sequences*, and RDF types for *unary predicates*. Values of RDF properties or CMPs can be either literals or other entities (containers), allowing composite nesting[2]. MusicXML is an application of the XML format. XML elements (also known as tags) are expressed in Façade-X as containers. Tag attributes are considered *key-value* pairs and represented as plain RDF properties and literal values. Container membership properties are instead used for specifying relations to child elements in the XML tree. These may include text, which can be expressed as RDF literals of type `xsd:string`. Finally, the tag name is represented as a unary attribute via the property `rdf:type`. Façade-X declares two namespaces:

```
fx: <http://sparql.xyz/facade-x/ns/>
xyz: <http://sparql.xyz/facade-x/data/>
```

If present, the transformation can apply namespaces declared within the XML document to name properties and types. A minimal example of XML and related Façade-X representation is provided in Figure 1.

[1] http://sparql-anything.cc

[2] We refer the reader to [8] and [9] for a thorough description of Façade-X and mappings to common file formats.

```
<items>                                  [ a xyz:items , fx:root ;
   <item letter="A" />                     rdf:_1 [ a xyz:item ; xyz:letter "A" ] ;
   <item letter="B" />                     rdf:_2 [ a xyz:item ; xyz:letter "B" ] ;
   <item letter="C">Text here</item>       rdf:_3 [ a xyz:item ;
</items>                                           rdf:_1 "Text here" ; xyz:letter "C" ] ] .
```

Figure 1: Minimal example of XML document and related Façade-X RDF interpretation.

3. Tasks definition

We now proceed to provide the reader with a more detailed description of the tasks that we wish to implement. We are assuming throughout that one is already familiar with the rudiments of western music notation and related concepts such as octave, pitch, duration, etc..

Melody extraction Melodies are a core component of many music computing tasks. We aim to extract information about the particular notes of a composition in an ordered fashion. By this we mean that we wish to extract the note data in the order that they have been encoded within their assigned part in MusicXML, for all parts in the encoded score or for a subset of those. By note information we mean their pitch attribute, made of the note's step, alteration and octave, and its duration, both in terms of its type (quarter, whole, etc.) and MusicXML numerator.

N-grams extraction An N-gram is a concept from computational linguistics that has been carried over to computational musicology [23]. It involves the construction of contiguous sequences of N information items from a given piece of information, usually a text or as in our case, an encoded score. For this task we will focus only on the extraction of 3-grams (trigrams) of pitches (the information) from a single or a collection of MusicXML files.

N-grams analysis The purpose of extracting N-grams from musical scores is to be able to conduct some empirical analyses on this information, so that one may be able to acquire further insight into a composition, composer, or style.

The fundamental statistical analysis of N-grams involves counting relative frequencies and estimating probabilities. This is what we aim to do.

Music Note ontology population Finally, we propose to extract information via a SPARQL query from a MusicXML file and to use that information to construct a representation of the note datum in accordance with a given ontology. For this task we have chosen to employ the Music Notation Ontology [24].

4. Tasks implementation

After having defined our tasks, we now show how we have implemented them via SPARQL Anything. Before we do that though, a few words should be spent describing the MusicXML format.

The MusicXML format represents a score in accordance with the hierarchical tree structure of a XML file. To do this, it begins with two alternative root elements, `<score-partwise>` or `<score-timewise>`, both of which can represent the score as a whole. Here we will only work with the first of these, as it is both the advised and most commonly used root element. 'Partwise' describes the fact that the score is divided into `<part>` elements each of which contains `<measure>` elements. The `<measure>` element is what includes the basic musical data such as the `<note>` elements, within which we will find all the other elements such as `<step>`, `<alteration>` etc. that we will be making use of in what follows below. For further information on MusicXML, we refer the interested reader to the format's current specification [25].

In the queries and results that follow, please assume the following list of namespaces:

```
1  PREFIX fx: <http://sparql.xyz/facade-x/ns/>
2  PREFIX xyz: <http://sparql.xyz/facade-x/data/>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4  PREFIX mno: <http://cedric.cnam.fr/isid/ontologies/
       MusicNote.owl#>
5  PREFIX alt: <http://polifonia.kmi.open.ac.uk/altdeu10/>
6  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
7  PREFIX ex: <http://example.org/>
```

The material developed for this research is also publicly available for reviewing and reproducibility [26].

4.1. Melody extraction

With the *getMelody.sparql* [3] query (Listing 1), we can extract from all melody parts in a MusicXML file with a `<score-partwise>` root element the ordered data that we need to construct the required note information, pitch and duration.

[3] Line 46 has been added for presenting the results in this paper does not generalise, as it relies on information that is particular to this document. This though does not affect the re-usable character of the query as it merely labels extracted data. This can be seen further below when this same query minus this line will be used for sub-querying.

Listing 1: SPARQL Query - *getMelody.sparql*

```
SELECT ?step ?alteration ?octave ?pitch ?type ?duration ?noteCount ?measureCount
      ?partCount ?voice
WHERE {
   SERVICE<x-sparql-anything:>{
      fx:properties fx:location "./musicXMLFiles/bwv153.1.musicxml"; fx:media-type "application/xml".
      ?score a fx:root .
      # Extracts the parts:
      ?score ?mProperty ?part .
      ?part a xyz:part .
      # Extracts the measures:
      ?part ?mPropPart ?measure.
      ?measure a xyz:measure .
      # Extracts the notes from each measure:
      ?measure ?mPropMeas ?note.
      ?note a xyz:note.
      {  # Extracts the pitch from the note element:
         ?note ?mPropNote ?pitch_.
         ?pitch_ a xyz:pitch.
         # Extracts the pitch's step:
         ?pitch_ ?mPropPitch1 ?stepElmnt.
         ?stepElmnt a xyz:step.
         ?stepElmnt rdf:_1 ?step.
         # Extracts the corresponding octave:
         ?pitch_ ?mPropPitch2 ?octaveElmnt.
         ?octaveElmnt a xyz:octave.
         ?octaveElmnt rdf:_1 ?octave.
         # Extracts the note's alterations, if present:
         OPTIONAL { ?pitch_ fx:anySlot [ a xyz:alter ; rdf:_1 ?alteration ]. }
         BIND( CONCAT( ?step, IF( BOUND( ?alteration ) && ?alteration != "0", CONCAT( "[",?alteration,"]" ), "" ), ?octave)
               AS ?pitch)
      }
      UNION
      { # Extracts the rest if present instead:
         ?note ?mPropNote ?restElmnt.
         ?restElmnt a xyz:rest.
         BIND(IF(BOUND(?restElmnt), "Rest", "") AS ?step)
         BIND(IF(BOUND(?restElmnt), "Rest", "") AS ?pitch)
      } # Extracts the corresponding type:
      OPTIONAL { ?note fx:anySlot [ a xyz:type ; rdf:_1 ?type ] } .
      # Extracts the corresponding duration:
      OPTIONAL { ?note fx:anySlot [ a xyz:duration ; rdf:_1 ?duration ] } .
   } # Generates measure and note position counters:
   BIND(fx:cardinal(?mPropPart) AS ?measureCount)
   BIND(fx:cardinal(?mPropMeas) AS ?noteCount)
   BIND(fx:cardinal(?mProperty) AS ?partCount)
   # Assigns the voice labels to the part numbers once these are known:
   BIND(IF(?partCount = 8, "Soprano", IF(?partCount = 9, "Alto", IF(?partCount = 10, "Tenor", "Bass")) ) AS ?voice)
}
ORDER BY ?partCount ?measureCount ?noteCount
```

We use Bach's chorale BWV 153.1 as an example, extracting the parts in the order that they have been encoded in. The following result, as a CSV file, is obtained, where we're only showing the first two measures of each voice:

```
step,alteration,octave,pitch,type,duration,noteCount,
      measureCount,partCount,voice
B,,4,B4,quarter,10080,3,1,8,Soprano
C,,5,C5,quarter,10080,1,2,8,Soprano
B,,4,B4,quarter,10080,2,2,8,Soprano
A,,4,A4,quarter,10080,3,2,8,Soprano
E,,5,E5,quarter,10080,4,2,8,Soprano
...
G,1,4,G[1]4,quarter,10080,3,1,9,Alto
A,,4,A4,quarter,10080,1,2,9,Alto
G,1,4,G[1]4,quarter,10080,2,2,9,Alto
A,,4,A4,quarter,10080,3,2,9,Alto
G,1,4,G[1]4,eighth,5040,4,2,9,Alto
A,,4,A4,eighth,5040,5,2,9,Alto
...
E,,4,E4,quarter,10080,3,1,10,Tenor
E,,4,E4,quarter,10080,1,2,10,Tenor
D,,4,D4,quarter,10080,2,2,10,Tenor
E,,4,E4,quarter,10080,3,2,10,Tenor
D,,4,D4,eighth,5040,4,2,10,Tenor
C,,4,C4,eighth,5040,5,2,10,Tenor
...
E,,3,E3,quarter,10080,3,1,11,Bass
A,,3,A3,quarter,10080,1,2,11,Bass
B,,3,B3,quarter,10080,2,2,11,Bass
C,,4,C4,quarter,10080,3,2,11,Bass
B,,3,B3,eighth,5040,4,2,11,Bass
A,,3,A3,eighth,5040,5,2,11,Bass
...
```

One can also make use of the 'id' attribute of the MusicXML's `<part>` (partwise) element to restrict extraction to only one of the parts.

To access the tenor part of the BWV 153.1 Chorale, for example, we write `?part xyz:id "P3".` after line 9 of the *getMelody.sparql* query to obtain the following:

```
step,alteration,octave,pitch,type,duration,noteCount,
      measureCount,partCount,voice
E,,4,E4,quarter,10080,3,1,10,Tenor
E,,4,E4,quarter,10080,1,2,10,Tenor
D,,4,D4,quarter,10080,2,2,10,Tenor
E,,4,E4,quarter,10080,3,2,10,Tenor
D,,4,D4,eighth,5040,4,2,10,Tenor
C,,4,C4,eighth,5040,5,2,10,Tenor
...
```

4.2. N-grams extraction

To extract any list of N-grams from a MusicXML file, we can use SPARQL Anything to construct a KG of those N-grams. Subsequently, through parametrisation, the same process can be extended to a given list of files.

Here we construct a KG of the 3-grams of the pitch information for each of the songs in the 'Altdeu 10' book.

The process is as follows. First we obtain a list of the URIs of the files to be queried as shown in Listing 2.

Listing 2: SPARQL Query - *getXMLPaths.sparql*

```
SELECT ?filePath ?fileName
WHERE {
 SERVICE<x-sparql-anything:>{
     fx:properties fx:location "./musicXMLFiles/AltDeu10".
     [] a fx:root;
       ?mProp ?filePath.
  FILTER(CONTAINS(?filePath, ".musicxml")) .
  BIND ( replace( replace ( ?filePath, ".*?\\/","" ), ".
       musicxml","") as ?fileName) .
 }
}
```

The result is outputted as the following SPARQL Result Set XML file:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
 <head>
  <variable name="filePath"/>
  <variable name="fileName"/>
 </head>
 <results>
  <result>
   <binding name="filePath">
    <literal>file:///C:/Users/Marco/Desktop/showcase-
          musicxml/./musicXMLFiles/AltDeu10/AltDeu10-256.
          musicxml</literal>
   </binding>
   <binding name="fileName">
    <literal>AltDeu10-256</literal>
   </binding>
  </result>
  ...
```

We require this to use a parametrisation in order to execute the same melody extraction query shown above on each file of the "AltDeu10" book. To do this, we use the above XML file, the BASIL variable '?_fileName' and modify the SERVICE clause of the *getMelody.sparql* query as in Listing 3.

Listing 3: SPARQL Query - *getMelodyParam.sparql*

```
SELECT ?step ?alteration ?octave ?pitch ?type ?duration ?
      noteCount ?measureCount ?partCount
WHERE {
   BIND (IRI(CONCAT("x-sparql-anything:media-type=
         application/xml,location=", ?_filePath)) AS ?fx)
   SERVICE ?fx {
     ?score a fx:root .
     # Continues with the rest of getMelody.sparql.
```

The result is a separate CSV file for each input file that follows the same pattern as our previous results in melody extraction above. That is, for example,

```
step,alteration,octave,pitch,type,duration,noteCount,
      measureCount,partCount
F,,4,F4,half,20160,2,1,6
F,,4,F4,half,20160,1,2,6
G,,4,G4,half,20160,2,2,6
A,,4,A4,half,20160,3,2,6
C,,5,C5,half,20160,4,2,6
B,-1,4,B[-1]4,quarter,10080,1,3,6
B,-1,4,B[-1]4,quarter,10080,2,3,6
A,,4,A4,half,20160,3,3,6
G,,4,G4,half,20160,4,3,6
...
```

and so on for each file in the given list.

Extracting the melodic information and representing it in a CSV file allows us now to use SPARQL Anything to build the required KGs of the 3-grams of pitches through a parametrised CONSTRUCT query. Indeed, as a condition to this second parametrised query, a second query on the file system has to be executed. As this is equal to the previous one but only for the value of the second parameter of the CONTAINS() function in the filter, which is "csv" in this case, we will omit it[4]. Thus, we have the query in Listing 4.

Listing 4: SPARQL Query - *trigKGBuildParam.sparql*

```
CONSTRUCT {
   ?entity a rdf:Seq;
     rdf:_1 ?note ;
     rdf:_2 ?note1 ;
     rdf:_3 ?note2 .
}
WHERE {
   BIND (IRI(CONCAT("x-sparql-anything:media-type=text/csv,
        location=", ?_filePath)) AS ?fx)
   SERVICE ?fx {
     fx:properties fx:csv.headers true.
   [] a fx:root;
     ?mProp [ xyz:pitch ?note ] ;
     ?mProp1 [ xyz:pitch ?note1 ] ;
     ?mProp2 [ xyz:pitch ?note2 ] ;
     FILTER ( fx:previous(?mProp1) =
       ?mProp && fx:previous(?mProp2) = ?mProp1)
   }
   BIND((fx:serial("trigram")) AS ?trigram) .
   BIND(fx:entity(ex:, ?trigram) AS ?entity) .
}
```

which constructs a KG of the 3-grams for each separate file that is given to it as an input.

The following example illustrates the content of each of these graphs.

Listing 5: SPARQL Result - *trigKG-AltDeu10-241.ttl*

```
ex:32 rdf:type rdf:Seq ;
      rdf:_1 "F4" ;
      rdf:_2 "F4" ;
      rdf:_3 "Rest" .

```

[4]The complete resources are available as supplemental material [26]

```
ex:13 rdf:type rdf:Seq ;
      rdf:_1 "G4" ;
      rdf:_2 "G4" ;
      rdf:_3 "G4" .

ex:19 rdf:type rdf:Seq ;
      rdf:_1 "Rest" ;
      rdf:_2 "G4" ;
      rdf:_3 "G4" .

ex:22 rdf:type rdf:Seq ;
      rdf:_1 "C5" ;
      rdf:_2 "F4" ;
      rdf:_3 "C5" .

ex:28 rdf:type rdf:Seq ;
      rdf:_1 "F4" ;
      rdf:_2 "Rest" ;
      rdf:_3 "G4" .
...
```

4.3. N-grams analysis

After constructing the KGs of 3-grams, one can begin to analyse them, still only using SPARQL Anything.

The query in Listing 6 combines the whole set of KGs of 3-grams of pitches produced above, counts their frequencies and estimates their probabilities.

Listing 6: SPARQL Query - *trigramAnalysis.sparql*

```
SELECT ?trigram ?frequency ( ( ?frequency / ?total ) AS ?probability)
WHERE {
   {# Subquery 1, extracts trigrams and their frequency.
    SELECT ?trigram ( COUNT( ?trigram ) AS ?frequency )
    WHERE {
       GRAPH ?g { ?s a rdf:Seq;
         ?mProp1 ?note1;
         ?mProp2 ?note2;
         ?mProp3 ?note3.}
       FILTER( fx:previous( ?mProp2 ) = ?mProp1 && fx:previous( ?mProp3 ) = ?mProp2 )
       BIND( CONCAT (?note1, "-", ?note2, "-", ?note3 ) AS ?trigram)
    }
       GROUP BY ?trigram
   }
   {#Subquery 2, counts the total number of resources.
    SELECT (COUNT(?r) AS ?total)
    WHERE { GRAPH ?g1 {?r a rdf:Seq.} }
   }
}
```

The results of this analysis are as in the following CSV example:

```
trigram,frequency,probability
D5-G5-F5,4,0.00024971906605069297040 8
G3-C4-C4,4,0.00024971906605069297040 8
A3-G3-D4,2,0.00012485953302534648520 4
Rest-Rest-C4,1,0.00006242976651267324260 2
E4-B4-C5,3,0.00018728929953801972780 6
C4-B3-B3,3,0.00018728929953801972780 6
F5-E5-Rest,2,0.00012485953302534648520 4
C5-E5-F5,3,0.00018728929953801972780 6
C4-D4-C4,20,0.00124859533025346485204 1
F[1]4-D4-E4,5,0.00031214883256336621301
A4-G4-E4,20,0.00124859533025346485204 1
A4-C5-F4,8,0.00049943813210138594081 7
E5-C[1]5-C[1]5,2,0.00012485953302534648520 4
...
```

4.4. Score ontology population

Using our above work in melody extraction and query parametrisation as a basis, we have implemented our final task by means of a SPARQL CONSTRUCT query (Listing 7, *populateOntology.sparql*), obtaining a collection of KGs describing each note of each song:

```
<http://polifonia.kmi.open.ac.uk/altdeu/altdeu-003/event/1>
       rdf:type    mno:Note ;
       mno:hasDuration "20160" ;
       mno:hasMeasure <http://polifonia.kmi.open.ac.uk/altdeu/altdeu-003/measure/1> ;
       mno:hasOctave "5" ;
       mno:hasPitch "D5" .

<http://polifonia.kmi.open.ac.uk/altdeu/altdeu-003/event/7>
       rdf:type    mno:Note ;
       mno:hasDuration "10080" ;
       mno:hasMeasure <http://polifonia.kmi.open.ac.uk/altdeu/altdeu-003/measure/3> ;
       mno:hasOctave "5" ;
       mno:hasPitch "D5" .

<http://polifonia.kmi.open.ac.uk/altdeu/altdeu-003/measure/5>
       rdf:type  mno:Measure ;
       mno:hasCount 5 ;
       mno:hasPart <http://polifonia.kmi.open.ac.uk/altdeu/altdeu-003/part/6> .
...
```

5. Evaluation and discussion

We now proceed to the evaluation of our implementations and to discussing some query design issues that were encountered and possible solutions.

5.1. Correctness of the data

To evaluate correctness, we have completed the melody extraction (with both a single and a collection of files), N-gram extraction and N-gram statistics tasks via Music21. The complete details of this can be found on the related GitHub page [26]. Here we summarise the results.

pitchCompare.py compares the 'pitches' columns of two CSV files in their order of appearance in each document. If the two lists are equal 'True' is returned. When *bwv153.1.csv* and *bwv153.1-Tenor.csv* are compared with their Music21 equivalents, we obtain the results as shown in Listings 8 and 9 (overleaf).

The *filesCompare.py* script uses the function `pitchCompare(file1, file2)` to compare the 'pitch' column of each CSV file that was produced by *getMelodyParam.sparql* to the same column of the CSV file produced with Music21 (see *getMelodyParam.py* in the GitHub repository). Executing the script prints the following to the terminal:

```
0 mismatches have been found between the files.
List of mismatches =
[]
```

Listing 7: SPARQL Query - *populateOntology.sparql*

```
CONSTRUCT {
  ?scoreEntity a mno:Score .
  ?scoreEntity mno:contains ?partEntity .
  ?partEntity a mno:Part .
  ?measureEntity a mno:Measure .
  ?measureEntity mno:hasPart ?partEntity .
  ?measureEntity mno:hasCount ?measureCount .
  ?measureEntity mno:hasUnit ?unit .
  ?eventEntity a ?eventType .
  ?eventEntity mno:hasMeasure ?measureEntity .
  ?eventEntity mno:hasPitch ?pitch .
  ?eventEntity mno:hasDuration ?duration .
  ?eventEntity mno:hasOctave ?octave .
}
#SELECT ?location ?fileName ?scoreEntity
WHERE {
   {{ # Melody Extraction Sub-query.
       SELECT ?step ?alteration ?octave ?pitch ?type ?duration ?noteCount ?measureCount ?partCount ?voice ?fileName ?unit
       #SELECT ?measure ?note ?step ?octave ?pitch
       WHERE {
          SERVICE <x-sparql-anything:> {
                fx:properties fx:location ?_filePath ; fx:media-type "application/xml".
                ?score a fx:root .
                # Part
                ?score ?mProperty ?part .
                ?part a xyz:part .
                # Extracts the measures:
                ?part ?mPropPart ?measure.
                #Extracts the notes from each measure:
                # Measure
                ?measure a xyz:measure .
                OPTIONAL {
                   # Measure attributes
                   ?measure fx:anySlot [
                       a xyz:attributes ;
                       fx:anySlot [
                          a xyz:time ;
                          fx:anySlot [ a xyz:beats ; rdf:_1 ?beats ] ;
                          fx:anySlot [ a xyz:beat-type ; rdf:_1 ?beatType ]
                       ]
                   ]
                   BIND ( CONCAT (?beats , "/", ?beatType) AS ?unit )
                  }
                  # Continues with the rest of getMelody.sparql.
           }
           ORDER BY ?partCount ?measureCount ?noteCount
   }}
      BIND ( fx:serial("event") AS ?eventOrder ) .
      # Mint entity IRIs
      BIND ( fx:entity(alt:, fx:String.toLowerCase( ?_fileName ) ) AS ?scoreEntity ) .
      BIND ( fx:entity(?scoreEntity, "/part/", ?partCount ) AS ?partEntity ) .
      BIND ( fx:entity(?scoreEntity, "/measure/", ?measureCount ) AS ?measureEntity ) .
      BIND ( fx:entity(?scoreEntity, "/event/", ?eventOrder ) AS ?eventEntity ) .
      BIND ( IF ( ?step = "Rest" , mno:RestEvent , mno:Note ) AS ?eventType ) .
      BIND ( xsd:int(?octave) AS ?octaveValue ) .
}
```

Listing 8: Comparing *bwv153.1.csv* and *bwv153.1Py.csv*

```
directory = 'C:/Users/Marco/Desktop/showcase-musicxml/'
file1 = directory + 'bwv153.1.csv' # SPARQL Anything
file2 = directory + 'pythonM21/bwv153.1Py.csv' # Music21
pitchCompare(file1, file2)
True
```

Listing 9: Comparing *bwv153.1-Tenor.csv* and *bwv153.1-TenorPy.csv*

```
directory = 'C:/Users/Marco/Desktop/showcase-musicxml/'
file1 = directory + 'bwv153.1-Tenor.csv' # SPARQL Anything
file2 = directory + 'pythonM21/bwv153.1-TenorPy.csv' #M21
pitchCompare(file1, file2)
True
```

Similarly, *trigAnalysisComparare.py* compares the two statistical analyses of the 3-grams extracted from this collection, *trigramAnalysis.csv* and *trigramAnalysisPy.csv*, in terms of differences in the constructed 3-grams, their individual frequencies and the total frequencies.

When executing the script we obtain the following.

As the 3-grams, their individual frequencies and total

Listing 10: Terminal output of *trigAnalysisCompare.py*

```
Different 3-Grams and frequency pairs found: 0
List of differences =
[]
Sum of frequencies:
SPARQL-Anything: 16018
Music21: 16018
```

Listing 11: `?measureCount` in SPARQL syntax

```
BIND( xsd:int(REPLACE( STR(?mPropPart), STR(rdf:_), "") )
      AS ?measureCount )
```

frequencies are the same, we deduce that the two analyses produce identical probability results, as a probability is a ratio of individual frequency to total frequency. The minor discrepancies that can be observed in the probability *estimates* given in the files are there due to rounding and/or truncation errors introduced by the respective systems and independent of the querying process.

5.2. Sequence functions help make queries more compact

Throughout the implementation of our tasks, we have had to overcome some specific technical problems related to how a SPARQL engine operates and the necessity of manipulating and presenting musical data in a way that is musically meaningful.

A musical composition is an ordered sequence of events. However, such information is not explicitly declared in the MusicXML format (for example, as an *index* XML attribute). Therefore, preserving the order of the XML elements is crucial in order to capture the notes' order.

Working with Façade-X helps providing solutions to this problem, as the container membership properties (rdf:_1, rdf:_2, etc.) are assigned to each XML element (in this use-case) according to their order of appearance, even when this order is not explicitly given in the original document. This gives us then the capacity of easily ordering the data through the use of a variable, bound to the integer of the membership property with which we can ORDER BY. Further, SPARQL Anything also offers specific functions which make such a task easier and more compact to program.

This can be seen in action in our *getMelody.sparql* query. With the original SPARQL syntax, constructing the variables required to order the data would look like the following.

The SPARQL Anything function `fx:cardinal(?a)`, which takes a container membership property as its input and returns its associated number cast as an integer, allows us to re-write this in the below simpler fashion.

Container membership properties have also enabled us

Listing 12: `?measureCount` in SPARQL Anything syntax

```
BIND( fx:cardinal(?mPropPart) AS ?measureCount )
```

Listing 13: *trigKGBuild.sparql*

```
[] a fx:root;
      ?mProp [ xyz:pitch ?note ] ;
      ?mProp1 [ xyz:pitch ?note1 ] ;
      ?mProp2 [ xyz:pitch ?note2 ] ;
      FILTER ( fx:previous(?mProp1) =
        ?mProp && fx:previous(?mProp2) = ?mProp1)
```

to easily construct a knowledge graph representation of all the possible trigrams of pitch information of a single composition.

The SPARQL Anything function `fx:previous(?a)` returns the container membership property immediately preceding `?a`, enabling us to filter through the n^N (n being the number of information, N the order of the N-gram) Cartesian products that the SPARQL engine generates when creating all the possible combinations of N information.

This can be seen in action in the following snippet of the *trigKGBuildParam.sparql* query as follows.

Façade-X is a nested RDF container data structure that correlates with a tree-like graph structure. As it converts every concrete data format to this tree-like structure, accessing the original data via a query can be viewed as a problem of successfully walking through this tree graph. While this process has its upside in the fact that it is essentially an iteration and variation of the same query pattern, it can also, because of this same reason, prove to be quiet lengthy and time consuming. The SPARQL Anything magic property `fx:anySlot` provides an initial solution to this problem.

In the *getMelody.sparql* query for example, the following code fragment is needed to extract the duration type (quarter, half, etc.) of a MusicXML note element:

```
?note ?mPropNote1 ?typeElmnt.
?typeElmnt a xyz:type.
?typeElmnt rdf:_1 ?type.
```

Using `fx:anySlot` this can be reduced to the following single line of code:

```
?note fx:anySlot [ a xyz:type ; rdf:_1 ?type ].
```

5.3. Discussion

Our goal was to explore the flexibility and robustness of the Façade-X model at the foundation of the SPARQL Anything technology, by querying encoded musical content in the form of MusicXML files and to proceed to see if there were the basis for employing the SPARQL Anything software to support tasks associated with computational musicology.

For this we have employed an exploratory-based approach, beginning with exploring the 'raw' data as it is presented by the corresponding façade and selecting the simplest features first. Then, incrementally, we have begun to extract more data and to use it to construct further information (e.g. pitches) and to re-use queries through sub-querying and/or parametrisation as a basis for implementing further more complex tasks.

Through our work, we have shown that it is possible to rely on Façade-X's representations of an XML and CSV document to extract the information from a MusicXML file, which was required for the completion of our proposed tasks. In fact, all of our queries have been constructed simply with having in mind the XML and CSV façades together with the MusicXML specification.

Indeed it is this foundation that enables the *potentially* 'universal' character of our querying process, as it relies on what the tree graph corresponding to the façade *must* have, while allowing us, via the OPTIONAL keyword, to include what that graph *may* have, with very few *ad hoc* goal based modifications.

Given this, some of the problems of having to work with a façade model follow. While at first this may prove to be a difficulty, as the façade for a specific data format is not actually 'in front us' if not for its general features, and because each data format (but also the same data format if it has slight variations, e.g. CSV files with or without headers) has a slightly different representation, the functionalities that SPARQL Anything offers help mitigating this process until the mental exercise of seeing the original file as a Façade-X graph has been mastered. SPARQL allows us to explore a dataset via queries, and because the façade is a tree structure, 'exploring the dataset' and 'walking through the graph' become equivalent operations. As such, SELECT queries that extract data can be built simply via this trial-and-error process of iterating the same basic 'atomic' query type that takes us from one node to its sub-nodes.

In terms of query design, this leads to two problems, the issue of having to work with container membership properties and the issue of possibly rather long and wordy queries emerging from graph walking. As discussed above, SPARQL Anything has tried to address both these issues through the implementation of specific functions such as `fx:cardinal(?a)` and the magic property `fx:anySlot`.

A second design issue can be identified when moving from a simple SELECT query to using the same as the basis of further SELECT or CONSTRUCT queries to achieve more complex tasks. For this two solutions have been implemented. We have either created 'pipelines' that via a process of extraction-conversion have taken us to where we wanted, or we have made use of the technique of sub-querying. In terms of development, this will likely be one of the issues to be addressed going forward, as the more complex the tasks that we will want to achieve become, the longer the number of intermediate steps are likely to become when processing the given encoded musical data. While sub-querying is likely to be a better usage pattern for this problem, the issue of query length and ease of design that may come with it will also need to be addressed. In this sense, new SPARQL Anything features such as query modularisation, for example, may prove to be fruitful to address query compactness.

6. Conclusions

In this paper we have explored the possibility of applying, in a preliminary and exploratory way, SPARQL Anything to musical content encoded in the MusicXML format and to execute a number of computational musicology related tasks through KG and KG related methods and possibilities.

As shown in the implementation and evaluation above, we have been able to construct a number of queries for the extraction of musical information that have shown themselves to be sufficiently general to aid with the construction of knowledge graphs (N-grams and ontology representation) related to musical content that may be of interest to computational musicologists. A further step in this direction would be to expand the breadth of our research by analysing a much greater database so as to also develop what we believe to be the potential data-format independence of our approach.

We have also provided further testing grounds for the SPARQL Anything technology and shown how a number of features that are specific to it, functions, magic properties, etc., aid query design and implementation when executing tasks that move away from the usual SPARQL use-cases.

If we can say that we have achieved positive results in terms of a proof of concept, we must also admit that we have just scratched the surface of what is required to successfully provide technical support to computational musicology and of the possibilities that the use of semantic technology can offer. Fundamental musical concepts such as that of the melodic or harmonic interval, for example, have not made an appearance here.

Despite this, what we have learnt from this research suggests to us that the first basic steps have been made in the right direction, and that it is possible to keep developing the SPARQL Anything technology to further support the construction of musical knowledge graph and to provide further technological support to computational musicologists for their undertakings.

Acknowledgments

This research was supported by the UK EPSRC intern-

ship Training Grant DTP 2020-2021 Open University and by the EU-funded project Polifonia: a digital harmoniser of musical cultural heritage (Grant Agreement N. 101004746), https://polifonia-project.eu.

References

[1] X. Zhu, Z. Li, X. Wang, X. Jiang, P. Sun, X. Wang, Y. Xiao, N. J. Yuan, Multi-modal knowledge graph construction and application: A survey, arXiv preprint arXiv:2202.05786 (2022).

[2] F. Simonetta, S. Ntalampiras, F. Avanzini, Multimodal music information processing and retrieval: Survey and future challenges, in: 2019 international workshop on multilayer music representation and processing (MMRP), IEEE, 2019, pp. 10–18.

[3] A. Scharnhorst, P. Van Kranenburg, F. Admiraal, P. Mulholland, C. Guillotel-Nothmann, The polifonia portal: a confluence of user stories, research pilots, data management and knowledge graph technology, in: DARIAH Annual event 2021, Interfaces, 2021.

[4] V. A. Carriero, F. Ciroku, J. de Berardinis, D. S. M. Pandiani, A. Meroño-Peñuela, A. Poltronieri, V. Presutti, Semantic integration of mir datasets with the polifonia ontology network, in: Proc. of the International Society for Music Information Retrieval Conference, ISMIR, 2021.

[5] M. Daquino, M. Wigham, E. Daga, L. Giagnolini, F. Tomasi, Clef. a linked open data native system for crowdsourcing, arXiv preprint arXiv:2206.08259 (2022).

[6] E. Daga, E. Motta, Capturing themed evidence, a hybrid approach, in: Proceedings of the 10th International Conference on Knowledge Capture, 2019, pp. 93–100.

[7] E. Daga, E. Motta, Challenging knowledge extraction to support the curation of documentary evidence in the humanities (2019).

[8] E. Daga, L. Asprino, P. Mulholland, A. Gangemi, Facade-x: an opinionated approach to sparql anything, Studies on the Semantic Web 53 (2021) 58–73.

[9] L. Asprino, E. Daga, P. Mulholland, A. Gangemi, Knowledge graph construction with a façade: a unified method to access heterogeneous data sources on the web, ACM Transactions on Internet Technology (TOIT) (2022) 58–65.

[10] A. Baratè, G. Haus, L. A. Ludovico, State of the art and perspectives in multi-layer formats for music representation, in: 2019 International Workshop on Multilayer Music Representation and Processing (MMRP), IEEE, 2019, pp. 27–34.

[11] A. Hankinson, P. Roland, I. Fujinaga, The music encoding initiative as a document-encoding framework., in: ISMIR, 2011, pp. 293–298.

[12] M. Good, G. Actor, Using musicxml for file interchange, in: Proceedings Third International Conference on WEB Delivering of Music, IEEE, 2003, p. 153.

[13] M. Daquino, E. Daga, M. d'Aquin, A. Gangemi, S. Holland, R. Laney, A. M. Penuela, P. Mulholland, Characterizing the landscape of musical data on the web: State of the art and challenges (2017).

[14] Y. Raimond, S. A. Abdallah, M. B. Sandler, F. Giasson, The music ontology., in: ISMIR, volume 2007, Citeseer, 2007, p. 8th.

[15] S. S.-s. Cherfi, C. Guillotel, F. Hamdi, P. Rigaux, N. Travers, Ontology-based annotation of music scores, in: Proceedings of the Knowledge Capture Conference, 2017, pp. 1–4.

[16] A. Poltronieri, A. Gangemi, The music note ontology, Workshop on Ontology Patterns (2021). URL: http://ceur-ws.org/Vol-3011/pattern2.pdf.

[17] A. Allik, G. Fazekas, M. B. Sandler, An ontology for audio features., in: ISMIR, 2016, pp. 73–79.

[18] A. Meroño-Peñuela, M. Daquino, E. Daga, A large-scale semantic library of midi linked data, in: 5th International Conference on Digital Libraries for Musicology (DLfM), Paris, France, 2018.

[19] M. Achichi, P. Lisena, K. Todorov, R. Troncy, J. Delahousse, Doremus: A graph of linked musical works, in: International Semantic Web Conference, Springer, 2018, pp. 3–19.

[20] M. Fell, E. Cabrio, M. Tikat, F. Michel, M. Buffa, F. Gandon, The wasabi song corpus and knowledge graph for music lyrics analysis, Language Resources and Evaluation (2022) 1–31.

[21] A. Meroño-Peñuela, E. Daga, List.MID: A MIDI-based benchmark for evaluating RDF lists, in: International Semantic Web Conference, Springer, 2019, pp. 246–260.

[22] E. Daga, A. Meroño-Peñuela, E. Motta, Sequential linked data: The state of affairs, Semantic Web (2021) 1–36.

[23] S. Downie, M. Nelson, Evaluation of a simple and effective music information retrieval method, in: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, 2000, pp. 73–80.

[24] F. Hamdi, S. S.-s. Cherfi, Music Notation Ontology, 2016. URL: http://cedric.cnam.fr/isid/ontologies/MusicNote.owl#, [Online; accessed 8-September-2022].

[25] MusicXML Specification Contributors, MusicXML, 2021. URL: https://www.w3.org/2021/06/musicxml40/, [Online; accessed 8-September-2022].

[26] M. Ratta, E. Daga, Sparql-anything/showcase-musicxml:, 2022. URL: https://doi.org/10.5281/zenodo.6966559. doi:10.5281/zenodo.6966559.