Mixing Music as Linked Data: SPARQL-based MIDI Mashups

Rick Meerwaldt[1], Albert Meroño-Peñuela[1], and Stefan Schlobach[1]

Department of Computer Science, Vrije Universiteit Amsterdam, NL
rickmeerwaldt@hotmail.com, {albert.merono,k.s.schlobach}@vu.nl

Abstract. A large number of datasets about music are available today in the Linked Open Data cloud, but most of them only describe music metadata. Datasets representing music notation (i.e. fine-grained musical transcriptions) are scarce, and hence musicians do not have the possibility to exploit Web technologies to their full potential. In particular, this situation hampers the musician's process of creating *mashups*, new musical compositions produced by remixing existing tracks. Recently, the MIDI Linked Data cloud has interlinked and published more than 300K MIDI songs as Linked Data. In this paper, we investigate the use of Semantic Web technology to produce musical mashups, and we present a framework to generate them systematically. We evaluate our approach with SPARQL-DJ, a prototype implementation that can be used to find, match, select and synchronize existing MIDI Linked Data, mix them, and create new musical content.

Keywords: MIDI, mashups, SPARQL, MIDI Linked Data

1 Introduction

Linked Data provides a way of 'publishing and connecting structured data on the Web' [5], effectively allowing to gain insight and value from data by enabling connections between data sets [8]. The four principles of Linked Data emphasize the use of the Resource Description Framework (RDF) to link data on the Web [3]. Following these principles, the Linked Open Data cloud has emerged on the Web as a vast amount of data of more than 100 billion RDF statements [2]. SPARQL, the RDF query language [11], is commonly used to retrieve and manipulate these data.

Music plays an important role in the Linked Open Data cloud, with large connected musical databases such as MusicBrainz, BBC Music and MySpace. However, most of these linked datasets only deal with music *metadata*: bands, musicians, song titles, albums, dates, firms, and so on. Recently though, there have been efforts to represent *music itself* as Linked Data. Music itself can be defined as music represented in a data structure in which information about the notes, rhythm, volume, instruments, etc. are stored. By representing music as Linked Data, it may be possible to gain a better understanding of music and

gain more insight in the semantics of music [9]. A possible way of representing music as Linked Data is by converting digital music in MIDI format to RDF, using the MIDI2RDF suite of converters [9]. MIDI (Musical Instrument Digital Interface) is a protocol for the interchange of musical information between musical instruments. It contains information about the music itself, and it can be compared to music scores. When MIDI files are converted to RDF they can be shared on the Web as Linked Data, and queried at a fine-grained level using SPARQL. The development of MIDI2RDF has lead to the conversion and linkage of large collections of MIDI music, giving birth to the MIDI Linked Data Cloud[1] [10]. The MIDI Linked Data Cloud connects 308,443 MIDI songs by representing their tracks, events, and notes as 10,215,557,355 RDF triples.

However, *musicians*, who are the key stakeholders in music as its genuine creators, can hardly take any advantage of this wealth of linked musical information. Concretely, many stages of the creative process that deal with remixing *existing* music to make new, genuine compositions could take advantage of this structured and interoperable representation. To narrow down the extensive possibilities within music creativity, we focus on *mashups*, a particular kind of music composition created by blending two or more pre-recorded songs. In this paper, we propose a framework and an implementation that supports the creation process of MIDI mashups within the Linked Data ecosystem, just by using RDF and SPARQL. We feed our pipeline with Linked Data coming from the MIDI Linked Data cloud, and we use these to create new mashups by combining existing songs, tracks and notes as Linked Data that can be played as MIDI. The lack of scientific research on the subject of composing digital music mashups is an added challenge in this process. To the best of our knowledge, no previous attempt has been made to compose digital music mashups by using only data and methods from the Semantic Web. Concretely, our contributions are:

- A framework for the creative process of composing digital music mashups, fitting both MIDI and Linked Data best practices (Section 3);
- A set of SPARQL query templates and queries to systematically find, filter, and select existing MIDI music represented as Linked Data, and blend them together as realistic mashups (Section 4);
- **SPARQL-DJ**, an implementation that uses these SPARQL queries to automate the process of creating MIDI mashups using MIDI Linked Data (Section 4.1);
- A discussion on key features that make these mashups work in terms of metric and harmony; and on future extensions (Sections 5 and 6).

2 Related Work

There have been various attempts in the domain of mashup automation. AutoMashUpper, for example, is described as an 'interactive system for creating music mashups by automatically selecting and mixing multiple songs together'

[1] `https://midi-ld.github.io`

[6]. AutoMashUpper determines the mashability of certain songs. Mashability is defined as how suitable multiple songs are to be a mashup together based on harmonic similarity. Mashh! is an online tool where people can find songs and loops and where they can mix their own mashups [16]. It is stated that mashups have gained popularity because music is ubiquitous on the Internet. It seeks for an answer to the question if a new form of music can be proposed.

A formal way of dealing with information about music on the Semantic Web is introduced by the Music Ontology [13]. Types of information described by this ontology are for example editorial, cultural and acoustic information of music. It is discussed that an improvement of the Music Ontology could be to create a score ontology. This score ontology could grant the possibility to deal with symbolic music notation or abstract composition rules. Furthermore, the timeline ontology represents a 'coherent backbone for addressing temporal information'. Whereas the chord ontology is proposed as a highly structured textual representation of chords [7]. It is stated, within the music information retrieval community, that much effort is spent on automatically describing the content of MIDI. Nevertheless, there is no standard methodology for chord annotation.

To be able to derive insight in music itself, music should be represented in a meaningful way. MusicNet is a large labeled dataset containing music [15]. It could be a source for research of machine learning methods for research in music. The prediction of nodes in recordings of music is described by means of a classification task on this dataset. By using these predictions, it is possible to transfer music to music scores.

MIDI is used as standard for the musical notation in this paper [14]. In the basis, a song in MIDI is called a pattern. A pattern consists of one or more tracks. A track can be most easily compared to a sheet of nodes that can be played by an instrument. These nodes are called events. However, these events describe more than only what nodes are played. A various number of different events exist to specify when a node is being played, when it is stopped being played, what the tempo of an event is and when the track has reached its end.

Besides MIDI there are various different solutions to musical notations. Some examples of this are MusicXML [1], the Notation Interchange File Format[2] (NIFF) and the Music Encoding Initiative[3] (MEI). MusicXML and NIFF represent musical notations, whereas MEI is a formal system for the encoding of musical documents.

There has been an ongoing effort in linking databases containing audio, metadata and musical notation in the form of music scores or MIDI. The linking of those databases could lead to a better music integration and retrieval. The Lakh MIDI dataset [12] is a dataset containing 176,581 unique MIDI files. 45,129 of these MIDI files have been matched to entries in the Million Song Dataset [4]. The Million Song Dataset is a dataset that contains metadata and audio features for a million songs. The goal of the Lakh MIDI dataset is to facilitate large-scale music information retrieval.

[2] http://www.music-notation.info/en/formats/NIFF.html

[3] http://music-encoding.org/

The possibilities of representing MIDI as Linked Data have been recently investigated [9]. Consequently, the MIDI2RDF suite of tools[4] has been developed, which allows for the lossless conversion of digital music in MIDI format to RDF and back. Also, the MIDI Linked Data cloud is being introduced [10]. More than 300,000 interconnected MIDI files in the form of Linked Data are included in this dataset. There is also an introduction of some applications in the domain of musicology and music information retrieval. An interesting example of this is the Linked-Data DJ. A potential usage of the MIDI Linked Data Cloud could be to use queries as a means for mixing music, providing possibly for a new approach of mixing and composing music. It is also stated that SPARQL can be used to filter on musical properties, with examples as keys, harmony and tempo.

3 A Framework for Making MIDI Linked Data Mashups

3.1 The Mashup Process

In the standard definition of a mashup, the requirements are limited. On MusicBrainz[5] a mashup is defined as a mix in which two or more songs are playing simultaneously. According to Wikipedia[6] a mashup is a song created out of pieces of two or more songs. However, just adding two songs to each other will generally not result in a mashup that sounds good. To create a genuine mashup, we identify the steps shown in Table 1 after consulting a number of sources[7]. According to Spin Academy, an online DJ school, the key skills to be mastered in order to make a mashup are *beatmatching* and *mixing in key* (harmonic mixing). Beatmatching is a technique for matching the beats per minute (BPM) of different tracks to synchronize them, by adjusting the tempos of the individual tracks. Harmonic mixing is the process of getting two tracks in the same or relative keys, so there are no dissonant tones between the tracks when they are mixed.

	Process	In scope
1	Make an intro	No
2	Import songs	Yes
3	Find the BPM and synchronize the songs (*beatmatching*)	Yes
4	Get the songs into key (*harmonic mixing*)	No
5	Mix instrumentals	Yes
6	Mix vocals	No
7	Add effects	No
8	Make an outro	No

Table 1. The identified steps in the mashup generation process

[4] https://github.com/midi-ld/midi2rdf/

[5] https://wiki.musicbrainz.org/Terminology

[6] https://en.wikipedia.org/wiki/Mashup_(music)

[7] See http://spin-academy.com/how-to-make-a-mashup/, https://www.youtube.com/watch?v=nbRauP-xyLM&t=2s, and https://www.reddit.com/r/mashups

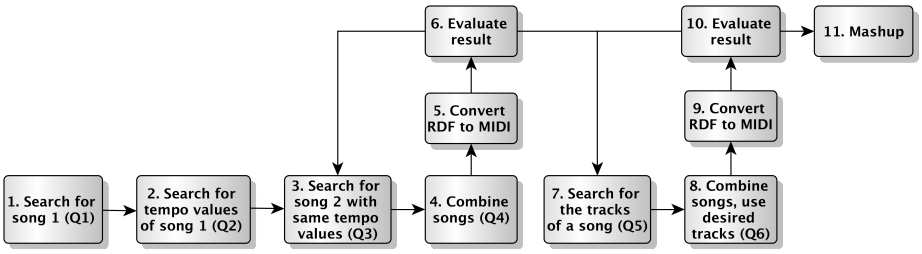

Fig. 1. The mashup creation framework

We focus on making mashups with a synchronized beat (step 3 of Table 1). This is done by using SPARQL queries on MIDI Linked Data. Before this can be done, we need the following queries: one for constructing a graph that can be converted to a MIDI file; on for adding multiple songs together (step 2 of Table 1, importing songs); some for filtering certain tracks of songs (mixing of instrumentals, step 5 of Table 1). Beat synchronization in this research is done by querying for two songs that have the same tempo. All these are covered in the remaining of Section 3, and in Section 4. We leave steps 1, 6, 7 and 8 of Table 1 to future work, since they are not considered central to distinguish a mashup from something that is not.

3.2 The Framework

After conducting research on how to make a mashup using SPARQL queries, a framework was constructed. The framework can be used as guidance in the generation of a mashup. It comes in the form of a diagram which shows a step-by-step use case to generate a mashup (see Figure 1). The framework refers to several steps where queries need to be used (Q1-Q6). For these steps the SPARQL endpoint for the MIDI Linked Data cloud[8] need to be used.

The queries that are referred to are used to get values for certain variables that are necessary to make a beat-synchronous mashup. Values have to be found for patterns, tempo values and the desired tracks of multiple songs. All the required queries can be found on the SPARQLmashup GitHub page[9]. This GitHub page contains everything there is to the framework. It contains the framework itself, as well as the corresponding queries. Furthermore, it contains a detailed explanation of how the queries can be used in the form of an example of the creation of a MIDI Linked Data mashup. Some mashups that were created in the process can also be found on the GitHub page.

In step 1 of the framework the first pattern is found. After finding the first pattern, the user searches for its tempo values (step 2). Using the tempo values of the first pattern, the user can search for the second pattern (step 3). After

[8] `http://virtuoso-midi.amp.ops.labs.vu.nl/sparql`

[9] `https://github.com/rickmeerwaldt/SPARQLmashup`

two patterns are found, they are combined in step 4. The result of the query in step 4 can be converted to MIDI, using the MIDI2RDF suite of tools (step 5). The result is evaluated in step 6.

Even though it is certain that the two songs are beat-synchronous, it is still possible that the two songs are not in harmony with each other or that the songs just don't match. Consequently, it is possible to search for a different second pattern (step 3). It is also possible to search for the tracks of the songs (step 7) and combine the songs using only certain tracks (step 8). Filtering out certain tracks that cause the dissonant tones can circumvent the problem of the songs not being harmonious. Filtering tracks can also cause for a better mix of instrumentals.

After combining the songs using only the desired tracks, the result of the query in step 8 can be converted to MIDI (step 9). The result is evaluated in step 10. If the songs (still) don't match, it is possible to filter even more tracks in or out or to choose for a different second track. However, if the result is the desired result, a beat-synchronous mashup has been created, using Semantic Web technologies (step 11).

The combination of two patterns in a convertible query is analogous to the importing of songs (step 2 of Table 1), since the combination causes that the songs can be converted and played. The search for the second song, using the tempo values of the first song, is analogous to the synchronization of two songs (step 3 of Table 1), since the second song is filtered on having the same tempo values as the first song. Due to this, the songs will be beat-synchronous with each other. The filtering of tracks can be compared to the mixing of instrumentals (step 5 of Table 1), since it allows for different combinations of instruments in the mashup. This can result in a better mix of instrumentals.

4 Implementation

As discussed before, an RDF file should have a certain structure so that it can be converted to MIDI. Therefore, we propose a general query template. This template can be used on the SPARQL endpoint to generate a result that can be converted to a working MIDI file. We achieved this template by inspecting RDF conversions of multiple MIDI files, and studying their structure. This results in the general SPARQL query template for the generation of a MIDI file shown in Listing 1.1, which we use as basis for the generation of correct MIDI files. We use the MIDI2RDF suite of tools [9], and the `rdf2midi` script, to convert the RDF generated by this query template to a MIDI file; by playing this MIDI file we can evaluate the results.

To make a query for the generation of a mashup, we extend the query in Listing 1.1. Additionally, several queries are necessary to get values of certain variables of this query. The rationale behind the queries of the framework will be discussed in this section. The values that need to be found for the mashup generation query are the values for the following variables:

- The pattern of the first song

```
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX mid: <http://purl.org/midi-ld/midi#>

CONSTRUCT { <pattern1> a mid:Pattern ;
mid:hasTrack ?track .
<pattern1> mid:format ?format .
<pattern1> mid:resolution ?resolution .
?track mid:hasEvent ?event .
?track a mid:Track .
?event a ?type .
?event ?property ?value .
}

WHERE {
<pattern1> mid:hasTrack ?track .
<pattern1> mid:format ?format .
<pattern1> mid:resolution ?resolution .
?track mid:hasEvent ?event .
?event a ?type .
?event ?property ?value .
}
```

Listing 1.1. The structure of a SPARQL query for the generation of a MIDI file

- The pattern of the second song
- The tracks of the first pattern
- The tracks of the second pattern

The first pattern can be found by filtering on the name of the song or the artist. As discussed before, the synchronization of two songs is done by querying on two patterns that have the same beat. When the first pattern has been found, the search for the second pattern can start. The tempo values of the first pattern can be found by using a certain query. After this, the second pattern can be found by filtering on the tempo values of the first pattern. Additionally, one can filter on the name of the song or the artist, to get a more specific result. The tempo values that are being filtered on are *resolution*, *numerator*, *denominator*, *metronome*, and *thirtyseconds*.

Timing in MIDI files is centered around ticks and beats. A beat is the same as a quarter note. Beats are divided into ticks, the smallest unit of time in MIDI. Each message in a MIDI file has a delta time, which tells how many ticks have passed since the last message. The length of a tick is defined in *resolution* and typically ranges from 96 to 480. The *numerator* and *denominator* are two numbers specified in the time signature, a notational convention used in Western musical notation to specify how many beats (pulses) are to be contained in each bar and which note value is to be given one beat. For example, in the time signature 3/4, the numerator is 3, the denominator is 4, and these mean that in each measure (bar) of the notation there are three (numerator) quarter-notes or crotchets (denominator). The value for *metronome* is the number of MIDI clocks per metronome tick. *Thirtyseconds* specifies the number of 1/32 notes per 24 MIDI clocks (8 is standard).

These values together comprise the tempo of a pattern in MIDI. If these values are the same for both songs, they will automatically be beat-synchronous. It can also be discussed that the beats per minute (BPM) and the microseconds

per quaternote (MPQN) should be set the same. However, when a MIDI file is played it automatically has a single BPM and MPQN. Therefore, synchronizing the BPM and MPQN does not have to be addressed.

After two patterns with the same beat are found, it is already possible to make a mashup by using all the tracks of the two patterns. This can be a good point to do an evaluation of the mashup so far. A problem that can arise is that there are a lot of dissonant tones between the two songs. If this is the case, there are two options. It is either possible to not use all tracks of both songs and to filter only on certain tracks. However, if this does not work it is still possible to choose for a different second pattern. Experience after making several mashups has proven that if two songs sound relatively harmonious together, a track filtering still can be used in order to let the mashup sound not too noisy or busy. Using a simple query, the tracks of each pattern can be found. Consequently, with an extra filter on both patterns in the mashup generation query, several tracks can be left out.

4.1 SPARQL-DJ

In order to evaluate our framework, and save users the tedious process of writing SPARQL queries, manipulating variable values, and manually executing the MIDI2RDF programs, we implement the whole process in the SPARQL-DJ[10]. SPARQL-DJ is a web-based user-interface based on the presented framework. In this prototype, users can easily search for songs and make mashups from them. Users don't see the queries at all, and only need to fill in some forms. The resulting MIDI mashups can be directly played in the browser, or be downloaded as either MIDI or Turtle. Figure 2 shows the abstract interaction with the interface elements, and Figure 3 shows an actual screenshot of SPARQL-DJ.

First, the user specifies the 'song title / author', and he can press a submit button to receive a list of songs matching the input. The user can look at the title or listen to the songs before deciding which one to choose. After he knows what song he wants, he presses select. A box pops up, showing the tempo values of this song. Also, the box to search for the second song pops up. In this box the user can again specify the 'song title / author'. Then, he can press whether he wants the second song to have the same tempo values as the first song. If he does not want the same tempo values for the second song, he presses 'no'. After this, a box pops up in which he can specify the tempo values himself. If the user wants the same tempo values for the second song, he presses 'yes'. After pressing the submit button or the 'yes' button a list of songs is received, matching the tempo values. When the both songs are selected, a box for the mashup appears. In this box the user can select or deselect certain tracks. The selection can be based on the names of instruments of certain tracks, or the user can listen to certain tracks. The user can also change the resolution of the song, using the resolution slide bar. A higher resolution will result in a faster mashup. The big 'PLAY THE MASHUP!' button can be pressed in order to play the mashup.

[10] `http://sparql-dj.amp.ops.labs.vu.nl`

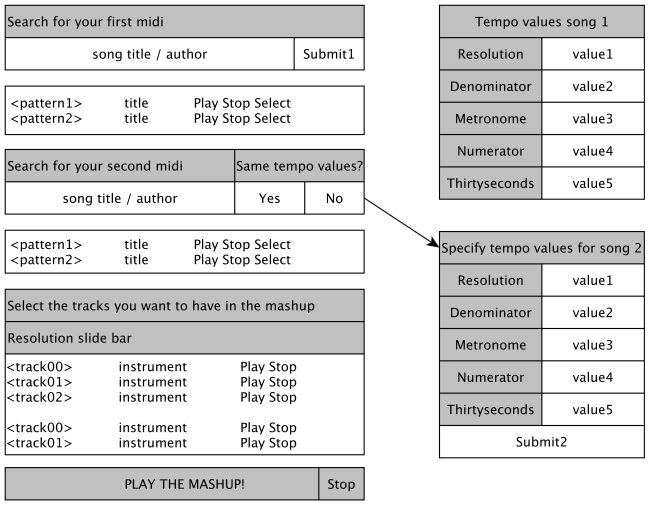

Fig. 2. Interaction model followed by the interface of the SPARQL-DJ

When the first song is found, the user can search for the second song by just searching for a title or artist. If this is done, the songs that are shown are automatically time synchronous with the first song, so the user does not have to look at the tempo variables of the songs. It is also possible to uncheck the 'same tempo values' box. Consequently, a set of new boxes appear in which the user can specify the tempo values (see Fig. 4). After selecting the second song, the tracks of both songs are displayed. Also, a check box per track is shown so the user can specify which tracks he wants in the mashup.

Note that the first track (track00) of a certain pattern is a track containing tempo values and tempo changes. Sometimes a certain pattern is selected that contains numerous tempo changes, for example a song that is twice as fast after a while. If this is the case, the first track can be deleted from a that pattern in the mashup in order to ignore those tempo changes. However, the first track of one of the songs should be preserved, since otherwise the tempo of the mashup is lost and the mashup will be of an undefined tempo.

After the desired tracks are selected, the user can play the MIDI Linked Data mashup directly in the browser. The user can also choose to alter the selected tracks even more or to download the mashup as a MIDI or Turtle file.

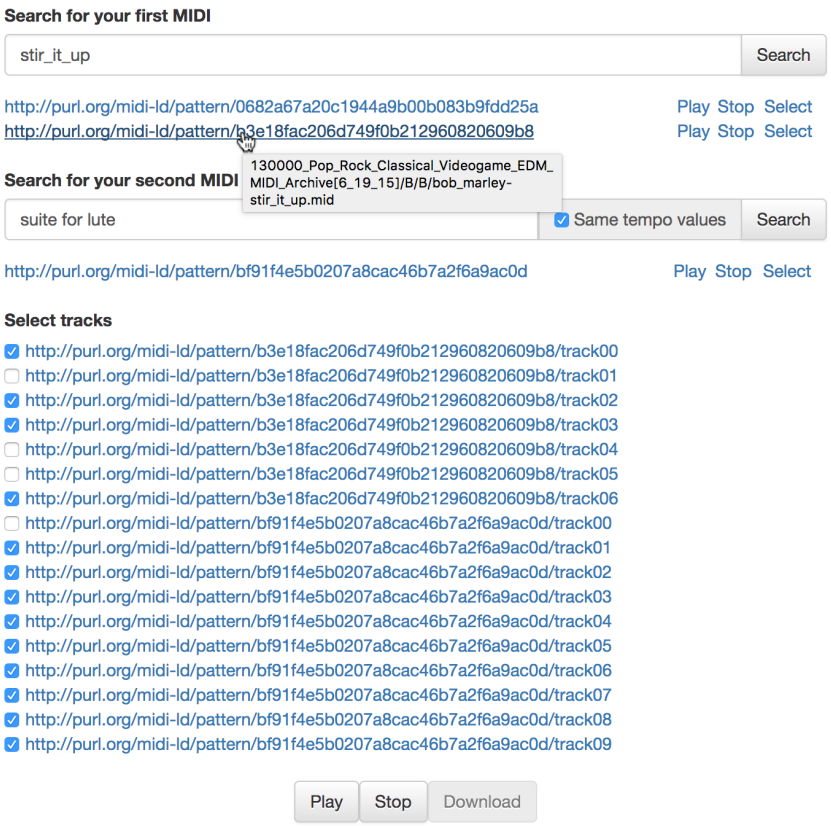

Fig. 3. A screenshot of the SPARQL-DJ tool

5 Discussion

The framework presented in this paper can be used for the automation of the mashup process. SPARQL-DJ is an example of this. It automatically performs the process of beatmatching by searching for songs with the same beat. However, the framework and the prototype do not provide for an automation of harmonic mixing. This could be addressed in the future. The framework and prototype are also limited in the number of mashups that can be made, since they do not allow for the manipulation of the tempo values to make two songs beat-synchronous. We found that the manipulation of these tempo values caused that the estimated execution time was too high, due to the optimization settings of the SPARQL endpoint. Hence, some of these queries are yet too expensive to be performed. We plan on letting users know when an operation will be costly, by adding visual aids in the interface.

It can be discussed that the mashups we produce are not genuine mashups, since not all requirements from Table 1 are met. However, these requirements are

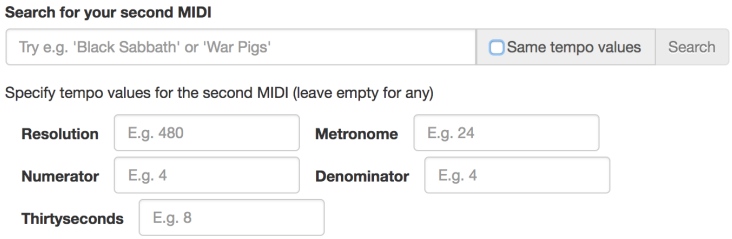

Fig. 4. A screenshot of the option to specify different tempo values

not strict requirements. In this paper we introduce the possibility to generate beat-synchronous mashups using the Semantic Web technology stack, and we have every reason to be optimistic about covering the remaining requirements.

6 Conclusion

This paper addresses limitations of musicians at using Semantic Web technologies in the mashup creation process. To address this, we propose a framework that enables the generation of mashups from existing MIDI data represented as Linked Data by using RDF and SPARQL. To prove its adequacy, we propose a set of SPARQL queries that match existing MIDI events following certain criteria and mixes them up as a mashup. Finally, we present the SPARQL-DJ, which implements our previous contributions and facilitates finding, selecting, playing, mixing, and generating beat-synchronous mashups by strictly using SPARQL over existing MIDI Linked Data. This covers the core requirements of the overall mashup creation process.

We plan on extending this work in several ways. First, we will propose ways to automatically manipulate the tempo values of two songs that are not beat-synchronous in order to synchronize them; a widely-used process in the domain of digital audio workstations. Secondly, we will extend our framework and implementation to also cover harmonic matching in MIDI, since at the moment only beat matching is performed and, consequently, dissonant mashups might occur. This can be done analogously to beat-matching: by either restricting search results to MIDIs that have the same key; or by allowing to manipulate and transpose MIDIs that have different keys in order to make them harmonically compatible. Third, we will improve the quality of mashups by adding an intro, and outro, and the remaining phases of mashup creation that are out of the scope of this paper. Fourth, we will research ways of leveraging links of the MIDI Linked Data Cloud to other musical databases (DBpedia, MusicBrainz, AccousticBrainz, etc.) to improve the mashup creation process; for example, by mixing MIDIs of the same/different genre. Finally, we will investigate ways of improving the performance of SPARQL at mixing MIDI Linked Data.

References

1. MusicXML 3.0 Specification. Tech. rep., MakeMusic, Inc. (2015), http://www.musicxml.com/
2. Abele, A., McCrae, J.P., Buitelaar, P., Jentzsch, A., Cyganiak, R.: Linking Open Data cloud diagram. http://lod-cloud.net/ (2017)
3. Berners-Lee, T.: Linked Data - Design Issues. https://www.w3.org/DesignIssues/LinkedData.html (2006)
4. Bertin-Mahieux, T., Ellis, D.P., Whitman, B., Lamere, P.: The Million Song Dataset. In: Proceedings of the 11th International Conference on Music Information Retrieval (ISMIR) (2011)
5. Bizer, C., Heat, T., Berners-Lee, T.: Linked data-the story so far. In: Semantic services, interoperability and web applications: emerging concepts, pp. 205–227 (2009)
6. Davies, M.E.P., Hamel, P., Yoshii, K., Goto, M.: AutoMashUpper: An Automatic Multi-Song Mashup System. In: International Conference on Music Information Retrieval (ISMIR). pp. 575–580 (2013)
7. Harte, C., Sandler, M., Abdallah, S., Gómez, E.: Symbolic representation of musical chords: A proposed syntax for text annotations. In: Proceedings of the International Conference on Music Information Retrieval (ISMIR). pp. 66–71 (2005)
8. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. In: Synthesis lectures on the semantic web: theory and technology, pp. 1–136. Morgan and Claypool, l edn. (2011)
9. Meroño-Peñuela, A., Hoekstra, R.: The Song Remains the Same: Lossless Conversion and Streaming of MIDI to RDF and Back. In: 13th Extended Semantic Web Conference (2016)
10. Meroño-Peñuela, A., Hoekstra, R., Gangemi, A., Bloem, P., de Valk, R., Stringer, B., Janssen, B., de Boer, V., Allik, A., Schlobach, S., Page, K.: The MIDI Linked Data Cloud. In: 16th International Semantic Web Conference (ISWC 2017) (2017)
11. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF - W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query/ (2008)
12. Raffel, C.: Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching. Columbia University (2016)
13. Raimond, Y., Abdallah, S., Sandler, M., Giasson, F.: The Music Ontology. In: International Conference on Music Information Retrieval (ISMIR). vol. 422 (2007)
14. The MIDI Manufacturers Association: The Complete MIDI 1.0 Detailed Specification. Tech. rep., The MIDI Manufacturers Association, Los Angeles, CA (1996-2014), https://www.midi.org/specifications/item/the-midi-1-0-specification
15. Thickstun, J., Harchaoui, Z., Kakade, S.M.: Learning Features of Music from Scratch. In: International Conference on Learning Representations (ICLR) (2017)
16. Tokui, N.: Massh!: a web-based collective music mashup system. In: Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts. pp. 526–527. ACM (2008)