IS THERE A DATA MODEL IN MUSIC NOTATION?

Raphal Fournier-S'niehotta
CNAM
`fournier@cnam.fr`

Philippe Rigaux
CNAM
`philippe.rigaux@cnam.fr`

Nicolas Travers
CNAM
`nicolas.travers@cnam.fr`

ABSTRACT

Scores are structured objects, and we can therefore envisage operations that change the structure of a score, combine several scores, and produce new score instances from some pre-existing material. Current score encodings, however, are designed for rendering and exchange purposes, and cannot directly be exploited as instances of a clear data model supporting algebraic manipulations. We propose an approach that leverages a music content model hidden in score notation, and define a set of composable operations to derive new "scores" from a corpus of existing ones. We show that this approach supplies a high-level tool to express common, useful applications, and can easily be implemented on top of standard components.

1. MOTIVATION

The digital encoding of music notation is a long standing endeavour, and has given rise to many proposals [1]. Nowadays, leading encodings are those which rely on the XML format to represent music notation as structured documents. MusicXML [2] is probably the most widespread one, due to its acceptance by major engraver software applications (Finale, Sibelius, and MuseScore) as an exchange format. This interoperability motivation yields an encoding which simultaneaously conveys structural, content and rendering information in a somewhat intricate representation. Another issue is the dependency of notation syntax and its interpretation on locations, periods, styles, and cultural contexts. Designing a format apt at capturing this high variability in a single and consistent representation is quite challenging. The MEI initiative [3, 4] attempts to address this challenge with an extensible format [5]. It relies on pre-defined components such as, for instance, the Common Music Notation (CMN) module. The initial discusions held in the recent W3C Music Notation Community Group [6] , launched in Sept. 2015, also point to the difficulty of a general, consistent, encoding framework that would capture the syntactic and semantic nuances of music notation throughout the specialized context of its use.

If we consider a specific notational context, and assume the existence of a specialized format that accurately covers the musical idiosyncrasies of this context (for instance an MEI module as stated above), then it makes sense to assume that this format encapsulates a data model for content encoding, at least for this particular part of the music repertoire. Focusing on Common Music Notation for instance, this data model can partly be identified through the commonalities of distinct formats such as, say, MusicXML, MEI and Lilypond. Beyond their different initial motivations and approaches, they share a basic set of features that characterize the music material to be represented.

Obviously, this notion of "content model" is controversial in the context of music notation: most of the elements that describe a score rendering can, to some point, be considered as significant and part of the global meaning conveyed by the notation. Although the separation of content and rendering components is a recurring topic of discussion for the designers (see for instance [7]), no score encoding, to our knowledge, has yet been designed with such a motivation in mind. One of the greatest benefits would indeed be the ability to "style" score contents, akin to what has been achieved for the rendering of HTML/XML content in the area of Web documents.

In the present paper, we address another motivation for separating content/rendering concerns. Being able to identify a content model opens the way to the vision of score corpora as a collection of structured objects, and makes it possible to envisage operations that "plays" with the structured content in order to extract useful parts, combine several scores, and derive new content from existing ones.

There are strong motivations for enabling such a system. It would indeed provide, via a high level language, a number of useful operations.

- *Automatic content management.* Split a score in parts, distribute them to digital music stands, apply transpositions and add decorations (directives) as needed; conversely, merge distinct parts as a single score;
- *Search and compare.* Search scores which satisfy some criteria; extract the matching fragments; align those fragments in a new score for further investigation;
- *Advanced analytic.* Derive analytic features (e.g., harmonic progression); annotate scores with these features; produce new representations emphasizing structural or compositional aspects.

Our perspective is to equip a Digital Score Library (DSL) with such an algebraic language, in order to derive "intentional" scores from "extensional" ones (the Library), in a direct correspondence with relational databases and the relational algebra [8]. We focus on operations that apply to the model space *in closed form*, i.e., which map instance(s) of the model to other instance(s). With closure comes composition: if s is a model instance (a "score") and o_1, o_2 are two operations, then $o_1(s)$ is a "score", $o_2(o_1(s))$ is also a "score", and we obtain an algebraic structure (in the mathematical sense) that lets us manipulate score material in order to produce new representations.

This approach brings, to the design and implementation of applications that deal with symbolic music, the standard and well-known advantages of specialized data management systems. Let us just mention the few most important: (i) ability to rely on a stable, well-defined and expressive data model, (ii) independence between logical modeling and physical design, saving the need to confront programmers with intricate optimization issues at the application level and (iii) efficiency of set-based operators and indexes provided by the data system.

In summary, we expose in the rest of the paper the following contributions

1. *A vision*. We describe in Section 2 a conceptual setting where digital libraries of score encodings can be leveraged to support structured content manipulation.

2. *A model*. Section 3 proposes a model that captures the most common features of CMN, along with a high-level, algebraic query language.

3. *Implementation guidelines*. Finally, we provide in Section 4 a technical discussion, based on our current implementation choices, showing the limited efforts required to achieve our vision.

Section 5 briefly discusses related work and Section 6 concludes the paper and discusses ongoing and future work.

2. VISION

Figure 1 summarizes the envisioned system. We propose in Section 4 a technical discussion related to our current implementation choices, but the figure exposes the main conceptual features at a convenient level of abstraction.

The bottom layer is a Digital Score Library (DSL) managing corpora of scores in some encoding, whether MusicXML, MEI, or any other legacy format (e.g., Humdrum). As explained in the introductory part, such encodings are not designed to support content-based manipulations, and, as a matter of fact, it is hardly possible to do so. Access to *explicit* music content information is intricate, due to the complex interleaving of content-oriented and rendering-oriented elements. Extracting a mere sequence of notes from MusicXML or MEI for instance is not a trivial task. Using implicit features that could be derived from the encoded content is even more difficult.

We therefore *map* the encoding toward a model layer where the content is extracted from the encoding and structured according to the model structures. One mapper has to be

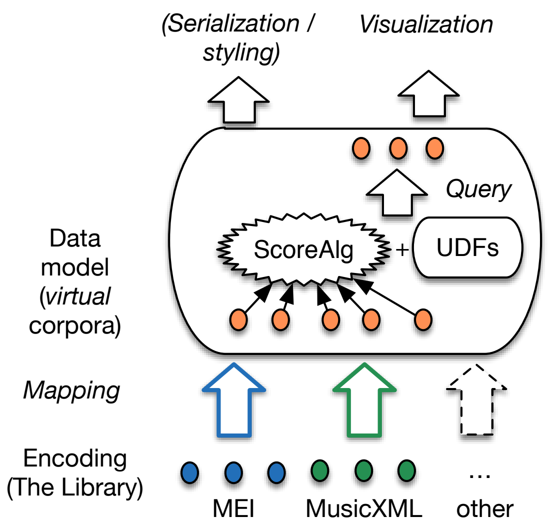

Figure 1. Envisioned system

defined for each possible encoding, as shown by the figure which assumes that MusicXML and MEI documents cohabit in a same DSL. Adding a new source represented with a new encoding is just a matter of adding a new mapper. Each document in the DSL is then mapped to a (virtual) instance of the model. These define virtual – no materialization occurs – corpora of music notation objects that we will call *vScores* in the following.

The data model layer encapsulates both data representation and data operations. We further distinguish two kinds of operations: *structural operators* and *user-defined functions* (UDFs). The former implement the idea that structured score management corresponds, at the core level, to a limited set of fundamental operations, grouped in a score algebra, that can be defined and implemented once for all. The latter acknowledges that the richness of music notation manipulations calls for a combination of these operations with user-defined functions at early steps of the query evaluation process. Modeling the structural operators and combining them with user-defined operations constitutes the operational part of the model. This yields a query language whose expressions define the set of transformations that produce new *vScores* from the base corpora.

The result of a query is itself a new, intentional corpora. This gives rise to several potential exploitations. First, the result can be kept in the user space, as a "view" (using database terminology) over the base corpora. A performer could for instance keep a set of parts for her next rehearsal/concert. Second, the query result can be visualized, possibly with representations that emphasize analytical aspects computed from the scores.

Finally, derived *vScores* can be in serialized in a permanent storage, in a format compliant to one of the encoding standards. This is where styling could take place, in a process which is the exact opposite of the mapping which abstracts a content from MusicXML or MEI documents. Serialization of *vScores* implies the "decoration" of pure content with rendering features. Voices must be assigned to staves, the clef must be chosen based on the voices range, alterations must be displayed according to some general policy, etc. We do not elaborate on this process which, as explained above, is directly related to the complex issues of concerns separation in the music notation domain and falls beyond the scope of our current work. We note that, in some sense, the vision outlined above constitutes a possible framework to investigate this issue. A way to provide a meaningful distinction between content and rendering would indeed be to define a pair of (mapping / serialization) operations that produce an alternative rendering of a score while preserving its content.

In the subsequent sections, we implement this vision with a conceptual model applied to CMN, and expose our technical choices to make the whole approach practical.

3. THE DATA MODEL

We now present a simple data model that extends the relational model with the concepts of voices and events. The model features a core algebra which is mostly illustrated via examples expressed in a high-level language. A formal presentation of the algebra can be found in [9].

3.1 Schema: events, voices, scores and opera

CMN scores are modeled as polyphonic pieces composed of "voices", each voice being a sequence of "events" in some music-related domain (notes, rests, chords, syllabs) such that only one event occurs at a given instant for a given duration. The concepts of voices and events (with non null duration) are shared by most of the encodings we are aware of, in the field of CMN.

3.1.1 Events

An event e is some value v observed during an interval $[t_1, t_2[$. For our purposes, v is any value taken from a domain **dom**, and we note $\mathcal{E}$(**dom**) the set of events on **dom**. Of particular interest are the following (musical) domains, with some internal operators.

- Sounds (**dsound**): represents n simultaneous tones, $n \geq 1$). This covers simple sounds (notes, $n = 1$) and composed sounds (chords, $n > 1$).
- Syllables, (**dsyll**).

Sound is a complex notion that can be decomposed in several components (height, intensity, timbre). In practice, we are limited to those captured by the notational system, mostly the frequency (pitch and octave). Other aspects are sometimes indirectly represented (for instance, timbre by the instrument name).

We do not restrict the events to musical domains. For instance, an event in the **dint** domain might represent the value of an interval between two voices at a given timestamp. Such events can be inferred from the notation, and can enrich the representation. Beyond this simplistic illustration, this permits the definition of generalized scores that extend the usual concept by combining musical events with non-musical domains representing, for instance, some analytic feature.

3.1.2 Voices as Time Series

A musical time series (or *voice* to make it short) is a mapping from the time domain $\mathcal{T}$ (a discrete, ordered set isomorphic to $\mathbb{Q}$) into a set of events $\mathcal{E}$(**dom**). We denote by **Voice(dom)** the type of a voice, where **dom** is the domain of interest.

A voice is an instance of a voice type. So, for instance:

- v_1: **Voice(dsound)** denotes a voice v_1 which represents a function from $\mathcal{T}$ to "pure" music events.
- v_2: **Voice(`dint`)** denotes a voice v_2 which represents a function from $\mathcal{T}$ to integers, such as the intervals between two (music) voice.
- v_3: **Voice(dsyll)** denotes a voice which represents a function from $\mathcal{T}$ to text, such as lyrics.

Since a voice is a function, there is exactly one event at each instant (in other words, events cannot overlap). We can partly relax this constraint by adding to each domain a distinguished *null value* $\perp$ which denotes the "absence" of event (see [9] for a detailed discussion).

3.1.3 Scores as synchronized time series

We can now define scores. At a basic level, a score is a synchronization of voice(s). We extend this definition to capture a recursive organization of scores built from subscores.

- v a voice, then v is a score.
- if $s_1, \cdots, s_n$ are scores, the sequence $< s_1, \cdots, s_n >$ is a score.

The type of a score is the enumeration of voices that constitute a score, associated with their names. For instance:

1. The type T_q of a quartet is

```
[violin1: dsound, violin2: dsound,
     alto: dsound, cello: dsound ]
```

2. The type T_v of a vocal part is:

```
[lyrics: dsyll, monody: dsound]
```

3. The recursive structure of a score with a vocal part of type T_v and a figured bass is

```
[vocal: Tv, bass: dsound]
```

Instances of these types are time series from $\mathcal{T}$ to, respectively, **dsound**^{4}, **dsound** $\times$ **dsyll**, and (**dsound** $\times$ **dsyll**) $\times$ **dsound**. Conceptually, the first one represents a function which associates to each timestamp a 4-tuple of music events, the second one a function which associates to each timestamp a pair (sound, syll). Essentially, a score extends the concept of voice (i) by allowing several events to occur simultaneously and (ii) by labelling events with names, providing a "hook" to refer to them with operations. Unifying the model for voices and scores makes it easy to define operations that remain in a consistent setting.

3.1.4 Corpora as extended relations

Finally, an *opus* is a tuple of values which can either be atomic values (strings, integers, floats) or scores. Opuses with similar structure can be grouped in a *Corpus*. If we compare with the standard relational approach, a corpus is a container of similar objects, akin to a table, and an opus is an element in the container (a row in the table). A database is a set of corpora.

Since a corpus gathers opera with similar type, this type can be summarized as a corpus schema. The following example shows a possible schema for a *Quartet* corpus.

```
Quartet (id: int,
         title: string,
         composer: string,
         published: date,
         music: Score [v1: dsound,
                       v2: dsound,
                       alto: dsound,
                       cello: dsound
                      ]
         )
```

3.2 User query language

We need a concrete syntax to express our score manipulations. Since we want to limit as much as possible the extension required to adapt our model to an existing system, this leaves two main options: SQL and XQuery. The examples below are based on XQuery which presents several features of interest, including the ability to incorporate functions, and fits naturally with the hierarchical nature of our data items (opuses, made of scores, made of voices, with possibly intermediate levels).

In order to avoid formal developments, the main characteristics of the language are introduced with examples. The interested reader is referred to a companion paper [10] that details the design of the query language and the related implementation issues. The examples below *cannot* be directly evaluated as XQuery expressions, since they are interpreted over virtual instances of the above score model. The actual evaluation relies on a lightweight query rewritting presented in the next section.

The examples rely on the *Quartet* corpus (refer to the previous section for its schema). Our first example creates a list of the Haydn's quartets, reduced to the violin's parts.

```
for $s in collection("Quartet")
where $s/composer="Haydn"
return $s/title, Score($s/music/v1, $s/music/v2)
```

Recall that `music` is an attribute of type **Score** of the *Quartet* corpus. This first query shows two basic operators to manipulate scores: projection on score/voices with the standard"/" XPath syntax, and creation of new scores with the `Score()` synchronizer operator.

A third operator that allows the derivation of new score contents is MAP: it represents a higher-order function that applies a given function f to each event in a voice, and returns the voice built from f's results. Here is an example: we want the quartets where the violin1 part is played by a B-flat clarinet. We need to transpose the `v1` part 2 semitones up.

```
for $s in collection("Quartet")
where $s/composer="Haydn"
let $clarinet := Map ($s/music/v1, transpose (2))
let $clrange  := ambitus ($clarinet)
return $s/title, $clrange,
          Score($clarinet, $s/music/v2,
                $s/music/alto, $s/music/cello)
```

This second query shows how to define variables that hold new content derived from the stored scores via *user defined functions* (UDFs). For the sake of illustration we create two variables, `$clarinet` and `$clrange`, calling respectively `ambitus()` and `transpose()`.

In the first case, the function has to be applied to each event of the violin voice. This is expressed with MAP which yields a new voice with the transposed events. By contrast, `ambitus()` is directly applied to the voice as a whole. It produces a scalar value (not a voice).

MAP is the primary means by which new voices can be created by applying all kinds of transformations. MAP is also the operator that opens the query language to the integration of *external* functions: any library can be integrated as a first-class component of the querying system, providing some technical work to "wrap" it conveniently (see next section).

By "mapping" a Boolean expression e to a voice, we can filter out the events that do not satisfy e, replacing them by the `null` event $\perp$. Note that this is different from *selecting* a score based on some property of its voice(s). The next query illustrates both functionalities: we select all the psalms such that the vocal part contains some word,"nullify" the events that do not belong to the first ten measures, and trim the voice to keep only non-null events.

```
for $s in collection("Psalters")
let $sliced := trim(select ($s/air/vocal/monody,
                    measure(5, 10)))
where contains ($s/air/vocal/lyrics, "Heureux")
return $s/title, Score($sliced)
```

We can take several opuses as input and produce an opus with several scores as output. The following example takes three chorals, and produces an opus with two scores associating respectively the alto and tenor voices.

```
for $c1 in coll("Chorals")[@id="BWV49"]/music,
    $c2 in coll("Chorals")[@id="BWV56"]/music,
    $c3 in coll("Chorals")[@id="BWV12"]/music
return <title>Excerpts of chorals</title>,
         Score($c1/alto, $c2/alto, $c3/alto),
         Score($c1/tenor, $c2/tenor, $c3/tenor)
```

Finally, our last example illustrates the extended concept of "score" as a synchronization of voices which are not necessarily "music" voices. The following query produces, for each quartet, a score containing the violin 1 and cello voices, and a third one measuring the gap (interval) between the two.

```
for $s in collection("Quartet")/music
let $intervals := Map(Score($s/v1,$s/cello),
                    interval())
return Score ($s/v1, $s/cello, $intervals)
```

Such a "score" cannot be represented with a traditional rendering. Additional work on visualization tools that would closely put in perspective music fragments along with some computed analytic feature is required.

4. IMPLEMENTATION

Our system has been fully implemented in NEUMA. It integrates an implementation of our score algebra, a mapping that transforms serialized scores to vScores, and off-the-shelf tools (a native XML database, BASEX [1], a music notation library for UDFs, MUSIC21 [2] [11]). This simple implementation yields a query system which is both powerful and extensible (only add new functions wrapped in XQuery/BASEX). We present its salient aspects.

4.1 Architecture and query processing

Figure 2 shows the main implementation modules. Data is stored in BASEX in two collections: the semi-virtual collection (e.g., Quartet) of music documents (called *opus*), and the collection of serialized scores, in MusicXML or MEI. Each virtual element `scoreType` in the former is linked to an actual document in the latter. Those collections are managed by the NEUMA digital library[12].

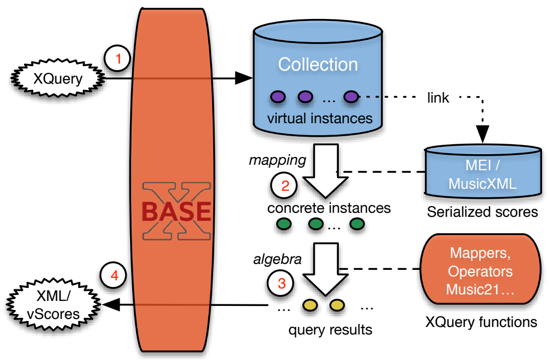

Figure 2. Architecture

The evaluation of a query proceeds as follows. First (step 1), BASEX scans the virtual collection and retrieves the opus matching the `where` clause. Then (step 2), for each opus, the embedded virtual score element has to be materialized. This is done by applying the mapping that extracts a vScore instance from the serialized score, thanks to the link in each opus.

Once a vScore is instantiated, algebraic expressions, represented as composition of functions in the XQuery syntax, can be evaluated (step 3). We wrapped several Python and Java libraries as XQuery functions, as permitted by the BASEX extensible architecture. In particular, algebraic operators and mappers are implemented in Java, whereas additional, music-content manipulations are mostly wrapped from the Python Music21 toolbox.

The XQuery processor takes in charge the application of functions, and builds a collection of results, finally sent to the client application (step 4). It is worth noting that the whole mechanism behaves like an ActiveXML [13] document which *activates* the XML content on demand by calling an external service (here, a function).

[1] http://basex.org
[2] http://web.mit.edu/music21

4.2 External components

How do we integrate *functions* that manipulate the score representation? In general, we need to resort on an external component. Getting the highest note of a voice for instance is hardly expressible in XQuery. In general, getting such features would require awfully complex expresssions. This is due to very detailed decomposition of any XML encoding which makes very difficult the reconstruction of high-level features.

XQuery is extendible to user-defined functions, and the point is technically harmless. In our current implementation, we simply "wrap" relevant functions in an external library compliant to BaseX. The following example retrieves all the quartets such that the first violin part gets higher than e6, using a *highest()* UDF.

```
for $s in collection("Quartet")
where highest($s/music/v1)  > 'e6' return $s
```

A naive, direct evaluation would load the MusicXML (or MEI) document from the underlying storage, pass it to the function and get the result. This works with quite limited implementation efforts. Such an evaluation raises, however, strong efficiency issues. In general, any function will need to access to the whole score encoding (or to put it differently, we cannot in general anticipate the part of the score it needs to access). This has to be done for each score in the collection: a clearly unacceptable burden, likely to make the full query process highly inefficient.

A solution is to materialize the results of User Defined Functions as metadata in the virtual document and to index this new information in BASEX. This can directly serve as a search criteria without having to materialize the vScore. The result of the *highest()* function is such a feature. Index creation simply scans the whole physical collections, runs the functions and records it result in a dedicated `index` sub-element of each opus, automatically indexed in BASEX. To evaluate the query above, it uses the access path to directly get the relevant opus.

```
for $s in collection("Quartet")
where index/v1/highest  > 'e6'
return $s
```

5. RELATED WORK

Music Information Retrieval has mostly considered so far unstructured search, and notably similarity search [14]. Unstructured search is convenient to the end user, and avoids intricate considerations related to music notation structure. A limitation is that the granularity of results stays at the document level, and cannot access to finer internal components. Our work allows such a fine-grained inspection.

An early attempt to represent scores as structured files and to develop search and analysis functions is the HumDrum format. Both the representation and the procedures are low-level (text files, Unix commands) which make them difficult to integrate in complex application. We are only aware of a few other approaches. An attempt to transpose database principles to score management is presented in [15]. The authors of [16] study how XQuery may be directly used over MusicXML. XQuery is a general-purpose query language which hardly adapts to the specifics of symbolic music manipulation. Besides, by ignoring the issue of the inherent underlying data model, closure of operations becomes undecidable, and the query language misses the essential properties that makes it safely usable in applications.

We make the case for a clear identification of the data model that underlies the operations on "scores". This allows to abstract from useless details, brings a support to the definition of closed operations, and enforces to review what kind of content we aim at manipulating. We might always (rightly) complain that part of the meaningful content is lost, and that rare features (e.g., chords with varying note durations) are not adequately captured by an abstract model, but this seems the price to pay for a clear understanding of the stakes. As a side effect, this allows to integrate distinct encodings in a consistent setting.

The mapping process by which this is achieved is reminiscent from mediation architectures used for data sources integration [17, 18, 19, 20], and can be seen as an application of method that combines queries on physical and virtual instances. It borrows ideas from ActiveXML [13], and in particular the definition of some elements as "triggers" that activate external calls.

Abstracting an agnostic score content from XML formats is a design shared by several earlier proposals, including NEUMA [21], Music21 [11] and formal approaches such as Euterpea [22] that attempt to model music content for generative or analytic purposes. This allows in particular to develop manipulation primitives independently from serialization concerns. We can re-use for instance in our implementation some of the analytic functions supplied by Music21, and combine these functions to the structural database operators that constitute the core of our contribution.

6. CONCLUSION AND FUTURE WORK

We propose a new approach to treat music notation as a structured source of information apt at supporting modern query techniques inspired by object and relational databases. A debatable aspect of the approach is the lossy mapping that extract "content" from the notation. There is no well-defined answer to the separation of score content from score rendering, which can be perceived as an encouragement to further investigate the issue.

On the other hand, having a high-level specification language to combine, change and derive scores offers quite promising perspectives for performance, teaching and analysis of music content. We are in particular keen to explore the following ideas:

- Maintain a tight synchronization between the parts of a score and the full ensemble, in order to reflect any change (e.g., an annotation).

- Propose new visualizations of music notation, dictated not by performance issues, but by the need to grasp some analytical aspect.

- Study styling mechanisms which can map an abstract music notational content to sheet representation.

Our implementation in NEUMA is available to the community of scholars, musicologist and data scientists who aim at investigating the corpora of this library for analytic purposes. We hope that the design presented in the paper is generic enough to inspire similar endeavours. We will be glad to provide our software components to anyone wishing to exploit these ideas in a similar system.

Acknowledgments

This work is partially supported by the Musica Consortium, `https://humanum.hypotheses.org/503`.

7. REFERENCES

[1] E. Selfridge-Field, Ed., *Beyond MIDI : The Handbook of Musical Codes*. Cambridge: The MIT Press, 1997.

[2] M. Good, *MusicXML for Notation and Analysis*. W. B. Hewlett and E. Selfridge-Field, MIT Press, 2001, pp. 113–124.

[3] P. Rolland, "The Music Encoding Initiative (MEI)," in *Proc. Intl. Conf. on Musical Applications Using XML*, 2002, pp. 55–59.

[4] "Music Encoding Initiative," http://music-encoding.org, Music Encoding Initiative, 2015, accessed Oct. 2015.

[5] A. Hankinson, P. Roland, and I. Fujinaga, "The Music Encoding Initiative as a Document-Encoding Framework," in *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, 2011, pp. 293–298.

[6] "W3C Music Notation Community Group," https://www.w3.org/community/music-notation/, 2015, last accessed Jan. 2016.

[7] L. Pugin, J. Kepper, P. Roland, M. Hartwig, and A. Hankinson, "Separating Presentation and Content in MEI," in *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, 2012.

[8] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[9] R. Fournier-S'niehotta, P. Rigaux, and N. Travers, "An Algebra for Score Content Manipulation," CEDRIC laboratory, CNAM-Paris, France, Tech. Rep. CEDRIC-16-3616, 2016, submitted for publication. [Online]. Available: http://cedric.cnam.fr/index.php/publis/article/FRT16d

[10] ——, "Querying XML Score Databases: XQuery is not Enough!" CEDRIC laboratory, CNAM-Paris, France, Tech. Rep. CEDRIC-16-3612, 2016, submitted for publication. [Online]. Available: http://cedric.cnam.fr/index.php/publis/article/FRT16c

[11] M. S. Cuthbert and C. Ariza, "Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data," in *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, 2010, pp. 637–642.

[12] P. Rigaux, L. Abrouk, H. Audéon, N. Cullot, C. Davy-Rigaux, Z. Faget, E. Gavignet, D. Gross-Amblard, A. Tacaille, and V. Thion-Goasdoué, "The design and implementation of neuma, a collaborative digital scores library - requirements, architecture, and models," *Int. J. on Digital Libraries*, vol. 12, no. 2-3, pp. 73–88, 2012.

[13] S. Abiteboul, O. Benjelloun, and T. Milo, "The active XML project: an overview," *VLDB J.*, vol. 17, no. 5, pp. 1019–1040, 2008. [Online]. Available: http://dx.doi.org/10.1007/s00778-007-0049-y

[14] R. Typke, F. Wiering, and R. C. Veltkamp, "A Survey Of Music Information Retrieval Systems," in *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, 2005.

[15] Z. Faget and P. Rigaux, "A Database Approach to Symbolic Music Content Management," in *Intl. Symp. on Exploring Music Contents (CMMR)*, 2010, pp. 303–320.

[16] J. Ganseman, P. Scheunders, and W. D'haes, "Using XQuery on MusicXML Databases for Musicological Analysis," in *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, 2008.

[17] H. Garcia-Molina, J. Ullman, and J. Widom, *Database System Implementation*. Prentice Hall, 2000.

[18] A. Doan, A. Halevy, and Z. Ives, *Principles of Data Integration*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012.

[19] S. Abiteboul and et al., "WebContent: Efficient P2P Warehousing of Web Data," in *VLDB'08 Very Large Data Base*, August 2008, pp. 1428–1431.

[20] N. Travers, T. T. D. Ngoc, and T. Liu, "Tgv: A tree graph view for modeling untyped xquery," in *(DASFA) 12th International Conference on Database Systems for Advanced Applications*. Springer, 2007, pp. 1001–1006. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-71703-4_92

[21] P. Rigaux and Z. Faget, "A database approach to symbolic music content management," in *Exploring Music Contents - 7th International Symposium, CMMR 2010, Málaga, Spain, June 21-24, 2010. Revised Papers*, 2010, pp. 303–320.

[22] P. Hudak, *The Haskell School of Music – From Signals to Symphonies*. (Version 2.6), January 2015.