Querying Music Notation

Raphaël Fournier-S'niehotta, Philippe Rigaux, Nicolas Travers
Cedric/CNAM, Paris
firstName.lastName@cnam.fr

Abstract—**We consider the emerging field of music score libraries, where documents rely on a music notation markup language such as MusicXML or MEI. We propose to model as synchronized time-series the music structure that can be extracted from such documents, together with an algebra that operates in closed form and allows manipulations, restructurings, and combinations of music scores stored in a database. We formally present the model, its algebraic operators, and finally show how our approach can serve as a building block for a query and analytic language on large collections of XML-encoded music scores.**

I. Introduction

Music is a temporal organization of sounds. Moreover, this organization takes in many cases (and, at the very least, for most of the Western music repertoire) the form of temporal sequences of musical events, where each event corresponds to the production of sound(s), at a given frequency and for a given duration. We will call *voice* such a sequence, a terminology that originates in early vocal music, and has been generalized to instrumental music.

Digital encoding of music is mostly represented by audio files, in which both the music structure and the charateristics of each individual voice are difficult to capture accurately. But music content can also be represented in a notational form that has been used for centuries to preserve and exchange music works, i.e., *music scores*. There is a growing availability of digital score libraries (DSL) of music encoding in XML-based languages such as MusicXML [1] or MEI [2], [3]. The W3C recently launched a normalization endeavor [4], confirming that we are heading toward the maturity of this emerging field. Those encodings provide fine-grained access to the components of the music notation, and open the perspective of future large collections of scores, calling for adapted querying facilities. In the present paper, we propose an approach to model and query score databases by manipulating music content as it is conveyed by its notation.

The structure of music scores

Figure 1 shows a first example of a *monophonic score*, where the musical content is essentially represented by a sequence of notes (with the exception of the last symbol that denotes a musical silence). The height of each note corresponds to its intentional frequency, whereas its "color" (black, white, hamped black, etc.) denotes its intentional duration. The horizontal notation of a voice represents the temporal axis.

An important feature of this encoding is that there is no overlapping of the timespans covered by two distinct notes

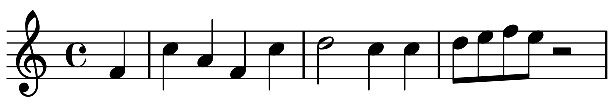

Figure 1. A monophonic score

(of course a musician is free to break this constraint to some extent, but we will ignore such performance artifacts). This is natural if we consider that the original intent of this notation is to encode the music part assigned to a single singer, who can hardly produce simultaneous sounds. This part is accordingly called a *voice*, and we will use this term to denote the basic structure of music objects representation as time series of musical events, assigned to timespans that do not overlap with one another.

Polyphonic scores are represented as a combination of several voices. Figure 2 gives an illustration (first measures of a Bach's choral). In terms of music content, the important information expressed by the notation is the *synchronization* of four voices, graphically expressed by their vertical alignment.

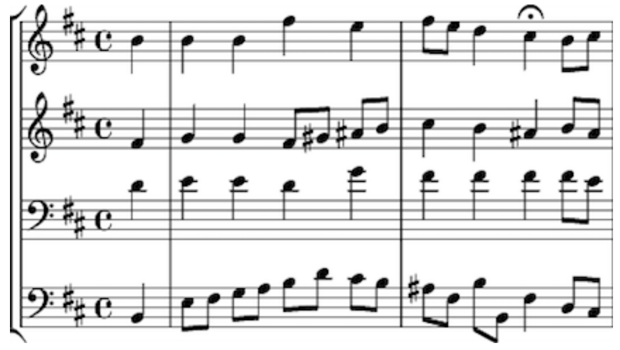

Figure 2. A polyphonic score

Voice synchronization is the second essential aspect that can be extracted from music notation systems, the first one being an accurate representation of the development of each voice. Together they define what we will consider, in the present paper, as the *music content* conveyed by music notation.

The rendering of a score contains several other symbols (clefs, lines, staves) and depends on choices (assignment of each voice to a separate staff for instance) that are more related to readability concerns than to content description. For our purposes, these rendering aspects can be safely ignored. This definition of music content leads to an (abstract) representation of scores as strongly structured temporal objects, and therefore allows to envisage operations that transform a score, retrieve scores that match some search criteria, combine several scores, and in general produce new score instances from some pre-existing material.

Scores as synchronized time series

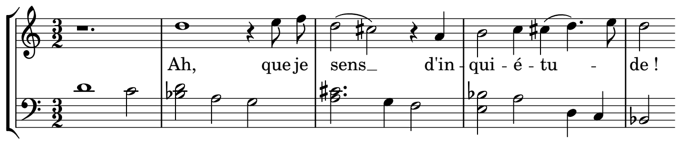

Figure 3. Running example

Finally, this modeling perspective can easily be extended to capture the more general concept of synchronized time series built from arbitrary value domains. Consider our third example shown on Figure 3 (used as a running example in the rest of the paper). It shows a score with two parts. The lower one consists of a single voice. The upper one is a vocal part, with two voices, the first one composed of sounds, and the second one of syllables (note that there is no one-to-one rythmic correspondence between syllables and notes, as some syllables cover several notes). The latter is an example of a temporal function that, instead of mapping timestamps to sounds, maps timestamps to syllabs.

This suggests a direct generalization of the concept of voice as functions mapping timestamps to values in arbitrary domains. This can obviously be useful to associate music score with additional content. For instance, we can take the two music voices of Figure 3, and compute a third, derived "voice" that continuously represents the gap (interval) between them, at each instant.

Goals and contribution

We propose a modeling of music scores as structured objects, in a database perspective that aims at manipulating large collections of such objects, in closed form, with expressive operators. This modeling ignores rendering (staves and other layout options) or performance (interpretation by a specific artist) concerns, and abstracts the music content as a synchronization (expressed by the vertical axis in the graphic representation) of temporal sequences of events (expressed by the horizontal axis). As a natural generalization of this modeling approach, we accept polymorphic events that can either represent sounds, or any data of interest that makes sense as a time-dependent information synchronized with the music content.

The present paper proposes the following contributions:

- a formal data model to represent generalized scores (Section II),
- a minimal algebra, operating in closed form in the space of generalized scores (Section III),
- and, finally, a description of how our algebra can be incorporated in a standard data management system to search and manipulate large collections of scores (Section IV).

To the best of our knowledge, this constitutes the first attempt to address modeling and querying issues on collections of music content in a data management perspective. We review the state-of-the-art in Section V, and conclude the paper in Section VI.

II. The Data Model

Throughout this section we will use the score of Figure 3 as a running example.

A. Time

Music rythm is usually organized with respect to a beat which can be divided by any ratio (1:3, 3:2, 9:15 and so on). The time domain $\mathcal{T}$ is therefore a discrete, countable ordered set isomorphic to $\mathbb{Q}$. For the sake of illustration, we will however simplify our examples by assuming that there exists a time unit representing the smallest interval between two musical events. In our running example, each beat can be divided by three at most, the time unit is an eighth note, and each measure can be decomposed in exactly 12 time units.

A *temporal partition* $\mathcal{P} = \{I_1, \cdots, I_n\}$ is a set of right-open time intervals $I_i = [t_1^i, t_2^i[$ such that $\forall i, j, I_i \cap I_j = \emptyset$. Note that, since a music piece covers a finite period and may contain "holes" (rests), we do not impose the usual coverage condition $\bigcup_i I_i = \mathcal{T}$.

We will need the notion of *merge-compatible partitions*, defined as follows.

Definition 1: Two temporal partitions $\mathcal{P}_1 = \{I_1, \cdots, I_n\}$ and $\mathcal{P}_2 = \{J_1, \cdots, J_m\}$ are *merge-compatible* iff their union is a temporal partition, i.e., $\forall i \in [1, n], j \in [1, m]$, either $I_i \cap J_j = \emptyset$ or $I_i = J_j$.

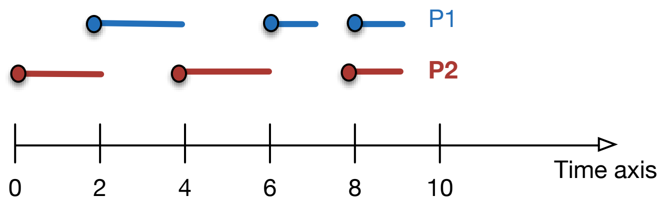

Figure 4. Merge-compatible temporal partitions

Intuitively, two partitions are merge-compatible if their union does not create any new interval. For instance, $P_1 = \{[0, 2[, [4, 6[, [8, 9[\}$ and $P_2 = \{[2, 4[, [6, 7[, [8, 9[\}$ are merge-compatible. Any interval in their union:

$$P_1 \cup P_2 = \{[0, 2[, [2, 4[, [4, 6[, [6, 7[, [8, 9[\}$$

can be found in either P_1 or P_2 (Figure 4). The concept is useful to capture two distinct interpretations of synchronized time series. The first one, that we call *polyphonic* in a music context, imposes each event (each note) to be an atomic value, and represents simultaneous events as distinct voices. The second one, called *harmonic* interpretation, accepts events made of complex values. We will futher illustrate this distinction and the related algebraic operator in Section III.

$$v_{sopr}(t) = \begin{cases} \bot, & t \in [0,12[\\ D5_{12}^{20}, & t \in [12,20[\\ \bot, & t \in [20,22[\\ E5_{22}^{23}, & t \in [22,23[\\ F5_{23}^{24}, & t \in [23,24[\\ D5_{24}^{28}, & t \in [24,28[\\ C\#5_{28}^{32}, & t \in [28,32[\\ \bot, & t \in [32,34[\\ A4_{34}^{36}, & t \in [34,36[\end{cases} \quad v_{lyrics}(t) = \begin{cases} \bot & t \in [0,12[\\ \text{Ah}_{12}^{20}, & t \in [12,20[\\ \bot, & t \in [20,22[\\ \text{que}_{22}^{23}, & t \in [22,23[\\ \text{je}_{23}^{24}, & t \in [23,24[\\ \text{sens}_{24}^{32}, & t \in [24,32[\\ \bot, & t \in [32,34[\\ \text{d'in}_{34}^{36}, & t \in [34,36[\end{cases} \quad v_{bass}(t) = \begin{cases} D4_{0}^{8}, & t \in [0,8[\\ C4_{8}^{12}, & t \in [8,12[\\ < B3es, D4 >_{12}^{16}, & t \in [12,16[\\ A3_{16}^{20}, & t \in [16,20[\\ G3_{20}^{24}, & t \in [20,24[\\ < A3, C4is >_{24}^{30}, & t \in [24,30[\\ G3_{30}^{32}, & t \in [24,32[\\ F3_{32}^{36}, & t \in [32,36[\end{cases}$$

Figure 5. Voice as time series for our running example (measures 1 to 3)

B. Events

An event e is some value v from a domain **dom** observed during an interval $[t_1, t_2[$.

Definition 2 (Event): Let **dom** be a domain of values. An *event* $e = a_{t_1}^{t_2}, a \in \textbf{dom}, t_1, t_2 \in \mathcal{T}, t_1 < t_2$, represents the fact that a is observed from t_1 (included) to t_2 (excluded). We note $\mathcal{E}(\textbf{dom})$ the set of events on **dom**.

Of particular interest are the following (musical) domains, with some internal operators.

- (**dsound**, :=:, :+:, :-:): is the domain of harmonic sounds, i.e., production of n simultaneous tones, $n \geq 1$. This covers single notes ($n = 1$) and composed sounds (chords, $n > 1$).
- (**dsyll**, $||$) is the domain of syllables (units of pronunciation).

Sound is a complex notion that can be decomposed in several components (height, intensity, timbre). In practice, we are here limited to those captured by the notational system, i.e., essentially the frequency (pitch and octave).

Domains can be equipped with internal operators. In the case of sounds, we will use the harmonic operator :=: (simultaneous playing of two sounds), the transposition operator :+:, and the interval operator :-:.

Example 1: Some examples of events drawn from our running example.

- $D5_{12}^{20}$ is a **dsound** event representing the first note of the upper voice[1] The value is a $D5$ and it extends from timestamp 12 (beginning of second measure) to timestamp 20 (excluded).
- Ah_{12}^{20} is a (syllabic) event over the same time period.
- $< B3es, D4 >_{12}^{16}$ is a **dsound** event (a chord), in the lower voice, over [12,16[

We do not restrict the events to musical domains. 2_{12}^{16} for instance is an event in the **dint** domain. It can be the result of an analysis, stating that two (2) notes are simultaneously played from 12 to 16 by the lower voice. Such events can be inferred from the notation, and can enrich the representation. Beyond this simplistic illustration, this allows the definition of generalized scores that extend the usual concept by combining musical events with non-musical domains representing, for instance, some analytic feature.

[1]We adopt the convention of representing note classes by the first seven letters of the Latin alphabet. See https://en.wikipedia.org/wiki/Musical_note.

C. Voice

Events are not represented individually, but always as part of time series, simply called *voices*.

Definition 3 (Voice): A voice v of type **Voice(dom)** is a partial function from $\mathcal{T}$ to $E_v \subset \mathcal{E}(\textbf{dom})$ such that

$$a_{t_1}^{t_2} \in E_v \Leftrightarrow v(t) = a_{t_1}^{t_2}, \forall t \in [t_1, t_2[$$

The definition of a voice as a function implicitly expresses a non-overlapping constraint: at any timestamp t, there is at most one event e such that $v(t) = e$. A voice also implicitly defines a temporal partition of $\mathcal{T}$, denoted $\mathcal{P}(v)$, and called in the following the *active temporal domain* of v. On the other hand, since the function is partial, we might find timestamps t such that $v(t)$ is undefined, noted $v(t) = \bot$, where $\bot$ somewhat represents the absence of an event.

Figure 5 shows the three voices of our running example for the first three measures. This representation can be found, in a more or less complex syntactic form, in common encoding formats, often mixed with other notational elements representing rendering instructions and performance directives. Our modelling can therefore be seen as a proposal to abstract, from the score notation, an information set that focuses on the music content aspects.

D. Scores

Voices can be *synchronized*. The synchronization of two voices v_1 **Voice(dom$_1$)** and v_2 in **Voice(dom$_2$)**, denoted $v_1 \oplus v_2$, is the voice v_3 in **Voice(dom$_1$ $\times$ dom$_2$)** such that $\forall t \in \mathcal{T}$, $e_1 \in \mathcal{E}(\textbf{dom}_1)$, $e_2 \in \mathcal{E}(\textbf{dom}_2)$, the following holds:

$$v_3(t) = (e_1, e_2) \Leftrightarrow v_1(t) = e_1 \textit{ and } v_2(t) = e_2$$

It can be easily checked that v_3 is a voice unambiguously defined in a space of composed events, and such that $\mathcal{P}(v_3) = \mathcal{P}(v_1) \cap \mathcal{P}(v_2)$. The synchronization combines two voices by a common interpretation of their respective time domain, defining a new voice on the cross product of their value domains.

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.

Definition 4 (Score): A score is a tree-like structure, inductively defined as follows:

- a voice is a score;
- if $s_1, \cdots, s_n$ are scores, then $s_1 \oplus \cdots \oplus s_n$ is a score.

The *type* of a score S is the tuple of its components' type, each labelled by a name.

The vocal part of our example in Figure 3 is a synchronization of a voice in **dsound** and a voice in **dsyll**; the whole score is a synchronization of the vocal part and a bass.

Example 2: We can recursively build the type of our example as follows:

- The type T_v of the upper part is `[sopr:` **`dsound`**`, lyrics:` **`dsyll`**`]`
- The type T_b of the whole score is `[vocal:` T_v`, bass:` **`dsound`**`]`

Assigning a label to a component of a score is akin to name an axis in a multidimensional space. A score can actually be seen as a time series in this space. An instance of T_b is a function $\mathcal{T} \rightarrow \mathcal{E}(\textbf{dsound}) \times \mathcal{E}(\textbf{dsyll}) \times \mathcal{E}(\textbf{dsound})$. Measure 3 for instance is represented as:

$$\begin{cases} (D5_{12}^{20}, \text{Ah}_{12}^{20}, < B3es, D4 >_{12}^{16}), t \in [12, 16[\\ (D5_{12}^{20}, \text{Ah}_{12}^{20}, A3_{16}^{20}), & t \in [16, 20[\\ (\perp, \perp, G3_{20}^{24}), & t \in [20, 22[\\ (E5_{22}^{23}, \text{que}_{22}^{23}, G3_{20}^{24}), & t \in [22, 23[\\ (F5_{23}^{24}, \text{je}_{23}^{24}, G3_{20}^{24}), & t \in [23, 24[\end{cases}$$

The same pair of atomic events ($D5_{12}^{20}$, Ah_{12}^{20}) from the `sopr` voice appears twice, associated with two distinct events of the bass voice, due to the non-homorythmic synchronization.

III. The score algebra

We now define the score algebra ScoreAlg. It consists of a set of *structural operators* that take score instances as input and produce a score instance as output (algebraic closure). Our algebra includes the mean to apply external functions to score contents. Being able to incorporate functions allows to extend the language to arbitrary content transforms, as long as closure constraints are fullfilled.

A. Functions

A *temporal function* is a mapping from $\mathcal{T}$ to $\mathcal{T}$. We consider only the class of linear functions $\tau_{m,n}, m \neq 0$ of the form:

$$\tau_{m,n}(t) = mt + n$$

A specific subclass consists of *warping functions*, of the form $warp_m : \mathcal{T} \rightarrow \mathcal{T}$, $t \mapsto mt$; and *shifting functions* of the form $shift_n : \mathcal{T} \rightarrow \mathcal{T}$, $t \mapsto t + n$. They respectively extend or shrink the event durations and move events in $\mathcal{T}$ (temporal translation).

A *domain function* maps a value from some multidimensional events space $\mathcal{E}(\textbf{dom}_1) \times \cdots \times \mathcal{E}(\textbf{dom}_\text{n})$ to a single value in some domain $\mathcal{E}(\textbf{dom}_\text{o})$. We extend their interpretation to events naturally. If $e = a_{t_1}^{t_2}$ is an event on **dom**, then:

1) (Temporal function) $\tau_{m,n}(a) = a_{\tau_{m,n}(t_1)}^{\tau_{m,n}(t_2)}$
2) (Domain function) if f is a domain function on **dom**, then $f(e) = f(a)_{t_1}^{t_2}$

The *core functions* of the algebra consists of (i) the linear temporal function and (ii) the internal domain operators. Here are some examples of applying core functions to the **dsound** event $e = D5_{12}^{20}$,

- $\tau_{2,0}(e) = D5_{24}^{40}$ (warping)
- $\tau_{0,5}(e) = D5_{17}^{25}$ (translation)
- :+:$(e, 2) = E5_{17}^{25}$ (transposition)

The algebra is designed to incorporate external (or *user-defined*, UDF) functions beyond the native ones. We will explain later how arbitrary functions can be integrated in a query expression.

B. Structural operators

ScoreAlg ($\oplus, \mu, \sigma, \circ$, map) consists of five operators for, respectively, synchronization, projection, selection, merge and a special *map* operator to apply external functions. Each operator takes one or two scores as input and produces a score.

Remark 1: For the sake of simplicity, the definition of operators is given on "flat" scores. Their extension to the full recursive structure is immediate.

The synchronization operator, $\oplus$, has already been introduced in Section II. It combines two scores.

Definition 5 (Synchronization, $\oplus$): If S_1 and S_2 are two scores, then $S_1 \oplus S_2$ is a score defined by

$$[S_1 \oplus S_2](t) = (S_1(t), S_2(t)), \forall t \in \mathcal{T}.$$

If the component names in S_1 and S_2 are not fully distincts, a renaming is necessary. The operator ρ (not presented here) is similar to the relational operator one. The synchronization of S_{sopr} and S_{lyrics} is:

$$[S_{sopr} \oplus S_{lyrics}](t) = \begin{cases} (\perp, \perp), & t \in [0, 12[\\ (D5_{12}^{20}, \text{Ah}_{12}^{20}), & t \in [12, 20[\\ (\perp, \perp), & t \in [20, 21[\\ (E5_{21}^{22}, \text{que}_{21}^{22}), & t \in [21, 22[\\ (F5_{22}^{23}, \text{je}_{22}^{23}), & t \in [22, 23[\\ (D5_{24}^{28}, \text{sens}_{24}^{32}), & t \in [24, 28[\\ (C5is_{28}^{32}, \text{sens}_{24}^{32}), & t \in [28, 32[\\ (\perp, \perp), & t \in [32, 34[\\ (A4_{34}^{36}, \text{d'in}_{34}^{36}), & t \in [34, 36[\end{cases}$$

The projection operator μ behaves as the corresponding relational operator. It extracts one or several voices from a score.

Definition 6 (Projection, μ): If S is a score with type $[v_1 : \textbf{dom}_1, \cdots, v_n : \textbf{dom}_\text{n}]$, then $\mu_{v_{i_1}, \cdots, v_{i_m}}(S), \forall i_j, j \in [1, n]$ is a score defined as:

$$[\mu_{v_{i_1}, \cdots, v_{i_m}}(S)](t) = (S.v_{i_1}(t), \cdots S.v_{i_m}(t))$$

The following example shows the score obtained by projecting out the lyrics voice (measure 3).

$$\mu_{sopr,bass}(S(t)) = \begin{cases} (D5_{12}^{20}, < B3es, D4 >_{12}^{16}), t \in [12,16[\\ (D5_{12}^{20}, A3_{16}^{20}), & t \in [16,20[\\ (\perp, G3_{20}^{24}), & t \in [20,22[\\ (E5_{22}^{23}, G3_{20}^{24}), & t \in [22,23[\\ (F5_{23}^{24}, G3_{20}^{24}), & t \in [23,24[\end{cases}$$

The selection operator σ_F keeps unchanged the score events that satisfy a condition F, and replace the other events value by $\perp$.

Definition 7 (Selection, σ): If S is a score with type $T = [v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$ and F a Boolean formula on $\mathcal{T}, \mathbf{dom}_1, \cdots, \mathbf{dom}_n$, then $\sigma_F(S)$ is a score with type T such that for each voice $S.v_i$:

$$[\sigma_F(S)].v_i(t) = \begin{cases} S.v_i(t), if\ F(S.v_i(t)) = \text{true} \\ \perp, & else \end{cases}$$

For instance, still taking our running example and the mono-voice scores of Figure 5.

$$[\sigma_{t\in[12,24[}(S_{sopr})](t) = \begin{cases} D5_{12}^{20}, t \in [12,20[\\ \perp, \quad t \in [20,22[\\ E5_{22}^{23}, t \in [22,23[\\ F5_{23}^{24}, t \in [23,24[\end{cases}$$

The merge operator $\circ$ operates a pairwise fusion of voices from two scores sharing the same type, under the constraint that their temporal partitions are merge-compatible (Def. 1). It allows, in particular, the sequential alignment of two scores.

Definition 8 (Merge, $\circ$): If S_1 and S_2 are two scores with type $T = [v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$ such that $\mathcal{P}(S_1.v_i)$ is merge-compatible with $\mathcal{P}(S_2.v_i) = \emptyset, \forall i \in [1,n]$, then $S_1 \circ S_2$ is a score with type T defined as:

$$[S_1 \circ S_2].v_i(t) = \begin{cases} S_1.v_i(t), if\ S_1.v_i(t) \neq \perp, S_2.v_i(t) = \perp \\ S_2.v_i(t), if\ S_1.v_i(t) = \perp, S_2.v_i(t) \neq \perp \\ \perp, \quad if\ S_1.v_i(t) = S_2.v_i(t) = \perp \\ (S_1.v_i(t), S_2.v_i(t)), & else \end{cases}$$

The following example shows a merge between a score S_1 (left) and a score S_2 (middle). Both scores share the same schema, with a unique voice v_1. One can check that their partitions are merge-compatible. The result $S_1 \circ S_2$ is shown on the right.

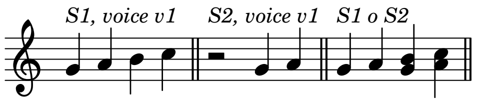

Figure 6. Example of a merge

It is interesting to note that, as long as they are merge-compatible, we obtain the same music by either synchronizing S_1 and S_2 with $S_1 \oplus S_2$, or merge them with $S_1 \circ S_2$. This indeed corresponds to the so-called polyphonic and harmonic interpretations already mentioned in Section II. In terms of data representation, we can either choose two disctincts, mono-sound voices, or a single voice with harmonic sounds.

By defining a notion of "music equivalence" (which is beyond the scope of the paper), we could state that if S_1 and S_2 are two merge-compatible scores, then $S_1 \circ S_2$ is music-equivalent to $S_1 \oplus S_2$. Properties of this kind are made possible by a formal, algebraic approach, and are part of our future investigations.

Finally, the Map operator MAP_f applies a function f to the voices of a score, and returns a mono-voice score containing the result of f.

Definition 9 (Transformation, map): If S is a score with type $[v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$, and f a function $\Pi_i^n \mathbf{dom}_i \rightarrow \mathbf{dom}_o$ then $\text{MAP}_{a:\ f}(S)$ is a score with type [a: $\mathbf{dom}_o$] defined as:

$$[\text{MAP}_{a:\ f}(S)].a(t) = f(S.v_1(t), \cdots S.v_n)(t), \forall t \in \mathcal{T}$$

MAP is a powerful operator to process a score and applies any transformation, as shown by the following examples:

- Translating a mono-voice score n temporal units to the right is expressed by:
$$\text{MAP}_{r:\ \tau_{0,n}}(S)$$
- Transposing a mono-voice score 5 **dsound** units up is expressed as:
$$\text{MAP}_{t:\ :+:5}(S)$$
- If S_{duo} is a two-voices score, computing the sequence of intervals is expressed as:
$$\text{MAP}_{i:\ :-:}(S_{duo})$$
When applied to $S_{duo} = \mu_{sopr,bass}(S)$ for instance, the following result is obtained:
$$\text{MAP}_{i:\ :-:}(S_{duo}) = \begin{cases} 12, t \in [12,16[\\ 17, t \in [16,20[\\ \perp,\ t \in [20,22[\\ 19, t \in [22,23[\\ 20, t \in [23,24[\end{cases}$$
where intervals are delivered as semi-tones.
- Finally, applying a user-defined function, such as computing the sequence of durations of a mono-voice score S, is expressed as:
$$\text{MAP}_{d:\ duration()}(S)$$
One obtains, again, a time series of integers.

C. Algebraic expressions

An expression is defined in a quite standard way:

- If S is a score, then S is an expression.
- If E_1 and E_2 are two expressions, then $E_1 \oplus E_2$, $\text{MAP}_{a:\ f}(E_1)$, $E_1 \circ E_2$, $\sigma_F(E_1)$, $\mu_{v_{i_1},\cdots,v_{i_m}}(E)$ are expressions, with some obvious validity conditions related to the input and output schemas.

Since the result of an operator is always an instance of the model (a score), we can compose operators to create arbitrarily complex expressions. The following list of examples illustrates the expressivity of the resulting language.

- Cutting a temporal slice $[t_1, t_2[$ in a score S, and translating the resulting fragment n temporal units to the right is expressed by:

$$\text{MAP}_{c:\ \tau_{0,n}}(\sigma_{t\in[t_1,t_2[}(S))$$

- We can cut another slice $[0, n-1[$ in another score S' (with same type) and merge with the previous one thanks to composition:

$$\sigma_{0,n-1[}(S') \circ \text{MAP}_{c:\ \tau_{0,n}}(\sigma_{t\in[t_1,t_2[}(S))$$

- Transposing voice v_i of a (mono-voice) score S is as simple as applying *:+:* with MAP.

$$\text{MAP}_{v:\ :+:x}(S)$$

- The following expression transposes voice v_i in S, and synchronizes the result with the other voices of S.

$$\mu_{type(S)-v_i}(S) \oplus \text{MAP}_{v_i:\ :+:x}(\mu_{v_i}(S))$$

- Finally, let us extrapolate by mentioning, once again, that our model instances are, in the larger sense, time series in a multi-dimensional space of events, and that these events take their value beyond the music notation domains. The following expression produces such a generalized score by synchronizing the rythm (sequence of durations) and the text of a monody with type [lyrics: dsyll, sopr: dsound].

$$\text{MAP}_{d:\ duration}(\mu_{sopr}(S)) \oplus \mu_{lyrics}(S)$$

D. *Closure and Expressivity*

The following property is immediate from the definitions.

Theorem 1 (Correctness): Let S be a score and E an expression, then $E(S)$ is a score.

The core algebra (without external function) also exhibits a property related to the generation and transformation of music scores.

Theorem 2 (Expressivity): Let S_1 and S_2 be two scores. Then, there exists an expression E in SCOREALG such that $S_1 = E(S_2)$.

Proof (sketch). Consider the simplest possible input, an *atomic score* S_a with a single voice and a single event $a_{t_1}^{t_2}$. Then:

- By applying the core functions *shift*, *warp* and *transpose* with MAP, we can obtain any other single-voice, single-event score $b_{t_1'}^{t_2}$, for any b, t_1', t_2'.
- Merging these scores with the merge operator $\circ$ yields any possible voice with complex events (chords).
- Synchronizing the generated voices with $\oplus$ yields any possible score.

Therefore, there exists an expression E_l such that $S_1 = E_l(S_a)$. Conversely, we can show that, from any score S, whatever its complexity, we can produce an expression from the selection (σ) and projection (μ) operators that yields a single event. Therefore there exists E_q such that $E_q(S_2) = S_a$. Finally, $S_1 = E_l(E_q(S_2))$. □

This property relates to the transformation/generation power of the algebra: it shows its ability to explore the space of music score (for the part of the notation which is covered by our score model). In practice, we can view SCOREALG as an abstraction of the core machinery of score editing, and as a candidate for a safe and complete score programming paradigm. The next section shows how it can also be used as part of a querying system.

IV. APPLICATION: MODELING AND QUERYING DIGITAL SCORE LIBRARIES

We now expose how our algebra can be used as a core component of a querying system for digital libraries of music scores (DSLs). Numerous potential useful exploitations can then be envisaged: search by content and/or metadata, extraction of score fragments and combination of these fragments to produce new scores, automatic production of a score summary, and in general exploration of music notation material for a wide range of purposes, including analytic ones.

To the best of our knowledge, no such tool exists at the moment. The closest possible solution at hand would be to consider a DSL as a (specific) XML database containing MusicXML or MEI documents, and to apply XQuery as an access method, as suggested in [5]. We believe that this approach falls short due to the intricate imbrication of rendering and content in score encoding. To state it briefly, XML is convenient as a format for serializing music notation, but *not* as a data model apt at supporting the operations that we envisage (see [6] for a longer discussion).

A. *Modeling documents featuring music content*

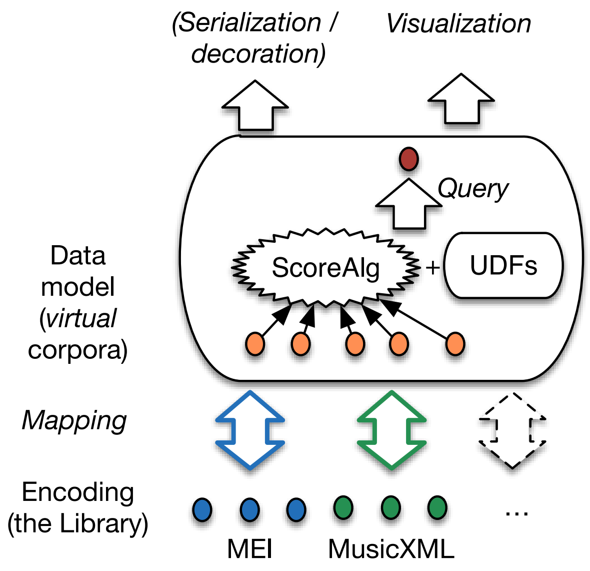

Figure 7. Envisioned system

In the system that we envision (Figure 7), score encoding is seen as a low-level, XML-based serialization format, from which structured music content (gScores), instance of our data model, is obtained by a *mapping* carried out at run-time

Algebraic transformations (Section III) operate on those instances and produce new gScores in the space of data model instances (closure). Finally, a gScore produced by an algebraic expression can either be exploited as such (visualization, analysis, or any user-oriented application), or serialized back in some encoding format.

This allows to incorporate music content, as first-class, queryable objects, in general-purpose databases. We call *opus* any document featuring music content as part of its description. For the sake of simplicity, the following definition assumes a relational representation.

Definition 10: An *opus* is a tuple of values which can either be atomic values (strings, integers, floats) or gScores. A *collection* is a set of opuses.

The following example shows a possible schema for a *Quartet* collection. It consists of standard attributes (a title, a composer's name) together with a `music` attribute of type `Score` that enumerates the list of voices expected from the collections's instances. Score objects are therefore nested, as complex objects, in relational tuples, following the long tradition of non-first normal form relational models [7].

Example 3: Example of a Quartet schema

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

In order to manipulate collection of nested object, we need a small extension of the relational algebra. Beyond the usual operators: selection σ, cartesian product $\times$, union $\cup$ and difference $-$, it features and extended projection, Π.

Definition 11 (Extended projection, Π): Let E be a valid relational algebraic expression with schema $(t_1, \cdots, t_m)$. If $e_1, \ldots, e_q$ are valid SCOREALG expressions, then

$$\Pi_{[e_1(t_1), \cdots, e_q(t_q)]}(E), q \leq m$$

is a valid (relational) algebraic expression.

Its semantics over an instance I is defined as :

$$\Pi_{[e_1(t_1), \cdots, e_q(t_q)]}(E) =$$

$$< e_1(r.t_1), \cdots, e_q(r.t_q) > | r \in E(I)$$

Let's take a concrete example. Assume that we wish to extract, from the collection of Joseph Haydn's quartets, the title and two scores, one built from the first violin and cello part, the other one from the second violin and alto parts.

$$\Pi_{\text{title}, \text{v1} \oplus \text{cello}, \text{v2} \oplus \text{alto}}(\sigma_{\text{composer}='\text{Haydn}'}(\text{Quartet}))$$

Essentially, the extended projection applies SCOREALG expressions to the opuses of a database in order to derive new music scores.

B. XQuery + score algebra = queries

We need a concrete syntax to express our score manipulations. 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). SCOREALG expressions are represented as XQuery functions. This yields a query system that limits as much as possible the extension required to implement our model.

Our first example creates a list of *Haydn*'s quartets, reduced to the titles and violin's parts. This first query shows two basic operators to manipulate scores: projection (μ, obtained with XPath), and creation of new scores with the `Score()` function that corresponds to the *synchronization* operator (Definition 5). We notice the simple usage of SCOREALG operators as functions calls.

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

The MAP operator (Definition 9) is illustrated in the following query. We want the quartets where the v1 part is played by a B-flat clarinet. We need to transpose the $v1$ part 2 semi-tones up.

```
for $s in collection("Quartet")/score
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 also how to define variables that hold new content derived from the gScore via *UDFs*. For the sake of illustration we create two variables, `$clarinet` and `$clrange`, calling respectively `transpose()` and `ambitus()`. While the first function is applied on all events of $v1$ (MAP operator), the `ambitus()` function is directly applied to the voice as a whole. It produces a scalar value.

MAP is the primary means by which new gScores 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, requiring only a small amount of technical work to wrap it conveniently.

The *Select* operator (Definition 7) takes as input a gScore, a Boolean expression e, and filters out the events that do not satisfy e, replacing them by a `null` event. Note that this is different from *selecting* a score based on some property of its voice(s). The next query illustrates both functionalities: a user-defined function "lyricsContains" selects all the psalms (in a *Psalters* collection) such that the vocal part contains a given word ("*Heureux*"), and the *Select* operator "nullifies" the events that do not belong to the measures five to ten.

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

Our system may also take several gScores as input and produce a document with several gScores as output. The following example takes three chorals, and produces a document with two gScores synchronizing voices from respectively the alto and tenor voices.

```
for
 $c1 in coll("Chorals")[@id="BWV49"]/score/music,
 $c2 in coll("Chorals")[@id="BWV56"]/score/music,
 $c3 in coll("Chorals")[@id="BWV12"]/score/music
return <title>Excerpts, chorals 49, 56, 12</title>,
   Score($s1/alto, $s2/alto, $s3/alto),
   Score($s1/tenor, $s2/tenor, $s3/tenor)
```

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

```
for $s in collection("Quartet")/score/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 here required.

C. Implementation

We are currently implementing a system based on off-the-shelf tools (a native XML database, BASEX[2] and a music notation library for UDFs, MUSIC21[3][8]). This simple implementation yields a query system which is both powerful and extensible *via* our SCORELIB library (by adding new functions wrapped in XQuery/BASEX).

The architecture presented in Figure 8 summarizes the components involved in query processing. Data is stored in BASEX in two collections: the semi-virtual collection (e.g., Quartet) of *opuses*, and the collection of serialized scores, in MusicXML or MEI. Each virtual instance is linked to a serialized score in the latter.

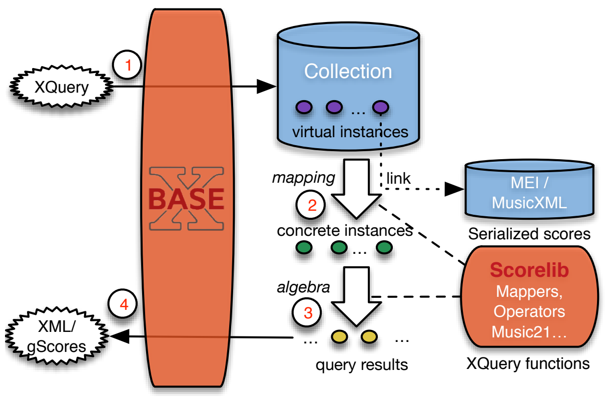

Figure 8. Architecture

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

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 gScore is materialized by applying the mapping that extracts a gScore instance from the serialized score.

Once a gScore 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 allowed 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 (gScores), finally sent to the client application (step 4). It is worth noting that the whole mechanism behaves like an ActiveXML [9] document which *activates* the XML content on demand by calling an external service (here, a function).

V. DISCUSSION AND RELATED WORK

Most of the research conducted in the *Music Information Retrieval* domain so far has focused on unstructured search and similarity measures [10]. This is user-friendly and convenient in a search engine as it avoids to deal with the complex structure of the music notation. However, a downside pertains to the granularity of the result which presents the whole document, or nothing. Our works allows to manipulate the content of notation at a very fine-grained level.

Formal languages for music content representation and manipulation have been proposed, notably Euterpea [11], and a few others [12], [13], [14], but their focus is more on music generation than on processing a persistent collection of scores. What we are targeting here is a language equivalent, for music notation, to the relational algebra: a limited number of types (ideally, one), a minimal set of operators that can be composed, and a well-defined expressive power. Some other works on time series manipulation try to resolve this target [15], [16]. While [15] computes XQuery updates in order to synchronize versions of a document, [16] applies similar operators on labelled structures with timed automata to produce new time series. Both algebras mainly differ on the underlying data model which implies huge differences on definitions of operators (merge, map, select, etc.) making it not proper to score manipulations. It has also a huge impact on the way to implement algebraic expressions and the queries' expressivity.

Our model shares important features with previous works on computational languages for music (notation) generation. Euterpea [11] is an approach based on functional programming (with Haskell) and a rich set of abstract data types to represent music concepts. It offers powerful means to generate complex objects and to study their formal properties. The T-Calculus [13] is an interesting modeling attempt to capture concisely and consistently music flows represented as *tiled streams* and synchronized with the *tiled group* operator. It supports in particular the concept of synchronization points (anacrousis, bars, weak/strong beats) which is so far ignored by our model. Other languages of interest focus on interactive scores [14], [17] or temporal grammars [18].

Some of our core concepts can be found in these earlier works: music events, sound domain with internal operators (we borrow the syntax from [11]), synchronization and concatenation of lists/sequences. What makes in our opinion our approach distinctive is the goal to define a limited set of operators that uniformly manipulate instances of a same data structure (our gScores). This formalism, operating in a closed form with a well-defined expressivity, let us express queries to transform, extract, select, and combine complex objects taken from some storage space. It is quite adapted to a database approach that takes an existing material and produce derived/filtered content. On the other hand, it does not conveniently fit the needs of a *generative* approach based on complex rules, such as the ones required for computational composition.

VI. Conclusion

We proposed an algebra dedicated to time-dependent objects represented as synchronized time series, with a strong focus on the particular case of music content as it is captured by its traditional notation. We believe that the work has an interest by itself, as it constitutes an attempt to formalize the core manipulation operators needed to create and transform scores. It has also a practical scope, since its integration with a collection-oriented query language such as XQuery yields a tool that allows to access large collections of XML scores. Our model and language are also, for their largest part, *generic*, since the dependency to the specific musical field is limited to the events domains, and to the choice of external function. The core of the model can arguably be applied to domain where the comparison of several times series tied by a synchronization constraint is of interest.

Ongoing work focuses on the implementation of the model in Neuma[4], an open digital library of music scores, based on the principles outlined in Setion IV-C. Our goal is to publish an interface that lets advanced user select, transform and associate music content material for analytic or pedagogical purposes. It should be possible for instance to create “working scores” gathering fragments selected from a collection, and sharing some common property. Producing a catalog of incipits, and visualizing derived content or time-dependent annotations, as allowed by our general synchronization mechanism, are some of the interesting perspectives offered by our work.

References

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

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

[3] “Music Encoding Initiative,” http://music-encoding.org, Music Encoding Initiative, 2015, accessed Oct. 2015.

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

[5] J. Ganseman, P. Scheunders, and W. D’haes, “Using XQuery on MusicXML Databases for Musicological Analysis,” in *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2008.

[6] R. Fournier-S’niehotta, P. Rigaux, and N. Travers, “Is There a Data Model in Music Notation?” in *Intl. Conf. on Technologies for Music Notation and Representation (TENOR’16)*, R. Hoadley, C. Nash, and D. Fober, Eds. Anglia Ruskin University, 2016, pp. 85–91.

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

[8] M. S. Cuthbert and C. Ariza, “Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data,” in *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2010, pp. 637–642.

[9] 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

[10] R. Typke, F. Wiering, and R. C. Veltkamp, “A Survey Of Music Information Retrieval Systems,” in *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2005.

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

[12] M. Balaban, “The music structures approach to knowledge representation for music processing,” *Computer Music Journal*, vol. 20, no. 2, pp. 96–111, 1996. [Online]. Available: http://www.jstor.org/stable/3681334

[13] D. Janin, F. Berthaut, M. Desainte-Catherine, Y. Orlarey, and S. Salvati, “The T-Calculus : towards a structured programing of (musical) time and space,” in *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling and design (FARM’13)*, 2013, pp. 23–34.

[14] D. Fober, S. Letz, Y. Orlarey, and F. Bevilacqua, “Programming Interactive Music Scores with INScore,” in *Sound and Music Computing*, Stockholm, Sweden, Jul. 2013, pp. 185–190.

[15] M.-A. Baazizi, N. Bidoit, and D. Colazzo, “Efficient encoding of temporal xml documents,” in *International Symposium on Temporal Representation and Reasoning (TIME)*. IEEE Computer Society, 2011, pp. 15–22.

[16] E. Fares, J.-P. Bodeveix, and M. Filali, “Event algebra for transition systems composition application to timed automata,” in *International Symposium on Temporal Representation and Reasoning (TIME)*. CPS, 2013, pp. 125–132.

[17] D. Fober, Y. Orlarey, and S. Letz, “Scores level composition based on the guido music notation,” in *Proceedings of the International Computer Music Conference*, ICMA, Ed., 2012, pp. 383–386.

[18] D. Quick and P. Hudak, “Grammar-based automated music composition in haskell,” in *ACM SIGPLAN workshop on Functional art, music, modeling and design*, ser. FARM’13, 2013, pp. 59–70.

[4]http://neuma.huma-num.fr