Modeling Music as Synchronized Time Series: Application to Music Score Collections

Raphaël Fournier-S'niehotta, Philippe Rigaux and Nicolas Travers
Conservatoire National des Arts et Métiers (CNAM) - firstName.lastName@cnam.fr

Abstract

Music is a temporal organization of sounds, and we can therefore assume that any representation of music content has a structure that reflects some conceptual and organisational principles. This structure is hardly explicitly accessible in many encodings, such as for instance audio files. However, it appears much more clearly in the *notation* of music.

We thus propose to use the music notation language as a framework to model the content of digital music objects, whatever their specific encoding. We describe an algebra that leverages this structured music representation to extract, restructure, and combine large collection of such objects. The data model leverages the hidden structure of digital music encodings to permit sophisticated manipulations of their content.

We apply the model to music scores collections, taking advantage of the precise representation of music structure that can be found in digital notation. We describe a system, based on an extension of XQuery with our algebra, that provides search, restructuration, extraction, and analytic services on top of large collections of XML-encoded digital scores. Beyond its application to music objects, our work shows how one can build on a structured content embedded in a complex XML encoding to develop robust collection management tools with minimal implementation effort.

1. Introduction

Music is a temporal organization of sounds. Digital encoding of music is mostly represented by audio files, in which this organization is blurred and 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. Music notation can be seen as a mature language to represent the discrete, typed, coordinated elements that together constitute the description of complex music pieces. The core concepts of this language are therefore fully relevant to capture the structure of music. Moreover, once such a structure is known and modeled, it becomes possible to build high-level data management tools to explore large collections of music objects and manipulate their content. The present paper is an attempt to propose such an approach.

The structure of music notation

Let us examine some example of music notation to grasp the essential of its principles. 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.
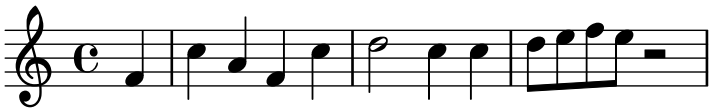


Figure 1: A monophonic score

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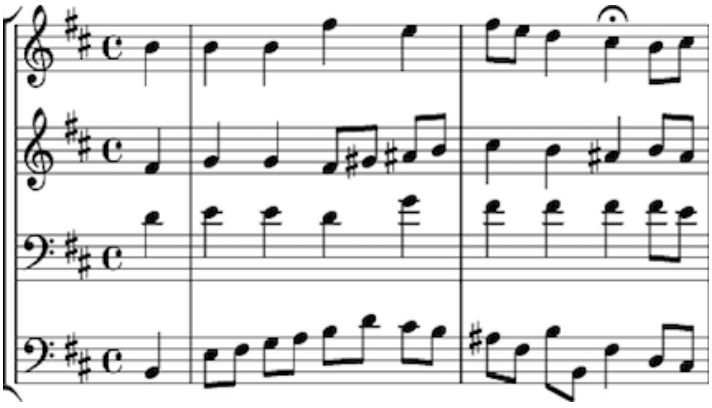


Figure 2: A polyphonic score

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 (structured) *music content*.

Music pieces as synchronized time series

Finally, this modeling perspective can be extended to capture the more general concept of synchronized time series built from arbitrary value domains, and to view the representation of music conveyed by its notation as a specific case. 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.

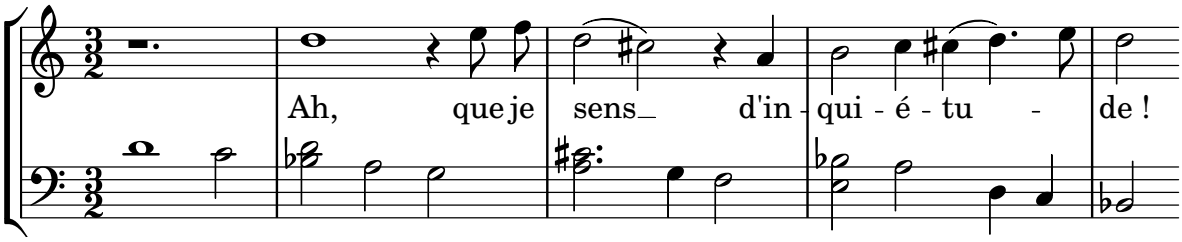


Figure 3: Running example

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, as an encoing of sounds, 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 frequency gap (interval) between them, at each instant.

Position, goals and contributions

The position adopted in the present paper relies on two ideas. First, music notation provides a proven, sophisticated and powerful formal language that provides the basis of a data model for modelling music content. T. Second, instances can be extracted from a digital music object, and this extraction yields a structured representation of this object by which its content can be inspected, decomposed and manipulated.

We can therefore envision a modeling of music as structured objects, in a database perspective that aims at manipulating large collections of such objects, in closed form, with expressive operators. This modeling abstracts the music content as a synchronization (expressed by the vertical axis in the score representation) of temporal sequences of polymorphic 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.

Figure 4 summarizes the envisioned system. We propose in Section 5.4 a technical discussion related to our current implementation choices, but the figure exposes the main conceptual features at a convenient level of abstraction.

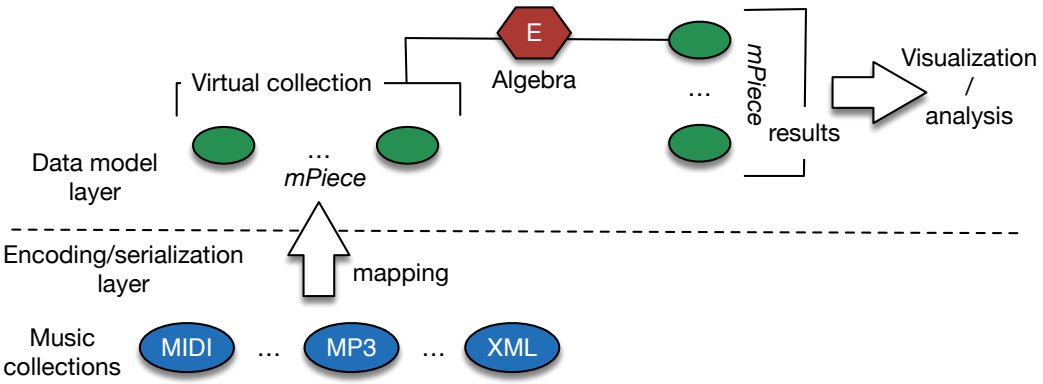


Figure 4: Envisioned system

The bottom layer is a Digital Music Library (DML) managing music objects in some encoding, whether audio (WAV, MP3), MIDI, or XMM. Such encodings are not designed to support content-based manipulations, and, as a matter of fact, it is hardly possible to do so. However, we can *map* the encoding toward a model layer where the content is extracted and structured according to the model structures. These define virtual – no materialization occurs – corpora of structured music objects that we will call *virtual music pieces* (vmPiece) in the following. The data model layer encapsulates both data representation and data operations, and yields a query language whose expressions define the set of transformations that produce new *vmPieces* from the base corpora.

The present paper proposes the following contributions:

- The definition of a robust and expressive *data model* to represent the essentials of music content. Our main guideline here is to consider the music notation language as a guideline.
- An *algebraic language*, consisting in a compact, minimal set of operations that apply to the model space *in closed form*, i.e., mapping instance(s) of the model to other instance(s).
- Finally, an application to collections of digital music scores, encoded in XML. We describe an implementation that supplies a mapping that converts XML scores to instances of our model, extends XQuery with our algebra, and allows to express user-oriented queries over collections of music scores. We fully implemented our system, and publicly demonstrate its capacity in the context of the NEUMA Digital Score Library at http://scorelib.huma-num.fr.

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.

The paper is organized as follows. We review the state-of-the-art in Section 2. We then present the data model in Section 3. The algebra is introduced in Section 4, followed by the integration with XQuery in Section 5. We give the essentials of our implementation choices,

along with query examples and a few directives to integrate on system in an existing platform. We finally conclude the paper in Section 6.

2. State of the art

2.1. Music representation

There exist essentially three different digital representations of music: audio, MIDI, and notation. The most common representation, by far, is sampled audio. The audio signal is sampled (41K samples by second in CD-quality formats), each sample giving an amplitude (loudness) and frequency. The sampling can reproduce the sound accurately enough for human ears. Formats that faithfully represent this information, such as the Waveform Audio File (WAV [1]) present huge storage requirements. A single minute of CD recording for instance consists of 2.4M samples. Compressed formats (MP3 [2]) achieve a compression of one order of magnitude by getting rid of marginal frequencies. Audio format encode an audio signal for sound reproduction purposes

Since the beginning of the century, two XML-based encoding have been proposed: MusicXML [Goo01] or MEI [Rol02, MEI15]. The W3C recently launched a normalization endeavor [W3C15], confirming that we are heading toward a normalized encoding of music scores. Those encodings provide fine-grained access to the components of the music notation, and open the perspective of future large collections of scores. There are built and maintained by librarians, researchers, publishers and active communities of users keen to disseminate rare pieces of music (see for instance the MuseScore repository at https://musescore.com/sheetmusic).

(Audio files, MP4 and MIDI) Most such objects are encoded as audio files [... explain that decoding the content is difficult ...].

(Symbolic notation) Music objects that are encoded in the language of music notation are much easier to process, and we will focus on them.

(Explain that all encodings ahev noises)

XML representations for music scores are however mostly intended as a serialization format for documents that encapsulate all kinds of information, ranging from metadata (creation date, encoding agent) to music content information (symbolic representation of sounds and sounds synchronization) via rendering instructions. The rendering of a score, as shown by Figs. 1, 2, and 3, usually 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, thanks to a mapping that takes the serialization and extracts a structured model instance. To put differently, there is a gap between a Digital Music Library, and a database system supplying a query language addressing the specifics of music content. The present paper proposes an approach to fill this gap.

[1] https://en.wikipedia.org/wiki/WAV
[2] https://en.wikipedia.org/wiki/MP3

Most of the research conducted in the *Music Information Retrieval* domain so far has focused on unstructured search and the related similarity measures, see [TWV05] for a survey on the matter. Unstructured search is convenient for search engines as it is user-friendly and it avoids to deal with the complex structure of the music notation. However, a major downside pertains to the granularity of the result which may only present whole documents, or nothing. The work we present in this article introduces operators to manipulate the internal musical content of the document at a very fine-grained level (e.g.: voices).

The digital encoding of music notation is a long standing endeavour, and has given rise to many proposals exposed in [SF97]. Nowadays, leading encodings are those which rely on XML to represent music notation as structured documents. MusicXML, initially described in [Goo01]) is probably the most widespread one, due to its acceptance as an exchange format by major engraver softwares (Finale, Sibelius, and MuseScore). The MEI initiative is an alternative proposal that borrows much of its inspiration from the Text Encoding Initiative. Its founding goals are presented in [Rol02] and its ongoing activities on [MEI15]. One of its explicit motivations with respect to MusicXML is to promote a declarative approach (stating what is represented) rather than a presciptive one (stating how the represented information must be processed). This is arguably more suitable in the perpective of supplying a robust representation that can support many possible applications, and matches one of the basic database design principles (see for instance the introductory part of [AHV95]). Extending music notation formats beyond the mere purpose of achieving interoperability between editing softwares is also one on the main concerns addresssed by the recent W3C Music Notation Community Group [W3C15] , launched in Sept. 2015.

Formal languages for music content representation and manipulation have been proposed, notably Euterpea, introduced by [Hud15], and a few others ([Bal96, JBDC$^+$13, FLOB13]), but their focus are more on music generation than on processing a persistent collection of scores. Our goal is a language equivalent, for music notation, of 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. In the domain of time series manipulation, [BBC11] and [FBF13] have worked on the subject. In [BBC11], authors compute XQuery updates in order to synchronize multiple versions of a document. [FBF13] present similar operators on labelled structures with timed automata to produce new time series. Both of these algebras mainly differ on the underlying data model which implies huge differences on definitions of operators (merge, map, select, etc.) making it not appropriate 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. [Hud15] proposed with Euterpa 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. In [JBDC$^+$13], authors presented the T-Calculus, an interesting modelling 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 (see [FLOB13, FOL12]) or temporal grammars (such as [QH13]).

Some of our core concepts can be found in these earlier works: music events, sound domain with internal operators (we borrow the syntax from [Hud15]), synchronization and concatenation of lists/sequences. Our approach is however distinct because we aim at defining a limited set of operators that uniformly manipulate several instances of a single data structure (the *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 well adapted to a database approach that takes an existing material and produces 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.

The mapping process at the core of our approach is reminiscent from mediation architectures used for data sources integration (see [GMUW00, DHI12, Aea08, TNL07]), and can be seen as an application of method that combines queries on physical and virtual instances. It borrows ideas from ActiveXML by [ABM08], 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 from [RF10], Music21 from [CA10] and formal approaches such as Euterpea that attempt to model music content for generative or analytic purposes. This allows in particular to develop manipulation primitives independently from serialization concerns. We may 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.

Timse series representation [**?**]

The work presented here has been prepared by several conference talks and short publications. The motivation for extracting a data model from raw music notation encodings is discussed in [FSRT16a]. The formal model and its algebra for synchronized time series is presented in [FSRT16b], and its interfacing wtih XQuery in [FSRT16c]. The current text unifies and expand these ideas in a uniform setting, and adds a concrete, publicly avaiblable system.

3. Music Content Modeling

3.1. 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 $\mathbb{Q}$. 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.

Remark. 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.

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 $\forall i, j \leq n, I_i \cap I_j = \emptyset$. Note that, since a music piece covers a finite temporal range 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* $I_i = J_j$.

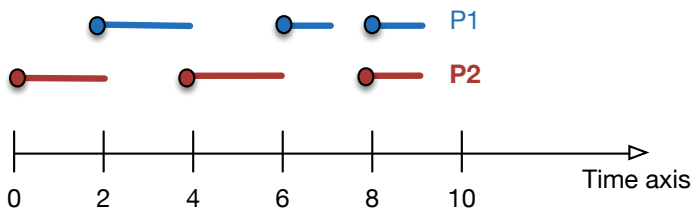


Figure 5: Merge-compatible temporal partitions

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 5). 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 the following.

3.2. Events

$$v_{sopr}(t) = \begin{cases} \bot, & t \in [0, 12[\\ D5^{20}_{12}, & t \in [12, 20[\\ \bot, & t \in [20, 22[\\ E5^{23}_{22}, & t \in [22, 23[\\ F5^{24}_{23}, & t \in [23, 24[\\ D5^{28}_{24}, & t \in [24, 28[\\ C\sharp 5^{32}_{28}, & t \in [28, 32[\\ \bot, & t \in [32, 34[\\ A4^{36}_{34}, & t \in [34, 36[\end{cases} \quad v_{lyrics}(t) = \begin{cases} \bot & t \in [0, 12[\\ \text{Ah}^{20}_{12}, & t \in [12, 20[\\ \bot, & t \in [20, 22[\\ \text{que}^{23}_{22}, & t \in [22, 23[\\ \text{je}^{24}_{23}, & t \in [23, 24[\\ \text{sens}^{32}_{24}, & t \in [24, 32[\\ \bot, & t \in [32, 34[\\ \text{d'in}^{36}_{34}, & t \in [34, 36[\end{cases} \quad v_{bass}(t) = \begin{cases} D4^{8}_{0}, & t \in [0, 8[\\ C4^{12}_{8}, & t \in [8, 12[\\ < B\flat 3, D4 >^{16}_{12}, & t \in [12, 16[\\ A3^{20}_{16}, & t \in [16, 20[\\ G3^{24}_{20}, & t \in [20, 24[\\ < A3, C4\sharp 4 >^{30}_{24}, & t \in [24, 30[\\ G3^{32}_{30}, & t \in [30, 32[\\ F3^{36}_{32}, & t \in [32, 36[\end{cases}$$

Figure 6: Voice as time series for our example (measures 1 to 3)

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 $e = a^{t_2}_{t_1}, a \in \textbf{\textit{dom}}, \mathrm{t}_1, \mathrm{t}_2 \in \mathcal{T}, \mathrm{t}_1 < \mathrm{t}_2$, *represents the fact that value* a *is observed from* t_1 *(included) to* t_2 *(excluded). We note* $\mathcal{E}(\textbf{\textit{dom}})$ *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 rests ($n = 0$), single notes ($n = 1$) and composed sounds (chords, $n > 1$).

  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).

- (**dsyll**, ||) is the domain of syllables (units of pronunciation).

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*[3] *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.*
- $< B\flat 3, D4 >_{12}^{16}$ *is a* ***dsound*** *event (a chord), in the lower voice, over [12,16[*

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 2. *Some examples of domain operators.*

- $B\flat 3$ *:=:* $D \equiv < B\flat 3, D4 >$
- $C4$ *:+:* $2 \equiv D4$ *(assuming that the transposition unit is a semi-tone)*
- $D4$ *:-:* $C4 \equiv 2$ *(same assumption)*

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, generalized scores that extend the usual concept by combining musical events with non-musical domains representing, for instance, some analytic feature.

[3] 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.

3.3. 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 $E_v \subset \mathcal{E}(\boldsymbol{dom})$ 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 $v(t) = e$. Moreover, the function is constant over the timerange covered by e. On the other hand, since the function is partial, we might find timestamps t such that $v(t)$ is undefined, noted $v(t) = \perp$, where $\perp$ somewhat represents the absence of an event. A voice therefore implicitly defines a temporal partition (see Definition 1) of $\mathcal{T}$, denoted $\mathcal{P}(v)$, and called in the following the *active temporal domain* of v.

Example 3. *Figure 6 shows the three voices of our running example for the first three measures. The temporal partitions defined by the voices are:*

1. $\mathcal{P}(v_{sopr}) = \{[12, 20[, [22, 23[, [23, 24[, [24, 28[, [28, 32[, [34, 36[\}$
2. $\mathcal{P}(v_{lyrics}) = \{[12, 20[, [22, 23[, [23, 24[, [24, 32[, [34, 36[\}$
3. $\mathcal{P}(v_{bass}) = \{[0, 8[, [8, 12[, [12, 16[, [16, 20[, [20, 24[, [24, 30[, [30, 32[, [32, 36[\}$

This representation can be found, in a more or less complex syntactic form, in common encoding formats, always 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.

3.4. Scores

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

Example 4. *The synchronization of voices v_{sopr} and v_{bass}, restricted to Measure 3, is a voice represented by:*

$$\begin{cases} (D5^{20}_{12}, < B3\flat, D4 >^{16}_{12}), & t \in [12, 16[\\ (D5^{20}_{12}, A3^{20}_{16}), & t \in [16, 20[\\ (\perp, G3^{24}_{20}), & t \in [20, 22[\\ (E5^{23}_{22}, G3^{24}_{20}), & t \in [22, 23[\\ (F5^{24}_{23}, G3^{24}_{20}), & t \in [23, 24[\end{cases}$$

The atomic event $D5^{20}_{12}$ from the `sopr` *voice appears twice, associated with two distinct events of the bass voice, due to the non-homorythmic Conversely, the bass event $G3^{24}_{20}$ appears also in two distinct composed events. Note that the partition of the synchronized voice results from the intersection of $\mathcal{P}(v_{sopr})$ and $\mathcal{P}(v_{bass})$.*

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.*

Example 5. *The score S of our running example, is a tree built as follows:*

- *The vocal part s_{vocal} is a synchronization of v_{sopr} and v_{lyrics}, i.e., $s_{vocal} = v_{sopr} \oplus v_{lyrics}$;*
- *The whole score S is a synchronization of s_{vocal} and v_{bass}, i.e, $S = s_{vocal} \oplus v_{bass}$.*

The type of S is obtained by assigning pairwise distinct labels to the components:

- *The type T_v of s_{vocal} is [*`sopr:` ***dsound****,* `lyrics:` ***dsyll****]*
- *The type T_b of the S 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. Under this perspective S is a function $\mathcal{T} \rightarrow \mathcal{E}(\mathbf{dsound}) \times \mathcal{E}(\mathbf{dsyll}) \times \mathcal{E}(\mathbf{dsound})$. Measure 3 of S for instance is represented as:

$$\begin{cases} (D5^{20}_{12}, \mathrm{Ah}^{20}_{12}, < B3\flat, D4 >^{16}_{12}), & t \in [12, 16[\\ (D5^{20}_{12}, \mathrm{Ah}^{20}_{12}, A3^{20}_{16}), & t \in [16, 20[\\ (\perp, \perp, G3^{24}_{20}), & t \in [20, 22[\\ (E5^{23}_{22}, \mathrm{que}^{23}_{22}, G3^{24}_{20}), & t \in [22, 23[\\ (F5^{24}_{23}, \mathrm{je}^{24}_{23}, G3^{24}_{20}), & t \in [23, 24[\end{cases}$$

Scores are object of a multidimensional, polymorphic space. We can now equip this space with a set of operators that transform those objects: the algebra. Moreover, we will show that this algebra is closed (the result of an operator is always a score), complete (any score S_1 can be transformed in any other score S_2), and minimal (removing any operator would break the completeness property).

4. The score algebra

The score algebra ScoreAlg 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.

4.1. Functions

A *temporal function* is a mapping from $\mathcal{T}$ to $\mathcal{T}$. In the context of music notation, it seems sufficent to 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 $\text{warp}_\text{m} : \mathcal{T} \to \mathcal{T}$, $t \mapsto mt$; and *shifting functions* of the form $\text{shift}_\text{n} : \mathcal{T} \to \mathcal{T}$, $t \mapsto t + n$. 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 $\mathcal{E}(\mathbf{dom}_1) \times \cdots \times \mathcal{E}(\mathbf{dom}_\text{n})$ to a single value in some domain $\mathcal{E}(\mathbf{dom}_\text{o})$. 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 *native functions* of the algebra consists of (i) the linear temporal function and (ii) the internal domain operators.

Example 6. *Here are some examples of applying native 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.

4.2. Structural operators

ScoreAlg $(\oplus, \mu, \sigma, \circ, \text{MAP})$ 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. 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:

$$[S_{sopr} \oplus S_{lyrics}](t) = \begin{cases} (\bot, \bot), & t \in [0, 12[\\ (D5^{20}_{12}, \text{Ah}^{20}_{12}), & t \in [12, 20[\\ (\bot, \bot), & t \in [20, 21[\\ (E5^{22}_{21}, \text{que}^{22}_{21}), & t \in [21, 22[\\ (F5^{23}_{22}, \text{je}^{23}_{22}), & t \in [22, 23[\\ (D5^{28}_{24}, \text{sens}^{32}_{24}), & t \in [24, 28[\\ (C5is^{32}_{28}, \text{sens}^{32}_{24}), & t \in [28, 32[\\ (\bot, \bot), & t \in [32, 34[\\ (A4^{36}_{34}, \text{d'in}^{36}_{34}), & t \in [34, 36[\end{cases}$$

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 $[v_1 : \boldsymbol{dom}_1, \cdots, v_n : \boldsymbol{dom}_n]$, 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).

$$\mu_{sopr,bass}(S(t)) = \begin{cases} (D5^{20}_{12}, < B3es, D4 >^{16}_{12}), & t \in [12, 16[\\ (D5^{20}_{12}, A3^{20}_{16}), & t \in [16, 20[\\ (\bot, G3^{24}_{20}), & t \in [20, 22[\\ (E5^{23}_{22}, G3^{24}_{20}), & t \in [22, 23[\\ (F5^{24}_{23}, G3^{24}_{20}), & t \in [23, 24[\end{cases}$$

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 $T = [v_1 : \boldsymbol{dom}_1, \cdots, v_n : \boldsymbol{dom}_\text{n}]$ and F a Boolean formula on $\mathcal{T}, \boldsymbol{dom}_1, \cdots, \boldsymbol{dom}_\text{n}$, then $\sigma_F(S)$ is a score with type T such that for each voice $S.v_i$:*

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

For instance, still taking our running example and the mono-voice scores of Figure 6.

$$[\sigma_{t\in[12,24[}(S_{sopr})](t) = \begin{cases} D5^{20}_{12}, t \in [12, 20[\\ \perp, \quad t \in [20, 22[\\ E5^{23}_{22}, t \in [22, 23[\\ F5^{24}_{23}, t \in [23, 24[\end{cases}$$

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 $T = [v_1 : \boldsymbol{dom}_1, \cdots, v_n : \boldsymbol{dom}_\text{n}]$ 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:*

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

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.
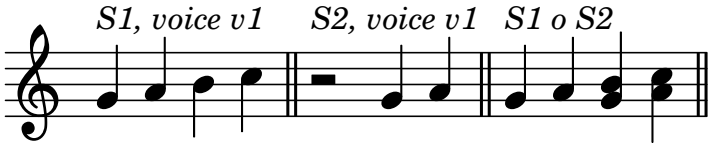


Figure 7: Example of a merge

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. In terms of data representation, we can either choose two disctincts, mono-sound voices, or a single voice with harmonic sounds.

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 $[v_1 : \boldsymbol{dom}_1, \cdots, v_n : \boldsymbol{dom}_n]$, and f a function $\Pi_i^n \boldsymbol{dom}_i \rightarrow \boldsymbol{dom}_o$ then $\text{MAP}_{a:\ f}(S)$ is a score with type [a: $\boldsymbol{dom}_o$] 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.

4.3. Algebraic expressions

An expression is defined in a standard way:

- If S is a score, then S is an expression.

- If E_1 and E_2 are two expressions, then $E_1 \oplus E_2$, $\text{MAP}_{a:\ f}(E_1)$, $E_1 \circ E_2$, $\sigma_F(E_1)$, and $\mu_{v_{i_1},\cdots,v_{i_m}}(E)$ 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].

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

4.4. 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.

5. 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). The objective is to use as much as possible the functionalities of an database system, with a thin extension to integrate our scores and operators.

5.1. System architecture

We begin with an overview of the system architecture to explain the main aspects of this integration (Figure 8). The left part shows the main level involved in the data querying process. At the lowest level, scores are encoded in any of the available format, MusicXML, MEI or even legacy formats such as HumDrum. The only requirements here is the existence of a *mapper* that can decode the serialized score, extract an instance of a gScore, and embed this gScore in a standard XML document. Essentially, score encoding is seen as a low-level, XML-based serialization format, from which structured music content (gScores) can be obtained at run-time.

The intermediate level (left part of Figure 8) consists of XML documents featuring a gScore. We call *opus* such documents. A gScore is embedded in an Opus thanks to a virtual, pseudo-XML representation that allow to see the whole document as a standard XML tree.

The embedding process is detailed in the right part of the figure, taking as example an MEI score. Whenever the Opus needs to be processed, the MEI serialization is mapped toward a

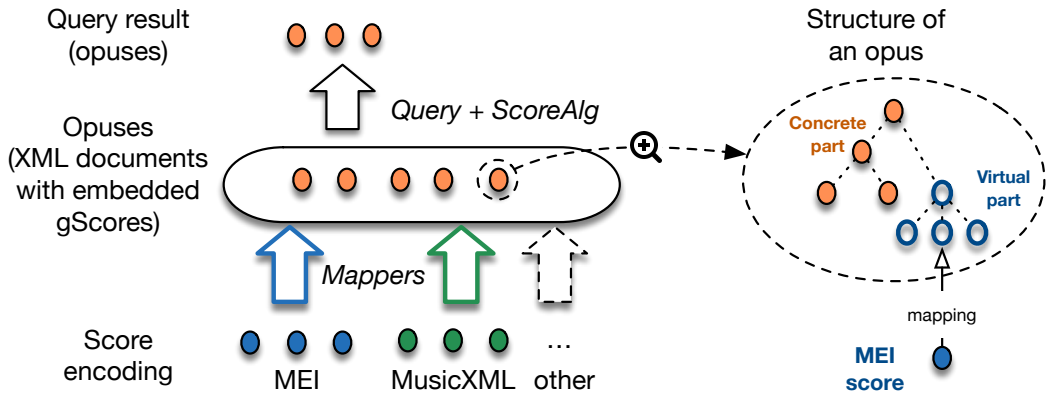


Figure 8: Envisioned system

virtual XML tree (blue border) which is integrated as a subtree of the complete (and concrete) opus XML tree. This allows to incorporate music content, as first-class, queryable objects, in a general-purpose XML database.

We can finally (returning to the left part) express queries on collections of opuses. Standard XQuery operators operate on the concrete part, whereas we apply the ScoreAlg operators, represented as XQuery functions on the virtual gScore(s).

5.2. Opus Modeling

We call *opus* any document featuring music content as part of its description. Recall that a (generalized) score in our model is a tree defined as follows:

- if 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.

In terms of XML representation, this structure is modeled by the following schema.

```
<xs:complexType name="scoreType" abstract="true">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:choice>
      <xs:element type="scoreType"/>
      <xs:element type="voiceType"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

A *score collection* is a set of opuses. Although the definition of a schema for a collection is not mandatory, it is a safe practice to ensure that all the documents it contains share a similar structure. In our context, a collection of opuses is a regular XML collection where one or several elements are of `scoreType` type. Here is a possible example:

```
<xs:complexType name="opusType">
   <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="composer" type="xs:string"/>
      <xs:element name="published" type="xs:string"/>
      <xs:element type="scoreType"/>
   </xs:sequence>
   <xs:attribute type="xs:ID" name="id"/>
</xs:complexType>
```

Note that this schema is strict regarding meta-data (title, composer) but very flexible for the music notation because we are using here the generic `scoreType`. It follows that, from one document to the other in standard notations (musicXML, MEI, HumDrum), the score structure, the number of voices and their identifiers may vary. Such a flexibility is definitely not convenient when it comes to querying a collection, since we cannot safely refer to the components of a score.

It seems more appropriate to base the organization of score collections on an homogeneous score structure. 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.

```
<xs:complexType name="quartetType">
   <xs:complexContent>
      <xs:restriction base="scoreType">
         <xs:sequence>
            <xs:element name="v1" type="soundVoiceType"/>
            <xs:element name="v2" type="soundVoiceType"/>
            <xs:element name="alto" type="soundVoiceType"/>
            <xs:element name="cello" type="soundVoiceType"/>
         </xs:sequence>
      </xs:restriction>
      <xs:attribute type="xs:ID" name="id"/>
   </xs:complexContent>
</xs:complexType>
```

Finally, a voice is a sequence of events. Its abstract definition is given by the following schema:

```
<xs:complexType name="voiceType" abstract="true">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element type="eventType"/>
  </xs:sequence>
</xs:complexType>
```

We can instantiate concrete voice types by simply replacing the abstract `eventType` by one of its derived types (e.g., `soundType`, `syllableType`, `intType`), like in the following example:

```
<xs:complexType name="lyricsType">
 <xs:extension base="voiceType">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
   <xs:element type="syllableType"/>
  </xs:sequence>
 </xs:extension>
</xs:complexType>
```

Enforcing the schema of an opus has two major advantages. First, the content of gScores can be validated, and this ensures that all the opuses of a collection share a common, homogeneous structure. Second, queries can safely address fine-grained components of a gScore, such as a voice referred to by its name, as shown by the following examples.

5.3. Querying Collections of Opuses

XQuery 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.

5.4. Implementation

Our system has been fully implented, integrated in an existing Digital Score Library (NEUMA), and realised as a virtual (Docker) machine at http://hub.docker.com. It relies on off-the-shelf tools, namely a native XML database, BASEX[4] and a music notation library for UDFs, MUSIC21[5][CA10]).

5.4.1. Architecture

The architecture presented in Figure 9 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 semi-virtual instance in the former is linked to a serialized score in the latter.

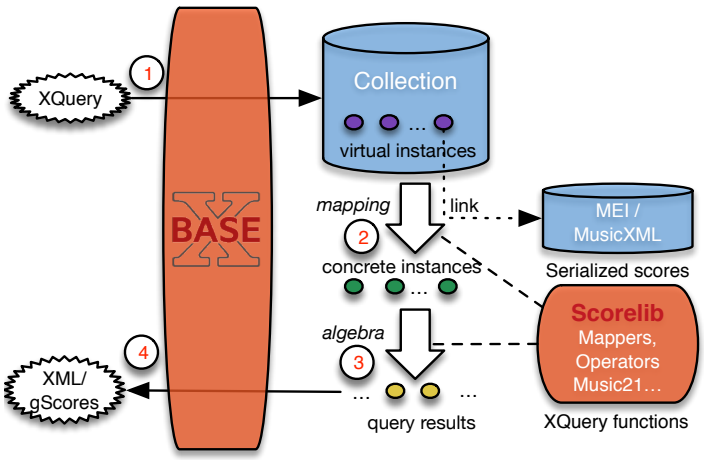


Figure 9: Architecture

The evaluation of a query proceeds as follows. First (step 1), BASEX scans the virtual collection and retrieves the set of opuses 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.

[4]http://basex.org
[5]http://web.mit.edu/music21

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). Incidently, it is worth noting that the whole mechanism behaves like an ActiveXML document, following the design presented in [ABM08] which *activates* the XML content on demand by calling an external service (here, a function).

The ScoreLib Java library is embedded in every BaseX query in order to process our algebraic operations on gScores. The library is responsible for managing the link between the virtual and the physical score, whatever the encoding format. Whenever a function *activates* a gScore by calling a function (whether a structural function of the algebra, or a user-defined function), the link is used to get and materialize the corresponding score.

In order to map scores from the physical representation to the virtual one, references to physical musical parts are matched, according to the collection schema. To achieve this, an ID is associated to each voice element of the materialized score. This ID directly identifies the underlying part of the physical document.

The main mapping challenge is to identify virtual and serialized voices, in particular when they are not standardized according to the collection schema. We need to generate IDs for each parts on the underlying encoding (MusicXML, MEI, etc.), even for monody and lyrics parts which can be merged on the physical document. Most of them can be done automatically with metadata information given by the collection schema. Once a gScore has been materialized, it is kept in a cache in order to avoid repeated and useless applications of the mapping. Temporary gScores produced by applying algebraic operators are kept in the cache as well.

The `Score()` operator creates the final XML document featuring one or several instances of our data model. It combines scores produced by operators and referred to by XQuery variables.

5.4.2. 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
```

5.4.3. Public demonstrator

[Here explain the public interface with Neuma]

5.4.4. Release of eth ScoreQL system

[The Docker image. Its API]

6. Conclusion (à revoir)

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.

- 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 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.

[ABM08] Serge Abiteboul, Omar Benjelloun, and Tova Milo. The active XML project: an overview. *VLDB J.*, 17(5):1019–1040, 2008.

[Aea08] Serge Abiteboul and et al. WebContent: Efficient P2P Warehousing of Web Data. In *VLDB'08 Very Large Data Base*, pages 1428–1431, August 2008.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Bal96] Mira Balaban. The music structures approach to knowledge representation for music processing. *Computer Music Journal*, 20(2):96–111, 1996.

[BBC11] Mohamed-Amine Baazizi, Nicole Bidoit, and Dario Colazzo. Efficient encoding of temporal xml documents. In *International Symposium on Temporal Representation and Reasoning (TIME)*, pages 15–22. IEEE Computer Society, 2011.

[CA10] Michael Scott Cuthbert and Christopher Ariza. Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 637–642, 2010.

[DHI12] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[FBF13] Elie Fares, Jean-Paul Bodeveix, and Mamoun Filali. Event algebra for transition systems composition application to timed automata. In *International Symposium on Temporal Representation and Reasoning (TIME)*, pages 125–132. CPS, 2013.

[FLOB13] Dominique Fober, Stéphane Letz, Yann Orlarey, and Frédéric Bevilacqua. Programming Interactive Music Scores with INScore. In *Sound and Music Computing*, pages 185–190, Stockholm, Sweden, July 2013.

[FOL12] Dominique Fober, Yann Orlarey, and Stéphane Letz. Scores level composition based on the guido music notation. In ICMA, editor, *Proceedings of the International Computer Music Conference*, pages 383–386, 2012.

[FSRT16a] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. Is There a Data Model in Music Notation? In Richard Hoadley, Chris Nash, and Dominique Fober, editors, *Intl. Conf. on Technologies for Music Notation and Representation (TENOR'16)*, pages 85–91. Anglia Ruskin University, 2016.

[FSRT16b] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. Querying Music Notation. In *International Symposium on Temporal Representation and Reasoning (TIME)*, pages 1–9, Kongens Lyngby, Denmark, October 2016.

[FSRT16c] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. Querying XML Score Databases: XQuery is not Enough! In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2016.

[GMUW00] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database System Implementation*. Prentice Hall, 2000.

[Goo01] Michael Good. *MusicXML for Notation and Analysis*, pages 113–124. W. B. Hewlett and E. Selfridge-Field, MIT Press, 2001.

[Hud15] Paul Hudak. *The Haskell School of Music – From Signals to Symphonies*. (Version 2.6), January 2015.

[JBDC+13] David Janin, Florent Berthaut, Myriam Desainte-Catherine, Yann Orlarey, and Sylvain 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)*, pages 23–34, 2013.

[MEI15] Music Encoding Initiative. http://music-encoding.org, 2015. Accessed Oct. 2015.

[QH13] Donya Quick and Paul Hudak. Grammar-based automated music composition in haskell. In *ACM SIGPLAN workshop on Functional art, music, modeling and design*, FARM'13, pages 59–70, 2013.

[RF10] Philippe Rigaux and Zoé 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*, pages 303–320, 2010.

[Rol02] Perry Rolland. The Music Encoding Initiative (MEI). In *Proc. Intl. Conf. on Musical Applications Using XML*, pages 55–59, 2002.

[SF97] Eleanor Selfridge-Field, editor. *Beyond MIDI : The Handbook of Musical Codes*. Cambridge: The MIT Press, 1997.

[TNL07] Nicolas Travers, Tuyêt Trâm Dang Ngoc, and Tianxiao Liu. Tgv: A tree graph view for modeling untyped xquery. In *12th International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 1001–1006. Springer, 2007.

[TWV05] Rainer Typke, Frans Wiering, and Remco C. Veltkamp. A Survey Of Music Information Retrieval Systems. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2005.

[W3C15] W3C Music Notation Community Group. https://www.w3.org/community/music-notation/, 2015. Last accessed Jan. 2016.