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

Raphaël Fournier-S'Niehotta, Philippe Rigaux, Nicolas Travers

▶ **To cite this version:**

Raphaël Fournier-S'Niehotta, Philippe Rigaux, Nicolas Travers. Modeling Music as Synchronized Time Series: Application to Music Score Collections. Information Systems, 2018, 73, pp.35-49. 10.1016/j.is.2017.12.003 . hal-02435620

HAL Id: hal-02435620
https://cnam.hal.science/hal-02435620

Submitted on 11 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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 music representation has a structure that reflects some conceptual principles. This structure is hardly explicitly accessible in many encodings, such as, for instance, audio files. However, it appears much more clearly in the language of *music notation*.

We propose to use the music notation language as a framework to model and manipulate the content of digital music objects, whatever their specific encoding may be. We describe an algebra that relies on this structured music representation to extract, restructure, and search such objects. The data model leverages the hidden structure of digital music encodings to enable powerful manipulations of their content.

We apply the model to collections of music scores. We describe a system, based on an extension of XQuery with our algebra, that provides search, reorganization, and extraction functionalities on top of large collections of XML-encoded digital scores. Beyond its application to music objects, our work shows how one can rely 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, and we can therefore assume that music content has a structure that reflects some conceptual and organisational principles. Digital encoding of music is mostly represented by audio files, in which this structure is blurred and difficult to capture accurately. But music content can also be encoded in a notational form that has been used for centuries to preserve and exchange music works. Music notation appears to be a mature language to represent the discrete, typed, coordinated elements that together constitute the description of complex music pieces.

Figure 1: Beethoven's *Ode to Joy* theme.

The structure of music notation

Fig. 1 shows a first example of a *monophonic score*, the famous "Ode to joy" theme from Beethoven's 9th symphony. The central element is a *note* representing the intended production of a single sound. A note symbol is a black or white dot that carries two essential informations: the *frequency* of the intended sound, measured in Hertz (Hz), and its *duration*, which is a value relative to the *beat*.

Encoding frequencies. The hearing frequency of human beings ranges approximately from 20 to 20,000 Hz. Although this range is in principle continuous, most music compositions rely on a discretization in two steps. First, the frequency range is partitioned in *octaves*: a note a is one octave above another note b if the frequency of a doubles that of b. Octaves are in turn divided in twelve equal *semi-tones*. Since the usual frequency range covers 8 octaves, one obtains $8 * 12 = 96$ possible frequency levels, or *pitches*.

These levels are materialized by horizontal lines on a score. A note can be positionned on a line or between two lines. The whole grid, covering the whole frequency range, would display $96/2 = 48$ lines and take a lot of useless space. Since the usual range of a single musician or singer is much more limited, music scores uses a 5-lines grid, called a *staff*, whose relative position in the complete grid is given by an initial symbol, the *clef* (Fig. 1).

Staves and clefs are example of semiotic artefacts mostly irrelevant in the perspective of using music notation as a language encoding music content. If we leave apart layout concerns, there exists a simple, non graphic convention to encode a music note. Inside an octave, seven pitches are related by close mathematical relationships, and form the Pythagorean scale (commonly called *diatonic scale* nowadays). They are encoded by a letter (A, B, C, ... G). The other pitches can be obtained by either adding (symbol $\sharp$) of substracting (symbol $\flat$) a semi-tone. To summarize: any pitch in common music notation is encoded by (i) a diatonic letter, (ii) an octave in the range 1-8, and (optionally) (iii) a $\sharp$ or a $\flat$ to denote a semi-tone up/down the diatonic frequency. The score of Fig. 1 starts with two E5 (frequency 659.25 Hz), followed by two F5 (frequency 698.46 Hz), two G5 (783.99 Hz), etc.

Encoding beats and durations. Time is discrete, and temporal values are expressed with respect to the *beat*, a pulse that (in principle) remains constant throughough a same piece of music. The *time signature*, a rational (here, 4/4) gives the beat unit, and the temporal organization of the music in groups of beats, or *measures*. In our example, the

denominator states that the beat corresponds to a black note (a quarter of the maximal note duration), and the numerator means that each measure contains four beats. All the possible durations are obtained by applying a simple ratio to the beat: a white note is twice a quarter, a hamped black is half a quarter, etc.

From these (basic) explanations, it follows that music notation can be seen as a way to represent sounds in a 2-dimensional space where each axis (frequencies and durations) is discretized according to some simple rules based on proportional relationships. Moreover, there exists a simple and commonly used encoding to denote each point in this space. We will use this discrete sound domain as a basis of our data model.

Scores as times series. Let us now turn our attention to the *sequence* of notes in Fig. 1. An implicit constraint is that a note starts immediatly after the end of its predecessor. In other words, there is no overlapping of the timespans covered by two distinct notes. 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.

Figure 2: *Ode to Joy*, three instruments.

Polyphonic scores are represented as a combination of several voices. Fig. 2 gives an illustration (the same theme, excerpt of the orchestra parts). In terms of music content, the important information expressed by the notation is the *synchronization* of sounds, graphically expressed by their vertical alignment. The first sound for instance is an harmonic combination of three notes: a C3, a C5, and a E5 (from bottom to top). The two upper notes share the same duration (a quarter), but the bottom one (the bass) is a whole note. This single note is therefore synchronized with 4 notes of the two upper parts.

In such complex scores, one can always adopt two perspectives on the music structure. A vertical perspective, called *harmonic*, focuses on the vertical superposition of sounds, whereas a horizontal one, called *polyphonic*, rather considers the sequential development of each voice. Those two aspects constitute, beyond all the semiotic decorations related to the graphic layout of a score, what could be called the "semantic" of music notation, since they encode all information pertaining to the sounds and their temporal organization (at least for the part of this information conveyed by the notation; some other important

features, such as intensity and timbre, are left to the performer's choice). Together, they define what we will consider in the following as the (structured) *music content*.

Music pieces as synchronized time series

Finally, this modeling perspective can be extended to cover the more general concept of synchronized time series built from arbitrary value domains. Consider our third example shown on Fig. 3, the same *Ode to Joy* enriched with lyrics. The lower staff consists of a single voice, the bass. The upper one is a vocal part which, in our model, consists of two voices, the first one composed of sounds, and the second one of syllables. The latter is an example of a temporal function that, instead of mapping timestamps to sounds, maps timestamps to syllables.

Figure 3: *Ode to Joy*, music and lyrics.

Position, goals and contributions

The position adopted in the present paper relies on two ideas. Firstly, music notation is a proven, sophisticated, powerful formal language that provides the basis of a data model for music content. Secondly, instances of this model can be extracted from digital music documents, and this extraction yields a structured representation of this object through which its content can be inspected, decomposed, transformed and combined with other contents. In the context of large collections of such music objects, this opens perspectives for advanced search, indexing and data manipulation mechanisms.

We can therefore envision a modeling that abstracts the music content as a synchronization (harmonic view, expressed by the vertical axis in the score representation) of temporal sequences of acoustic events (polyphonic view, expressed by the horizontal axis). As a natural generalization of this modeling approach, we accept polymorphic events that can either represent sounds, or features that make sense as time-dependent information synchronized with the music content. Fig. 4 summarizes the envisioned system. The bottom layer is a Digital Music Library (DML) managing music objects in some encoding, whether audio (WAV, MP3, MIDI), image (PDF, PNG), or XML (MusicXML, MEI). 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.

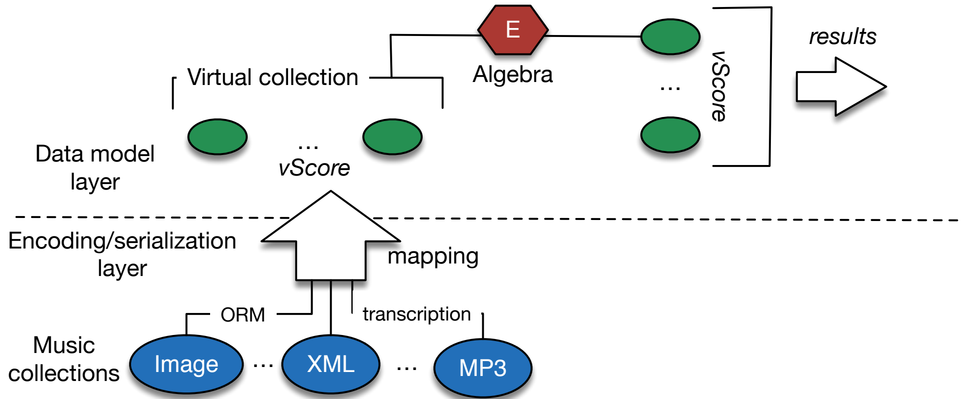


Figure 4: Envisioned system

This automatic mapping is called *transcription* for audio files, *optical music recognition* (OMR) for images, and is a much simpler extraction for XML formats. It defines a virtual – no materialization occurs – abstraction of structured music objects that we will call *virtual music score* (vScore) 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 *vScores* from the base corpora. The present paper develops this approach and proposes the following contributions:

- The definition of a *data model* to represent the core structure of music content, guided by the music notation concepts.
- An *algebraic language*, consisting in a compact, minimal set of operations that apply to the model domain in closed form.
- Finally, an application to collections of digital music scores, encoded in XML. We describe a system that converts on-demand XML scores to instances of our model and extends XQuery with our algebra, thereby allowing to express user queries over collections of music scores.

The work presented here has been prepared by conference talks and short publications, addressed to the most relevant communities (music information retrieval and temporal data management), that helped us in testing and confirming the validity of our approach . The motivation for extracting a data model from music notation encodings is discussed in [FSRT16a]. The formal model for synchronized time series is presented in [**?**], and the principles of its interface with XQuery in [FSRT16b]. The current text details the modeling approach, positions our work with respect to the state-of-the art, clarifies the algebra with examples. Moreover, we now present a full implementation, describe its architecture and propose optimization guidelines. This system, resulting from a strong implementation

effort, integrated in a Digital Score Library at http://neuma.huma-num.fr, is also publicly available as a virtual Docker image, ready to be used in any similar environment.

To the best of our knowledge, this constitutes the first attempt to address modeling and querying issues on collections of music content with a data management perspective. Whatever their origin, transcription, optical recognition, or direct encoding, large collections of digital music are being produced. They will probably result in a near future in publicly available repositories supplying an open access to music heritage, in the wake of the Open Data movement. We believe that music-aware data management methods will then be of great help, just as research results on spatial data handling helped to leverage the open access to formerly proprietary geographical data, one decade ago.

Section 2 reviews the state-of-the-art. We present the data model in Section 3 and the algebra in Section 4. In Section 5, we describe a music score management system, along with query examples and implementation guidelines. We finally conclude the paper in Section 6.

2. Context and State of the art

Music representation. There exists essentially two different digital representations of music: audio and notation. The most common representation, by far, is sampled audio. The audio signal is sampled (44.1K samples per second in CD-quality formats), each sample giving an amplitude (loudness) and frequency. Formats that faithfully represent this information, such as the Waveform Audio File (WAV[1]), present huge storage requirements. Compressed formats (e.g., MP3[2], AAC [3]) achieve a compression of one order of magnitude by getting rid of marginal frequencies.

Audio files encode an audio signal for sound reproduction purposes, and no attempt to identify the music structure (such as for instance separating instruments and voices) is made. Without further analysis, no "symbolic" manipulation of the content is possible (e.g., replace instruments, or isolate a voice).

Another common format is the Musical Instrument Digital Interface (MIDI). As its name implies, a MIDI document represents the input or output of an electronic sound device (a digital keyboard for instance), and the synchronization of several outputs (or channels). Unlike audio files, the representation is *symbolic*: the intended sound is not encoded as a signal but as a set of discrete events described by many parameters: pitches, instrument, onsets/offsets, among others.

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

Music can also be represented via the traditional music notation language as "sheet music", or simply, music scores. The notation of music has been the primary mean to preserve and disseminate music pieces for centuries, at least in the Western area. It proposes a sophisticated semiotic system able to convey highly complex successions and combinations of sounds played simultaneously by several instruments or singers. This language has proved its flexibility and representational power throughout the ages, and its ability to adapt to a wide range of styles and forms. It constitutes therefore a language of choice to describe a piece of music, its structure (parts, voices) and the detailed content of each component (pitches, rhythms).

Since the beginning of this century, two XML-based encodings of music notation have been proposed: MusicXML [Goo01] and MEI [Rol02, MEI15]. The W3C recently launched a normalization endeavor [**?**], 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 to future large collections of scores.

XML representations for music scores are 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 (positioning of the score objects). The rendering of a score, as shown by Figs. 1, 2 and 3, 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. To put it differently, the music content is tangled with a lot of irrelevant information, which makes XML formats unsuitable for complex data manipulations as is.

Digital Music Collections. Large collections of music pieces are nowadays available online. Audio is still the prominent format, and the process of extracting a MIDI or notational representation from audio files is known as *music transcription* [Kla04, BDG+13, USG14]. It relies of signal analysis algorithms that decode acoustic flows to identify frequencies, tones, rhythms and attempt to produce a symbolic notation. Among many issues, the separation of voices (e.g., instruments) still does not achieve full reliability. The field is however mature enough and several commercial applications produce digital notation from audio files, e.g., *audioScore-ultimate*[4], *AnthemScore*[5] or *Widisoft*[6].

Collections of music pieces can also commonly be found as images (PDF). The IMSLP Petrucci Music Library (http://imslp.org) for instance contains, at the time of writing, more than 120K complete music pieces. A notational description of these pieces can be obtained thanks to an Optical Music Recognition (OMR) software. The results are however highly dependent on the image quality and sometimes fail to supply an accurate description

[4]https://www.avid.com/products/audioscore-ultimate
[5]https://www.lunaverus.com/
[6]http://www.widisoft.com/index.html

of the score [BB01, RFP+12].

There is an increasing production and publication of scores directly encoded in one of the two main XML formats, MusicXML or MEI. They are created and maintained by librarians, researchers, publishers and active communities of users keen to make publicly accessible the priceless heritage of music that has been preserved for centuries via their notation. The OpenScore initiative for instance (https://www.openscore.cc/) is an ongoing crowdsourcing endeavour to encode the content of the IMSLP library, and will eventually result in large collection of high-quality score encodings.

Music information retrieval. Most of the research conducted in the *Music Information Retrieval* domain so far has focused on unstructured search and similarity measures, see [TWV05, CVG+08, SGU14] for surveys 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. Finding a good similarity measure for highly diverse and complex objects is a difficult task (see [Bel11, SYBK16] for some recent proposals).

The work presented in this article introduces operators to manipulate the internal musical content of the document at a very fine-grained level (e.g., voices) and avoids to rely on fragile similarity evaluations altogether. It can be related to formal languages for music content representation and manipulation, notably Euterpea, introduced by [Hud15], and a few others [Bal96, JBDC+13, FLOB13]. Their focus are a modeling of music language for music generation, rather than managing a persistent collection of scores. To some extent our goal is **a language which is the equivalent for music notation of the relational algebra**: a limited number of types, a minimal set of operators that can be composed, and a well-defined expressive power.

Time series (TS) are intensively studied objects, but most often in a data mining perspective (prediction) which is quite different from ours which only considers static objects whose internal description is made of fixed-length TS. Some relevant works are [BBC11] and [FBF13]. 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 inappropriate to score manipulations.

Our design involves a mapping that extracts from the raw music encoding a structured model instance. This mapping gets rid of rendering aspects in the case of XML-encoded scores for instance. This is reminiscent from mediation architectures used for data sources integration (see [GMUW00, DHI12, AAC+08, TNL07]), and can be seen as an application of methods that combines queries on physical and virtual instances. It borrows ideas from ActiveXML [ABM08], and in particular the definition of some elements as "triggers" that

activate external calls.

Time series representation is a common way to model Music Information [ABSW04]. Amongst query languages based on time-series, we can cite AQuery [LS03, LSW+04] which proposes to extend SQL to time series represented as sorted relations. Even if the data model is dedicated to relational representation (tabular), the semantic of sequences with order dependency and a detailed algebra gives has also inspired our data model and algebra for temporal partitions. However, the language and algebra are more dedicated to time aggregations than time transformations.

Since modern score formats adopt an XML-based serialization, XQuery [XQu07] has been considered as the language of choice for score manipulation [GSD08]. THoTH [Whe15] also proposes to query MusicXML with patterns analysis. We believe that a pure XQuery approach is too generic to handle the specifics of music representation. Score encodings closely mix information related to the *content* and to a specific *rendering* of this content. This leads to an intricate encoding from which selecting the relevant information becomes extremely difficult.

3. Music Content Modeling

This section presents our data model. We use as a running example the score fragment of Fig. 5. It shows a score with two parts. The lower one, called the "bass" in the following, consists of a single voice. The upper one is a vocal part which, in our model, consists of two voices, the first one (the "soprano") composed of sounds, and the second one (the "lyrics") of syllables. Note that there is no one-to-one rhythmic correspondence between syllables and notes, as some syllables cover several notes.

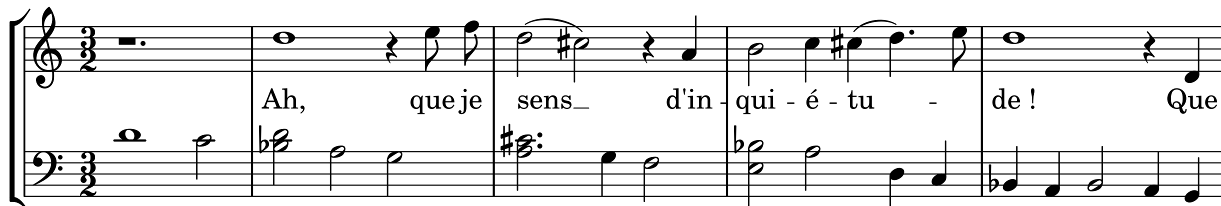


Figure 5: Running example (French air, http://neuma.huma-num.fr/home/opus/timbres:airsmercure)

3.1. Time

Music notations organizes temporal features (onsets and durations) 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 $\mathbb{Q}$. However, if we consider a single piece of music, there exists a maximal level of decomposition of the beat, and this level

defines the time unit. In our running example (Fig. 5), the time signature (3/2) indicates that the beat is a half note (hamped white symbol), with three beats per measure. Since a beat is divided by four at most, the time unit is an eighth note, and each measure consists of exactly 12 units. Measure 1 extends from timestamp 0 to 11, measure 2 from timestamp 12 to 23, etc.

Recall from the introduction that a sequence of notes is a time series of musical events, assigned to time ranges that do not overlap with one another. This defines a *temporal partition* $\mathcal{P} = \{I_1, \cdots, I_n\}$ as a set of right-open time intervals $I_i = [t_1^i, t_2^i[$ such that $\forall i \neq j \leq n, I_i \cap I_j = \emptyset$. Note that, since a music piece covers a finite temporal range and may contain "holes" (rests), we do not impose the usual coverage condition $\bigcup_i I_i = \mathcal{T}$. Each voice in the horizontal, polyphonic perspective, defines a temporal partition. In order to combine several voices in the harmonic perspective, 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* $I_i = J_j$.

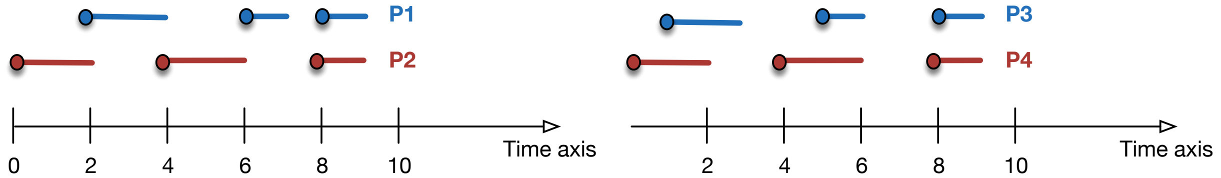


Figure 6: Merge-compatible (left) and non-compatible (right) temporal partitions

For instance, $P_1 = \{[0, 2[, [4, 6[, [8, 9[\}$ and $P_2 = \{[2, 4[, [6, 7[, [8, 9[\}$ are merge-compatible. Their union:

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

is a temporal partition such that each interval can be found in either P_1 or P_2 (Fig. 6, left). P_3 and P_4 (Fig. 6, right), on the other hand, are not merge-compatible, since some intervals overlap, or are contained in one another. The concept will prove useful for the combination of scores.

3.2. Values

Our model is built on sets of constants, called *domains*. The main one, denoted **dpitch**, is the set of atomic sounds. Sound is a complex notion that can be decomposed in several

components (frequency, intensity, timbre). In our context, we are here limited to those captured by the notational system, i.e., essentially the frequency. As explained in the Introduction, most of music pieces (and at least those that can be represented by music notation) are based on less than 100 music frequencies, and can be encoded by a diatonic letter ranging from A to G, an octave ranging from 1 to 8, and an optional accidental $\sharp$ or $\flat$. The bass voice of our running example starts with a D4, followed by a C4. Both correspond to atomic frequencies. The next value met in the bass voice however combines two atomic sounds: a B$\flat$3 and a D4. The simultaneous production of two distinct sounds, *for the same duration*, constitutes what will be called an *harmonic sound*, or chord, in the following. There is no *a priori* limitation on the number of sounds that can be combined, and we therefore define the domain **dsound** as the union of **dpitch**, $\mathbf{dpitch}^2$, $\mathbf{dpitch}^3$, $\cdots$, $\mathbf{dpitch}^{\mathrm{i}}$, $i \in \mathbb{N}$. The third value of the bass voice belongs to $\mathbf{dpitch}^2$ and will be noted $< B\flat 3, D4 >$.

Each domain comes with a set of core operators. For **dsound**, we consider

- the harmonic operator, Ξ, creates an harmonic sound from two harmonic sounds; for instance, $B\flat 3 \;\Xi\; D4 =< B\flat 3, D4 >$
- the transposition operator, $\updownarrow$, moves a frequency by a fixed number of semi-tones; for instance, $C4 \updownarrow 2 = D4$

dsound is the essential domain by which we can describe music but, as shown by our examples so far, we also need the lyrics part which is a first-class component of music content modeling. The domain, **dsyll** is the set units of pronunciation, or syllables. The only required operator is the concatenation, $||$.

3.3. 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 $e = a_{t_1}^{t_2}, a \in \boldsymbol{dom}, \mathrm{t}_1, \mathrm{t}_2 \in \mathcal{T}, \mathrm{t}_1 < \mathrm{t}_2$*, represents the fact that value a is observed from t_1 (included) to t_2 (excluded). Interval $[t_1, t_2[$ is the* range *of the event, and we note $\mathcal{E}(\boldsymbol{dom})$ the set of events on **dom**.*

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. The value is a $D5 \in$ **dsound** and it extends from timestamp 12 (beginning of second measure) to timestamp 20 (excluded).*

- *'Ah'* $^{20}_{12}$ *is a (syllabic) event. The value is a syllable Ah* $\in$ ***dsyll****, and it extends over the same time period.*
- $< B\flat 3, D4 >^{16}_{12}$ *is a* ***dsound*** *event (a chord), in the lower voice, over [12,16[.*

We do not restrict the events to musical domains. 2^{16}_{12} 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, we can build generalized scores that extend the usual concept by combining musical events with non-musical domains representing, for instance, some analytic feature.

3.4. Voice

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

Figure 7: Voices as time series for our example (measures 1 to 3 of Fig. 5)

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*

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

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 $v(t) = \emptyset$, where $\emptyset$ somewhat represents the absence of an event (rests).

Remark 1. *In order to simplify the notation, we will assimilate a voice in* ***dom*** *to be a* total *function from* $\mathcal{T}$ *to* $\mathcal{E}(\boldsymbol{dom} \cup \emptyset)$.

It follows that the set of intervals of the events in E_v defines a temporal partition of $\mathcal{T}$, denoted $\mathcal{P}(v)$, and called in the following the *active temporal domain* of v.

Example 2. *Fig. 7 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. $\mathcal{P}(v_{bass}) = \{[0, 8[, [8, 12[, [12, 16[, [16, 20[, [20, 24[, [24, 30[, [30, 32[, [32, 36[\}$

The voice concept is the core structure of a music piece description, and a voice is the basic object that will be manipulated by our algebraic operators, to be presented next. Any mapping that produces instances of our model from a digital music representation should therefore be able to extract the voices. This is almost straightforward for XML-based encoding formats, where the mapping simply gets rid of notational elements representing rendering instructions and performance directives. This is obviously more difficult for audio files (transcription) or score images (optical recognition). However, in all cases, our modelling can be seen as a proposal to abstract, from the digital music encoding, an information set that focuses on the music content structure.

3.5. Scores

Voices can be *synchronized*. The synchronization of two voices v_1 in $\textbf{Voice}(\texttt{dom}_1)$ and v_2 in $\textbf{Voice}(\texttt{dom}_2)$, denoted $v_1 \oplus v_2$, is the voice v_3 in $\textbf{Voice}(\textbf{dom}_1 \times \textbf{dom}_2)$ 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:

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

Obviously, since v_1 and v_2 are functions, so is v_3. Moreover, if $v_3(t) = (e_1, e_2)$, with $e_1 = a_{t_1^1}^{t_2^1}$ and $e_2 = b_{t_1^2}^{t_2^2}$, the range of the composite event (e_1, e_2) is $[t_1^1, t_2^1[\cap [t_1^2, t_2^2[$. Thus, v_3 is a voice unambiguously defined in a space of composite events $\mathcal{E}(\textbf{dom}_1) \times \mathcal{E}(\textbf{dom}_2)$, and such that $\mathcal{P}(v_3) = \mathcal{P}(v_1) \cap \mathcal{P}(v_2)$.

Example 3. *The synchronization of voices v_{sopr} and v_{bass} is illustrated by Fig. 8 (first three measures). Let us focus on Measure 2 and detail its content. It is a voice represented by:*

$$\begin{cases} (D5_{12}^{20}, < B\flat 3, D4 >_{12}^{16}), & t \in [12, 16[\\ (D5_{12}^{20}, A3_{16}^{20}), & t \in [16, 20[\\ (\emptyset, G3_{20}^{24}), & t \in [20, 22[\\ (E5_{22}^{23}, G3_{20}^{24}), & t \in [22, 23[\\ (F5_{23}^{24}, G3_{20}^{24}), & t \in [23, 24[\end{cases}$$

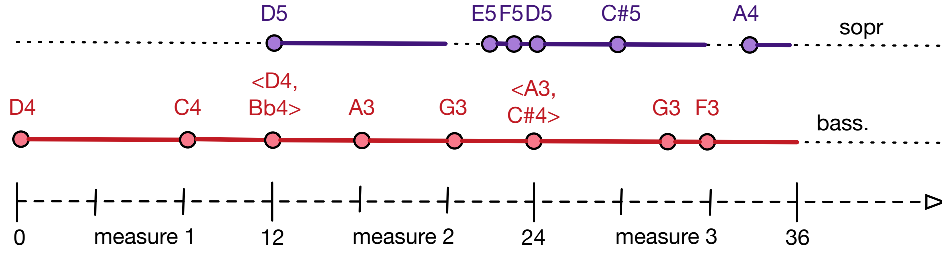


Figure 8: Voice synchronization for the running example (measures 1-3)

The atomic event $D5_{12}^{20}$ from the **sopr** *voice appears twice, associated with two distinct events of the bass voice, due to the non-homorythmic synchronization. Conversely, the bass event $G3_{20}^{24}$ appears also in three distinct composed events. The active domain of the synchronized voice results from the intersection of $\mathcal{P}(v_{sopr})$ and $\mathcal{P}(v_{bass})$.*

We can now define (abstract) 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 4. *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`: Voice(**dsound**), `lyrics`: Voice(**dsyll**)]*
- *The type T_b of the S is [`vocal`: T_v, `bass`: Voice(**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 $\mathcal{T} \rightarrow \mathcal{E}(\textbf{dsound}) \times \mathcal{E}(\textbf{dsyll}) \times \mathcal{E}(\textbf{dsound})$.

Example 5. *S on measure 3 is the function from $\mathcal{T}$ to $\mathcal{T} \rightarrow \mathcal{E}(\boldsymbol{dsound}) \times \mathcal{E}(\boldsymbol{dsyll}) \times \mathcal{E}(\boldsymbol{dsound})$ defined as*

$$S(t) = \begin{cases} (D5_{12}^{20}, Ah_{12}^{20}, < B\flat 3, D4 >_{12}^{16}), & t \in [12, 16[\\ (D5_{12}^{20}, Ah_{12}^{20}, A3_{16}^{20}), & t \in [16, 20[\\ (\emptyset, \emptyset, G3_{20}^{24}), & t \in [20, 22[\\ (E5_{22}^{23}, que_{22}^{23}, G3_{20}^{24}), & t \in [22, 23[\\ (F5_{23}^{24}, je_{23}^{24}, G3_{20}^{24}), & t \in [23, 24[\end{cases}$$

Scores are objects of a multidimensional, polymorphic space. We can now equip this space with a set of operators: 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 sufficient 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 $\mathrm{warp_m} : \mathcal{T} \rightarrow \mathcal{T},\ t \mapsto mt$; and *shifting functions* of the form $\mathrm{shift_n} : \mathcal{T} \rightarrow \mathcal{T},\ t \mapsto t + n$. 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 domain $\mathbf{dom}_1 \times \cdots \times \mathbf{dom}_\mathrm{n}$ to a single value in some domain $\mathbf{dom}_\mathrm{o}$. We extend the interpretation of temporal and domain functions to events naturally. If $e = a_{t_1}^{t_2}$ is an event on **dom**, then:

1. (Temporal function) $\tau_{m,n}(e) = a_{\tau_{m,n}(t_1)}^{\tau_{m,n}(t_2)}$
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 functions and (ii) the internal domain operators.

Example 6. *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)*
- $\updownarrow(e, 2) = E5_{12}^{20}$ *(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

Our algebra ScoreAlg $(\oplus, \pi, \sigma, \circ, \text{MAP})$ 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 2. *For the sake of simplicity, the definition of operators is given on "flat" scores. Their extension to the full recursive structure is trivial.*

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 labels in S_1 and S_2 are not fully distinct, a renaming is necessary. The operator ρ (not presented here) is similar to the relational renaming operator.

Example 7. *The synchronization of S_{sopr} and S_{lyrics} is:*

$$[S_{sopr} \oplus S_{lyrics}](t) = \begin{cases} (D5_{12}^{20}, Ah_{12}^{20}), & t \in [12, 20[\\ (\emptyset, \emptyset), & t \in [20, 21[\\ (E5_{21}^{22}, que_{21}^{22}), & t \in [21, 22[\\ (F5_{22}^{23}, je_{22}^{23}), & t \in [22, 23[\\ (D5_{24}^{28}, sens_{24}^{32}), & t \in [24, 28[\\ (C\sharp5_{28}^{32}, sens_{24}^{32}), & t \in [28, 32[\\ (A4_{34}^{36}, d'in_{34}^{36}), & t \in [34, 36[\end{cases}$$

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 $[v_1 : \boldsymbol{dom}_1, \cdots, v_n : \boldsymbol{dom}_{\rm n}]$, then $\mu_{v_{i_1}, \cdots, v_{i_m}}(S), \forall i_j, j \in [1, n]$ is a score defined as:*

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

Example 8. *The following example shows the score obtained by projecting out the lyrics voice (measure 3).*

$$\mu_{sopr,bass}(S(t)) = \begin{cases} (D5_{12}^{20}, < B\flat 3, D4 >_{12}^{16}), t \in [12, 16[\\ (D5_{12}^{20}, A3_{16}^{20}), \qquad t \in [16, 20[\\ (\emptyset, G3_{20}^{24}), \qquad t \in [20, 22[\\ (E5_{22}^{23}, G3_{20}^{24}), \qquad t \in [22, 23[\\ (F5_{23}^{24}, G3_{20}^{24}), \qquad t \in [23, 24[\end{cases}$$

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

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

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

For instance, still taking our running example and the mono-voice scores of Fig. 7.

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

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.

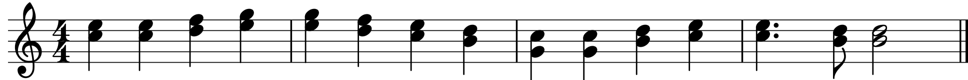
Definition 8 (Merge, $\circ$). *If S_1 and S_2 are two scores with type $T = [v_1 : \boldsymbol{dom}_1, \cdots, v_n : \boldsymbol{dom}_n]$ such that the active temporal domain $\mathcal{P}(S_1.v_i)$ is merge-compatible with $\mathcal{P}(S_2.v_i), \forall i \in [1, n]$, then $S_1 \circ S_2$ is a score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

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

Let us turn back to the *Ode to Joy*. Fig. 9 shows (top part) the soprano voice (left) and the tenor voice (right). They are obviously merge-compatible, showing a strict pairwise correspondence of their respective notes.

Ode to joy, the soprano voice

Ode to joy, the tenor voice



Ode to joy, after merging the soprano and tenor voices

Figure 9: Illustration of the merge operator

The bottom part of the figure shows the result of the merge. It consists of a single voice with harmonic sounds in $\mathbf{dsound}^2$. It is interesting to note that one obtains exactly the same content (i.e., the same global set of events) by synchronizing the voices instead of merging. Both operations result in a different syntactic way of representing this music object: as a voice with complex sounds in the first case (merge), or as two voices with atomic sounds. Recall that there exists two perspectives to explore a music score. The harmonic point of view models the music as a sequence of simultaneous sounds, whereas the polyphonic point of view focuses on independent voices. The merge compatibility property characterizes the situation where the two perspectives coincide.

Property 1. *Let v_1 and v_2 be two voices such that their active temporal domain are merge compatible. Then:*

$$v_1 \oplus v_2 \equiv v_1 \circ v_2$$

where ≡ expresses the equivalence of music content.

In practice, this means for instance that the two notations (synchronized or merged) can be used equivalently to represent the same music content.

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

Example 9. *Translating a mono-voice score 12 temporal units to the right is expressed by:*

$$\text{MAP}_{r:\ \tau_{0,12}}(S)$$

Transposing a mono-voice score 5 ***dsound*** *units up is expressed as:*

$$\text{MAP}_{t:\ \updownarrow 5}(S)$$

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 types.

The following property is immediate from the definitions.

Theorem 1 (Correctness). *Let S be a score and E a well-typed expression, then $E(S)$ is a score.*

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

Theorem 2 (Completeness). *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. We first show that, given an elementary score S_0 with a unique voice V_0 and a single event $\perp_0^1$, where $\perp$ denotes the lowest possible pitch in the **dsound** domain, we can generate S_1.

Let v_1 be a voice in S_1, and e be some event in v_1 of the form $< a_1, \cdots a_n >_{t_1}^{t_2}$ for some $n > 0$. The following expression creates, from S_0, a voice v_e^1 with a single event ${a_1}_{t_1}^{t_2}$.

$$\text{MAP}_{v_e^1\text{: } \updownarrow_{a_1}}(\text{MAP}_{y\text{: shift}_{t_1}}(\text{MAP}_{x\text{: warp}_{t_2-t_1}}(S_0)))$$

When applied to the $\perp_0^1$ event, this expression successively produces $\perp_0^{t_2-t_1}$, $\perp_{t_1}^{t_2}$, and ${a_1}_{t_1}^{t_2}$. Similarly, we can generate voice $v_e^2, \cdots, v_e^n$ with the respective events ${a_2}_{t_1}^{t_2}$, $\cdots$, ${a_n}_{t_1}^{t_2}$. All these voices are merge-compatible, and merging them

$$v_e^1 \circ v_e^2 \circ \cdots \circ v_e^n$$

yields a voice v_e with event e. We can similarly generate a voice $v_{e'}$ for any other events e' of v_1. Since the ranges of e and e' do not overlap, v_e and $v_{e'}$ can be merged. Merging all these voices for all the events of v_1 produces v_1. The process can be repeated to generate all the voices $v_2, \cdots, v_m$ in S_1, and the synchronization $v_1 \oplus v_2 \oplus \cdots \oplus v_m$ produces S_1. Therefore, there exists an expression E_l such that $S_1 = E_l(S_0)$.

Conversely, we can show that, from any score S, we can produce an algebraic expression that yields S_0. Indeed, let v be a voice in $S_@$, and e be some event in v of the form $< a_1, \cdots a_n >_{t_1}^{t_2}$ for some $n > 0$ The following expression produces S_0 from S_2.

$$\text{MAP}_{V_0\text{: } \updownarrow_{-a_1}}(\text{MAP}_{y\text{: warp}_{\frac{1}{t_2-t_1}}}(\text{MAP}_{x\text{: shift}_{-t_1}}(\sigma_{pitch=a1}(\pi_v(S)))))$$

Therefore there exists E_q such that $E_q(S_2) = S_0$. 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 scores (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: The ScoreQL system

We now expose how our algebra can be used as a core component of a querying system for Digital Score Libraries (DSLs). We implemented such a system for the Neuma DSL, located at http://neuma.huma-num.fr. The following section focuses on architectural issues, and discusses implementation choices.

5.1. System architecture

The objective is to use as much as possible the functionalities of an existing database system, with a lightweight extension to integrate scores and operators. We begin with an overview of the system architecture to explain the main aspects of this integration (Fig. 10).

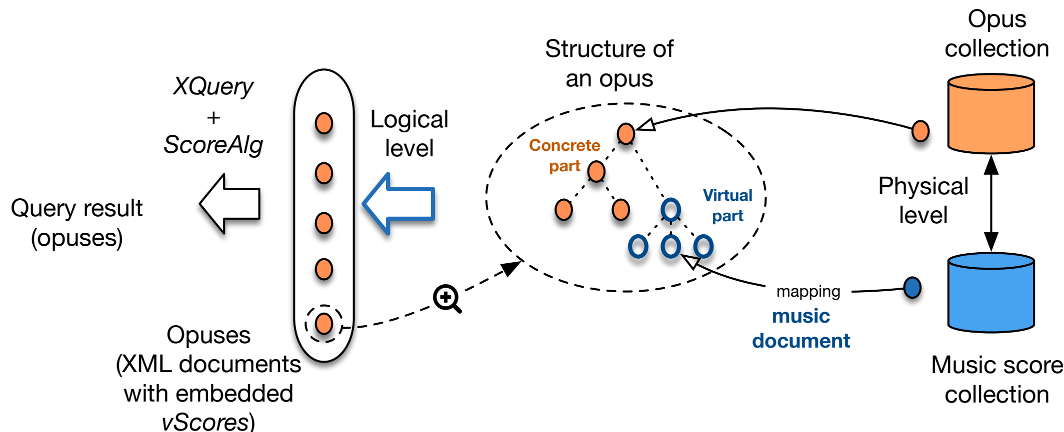


Figure 10: Conceptual architecture

From the user point of view, the system is a general-purpose XML database, and the query language is essentially XQuery. What makes the queryable collections particular is that their documents, called *opus* in the following, contain one or several *vScore* XML subtrees. Such vScores are XML encodings of our data model, and can be manipulated either directly with XQuery operators, or more conveniently by applying our algebraic operators, represented in the XQuery space by functions.

A vScore is virtual, and is actually instanciated one the fly at query evaluation time. At the physical level, indeed, opuses are represented as standard concrete XML documents featuring score links referring to score document, encoded in MusicXML or MEI. Whenever required, a mapping takes the score document, gets rid of useless information and exposes its music content, in XML form, as an instance of a *vScore* embedded in the opus tree. To put it differently, existing encodings are seen as a low-level, physical format, from which structured music content can be obtained at run-time.

A *music collection* is a regular XML collection containing documents where one or several elements are instances of a `scoreType` type. Here is an example of a possible Opus schema.

```
<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>
```

According to Definition 4, a vScore is a tree of either sub-vScores or voices. This is modeled in XML 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>
```

The above schema is strict regarding meta-data (title, composer) but very flexible for the music notation, due to the generic `scoreType` definition. It follows that, from one document to the other, the number of voices and their identifiers may vary. The subtyping mechanism of XML Schema makes it easy to define more constrained types. Here is for instance the schema of a *Choral* 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="soprano" type="soundVoiceType"/>
            <xs:element name="alto" type="soundVoiceType"/>
            <xs:element name="tenor" type="soundVoiceType"/>
            <xs:element name="voices" type="soundVoiceType"/>
```

```
        </xs:sequence>
      </xs:restriction>
      <xs:attribute type="xs:ID" name="id"/>
   </xs:complexContent>
</xs:complexType>
```

Enforcing the schema of an opus ensures that all the opuses of a collection share a common, homogeneous structure, and that queries can safely address fine-grained components of a vScore, such as a voice referred to by its name.

We could develop further the data model by introducing XML representation for events, and specify the content of a voice instance as a sequence of events. However, we chose to limit the granularity exposed to the user to voices, the content of which is a black box queryable only by applying algebraic functions. This is not really necessary, since our algebra is designed to operate on voices and, as shown in the previous section, supplies all the possible manipulations that can be applied to voices. The following section illustrates the resulting XQuery-based facility with some representative examples.

5.2. Querying Music Collections

The following examples are based on a collection of Bach's chorales. Each opus consists of four voices, respectively named *soprano*, *alto*, *tenor* and *bass*, matching the `Choral` XML Schema given in the previous section[7].

Our first example extracts the first 5 measures of the soprano voice for all the chorales. Creating such a collection of *incipits* is quite useful to obtain at a glance a summary of the collection (in this specific case, the first measures of soprano voice is usually sufficient to identify a single choral). The XQuery expression is as follows:

```
for $o in collection("Chorals")/opus
let $incipit := scoreql:select ($o/score/soprano, "measure() in [1,5]")
return <result>
            <title>{$o/title}</title>
            <incipit>{scoreql:eval($incipit)}</incipit>
       </result>
```

Query 1: Metadata filtering, Select and Projection operators

[7]The full collection is accessible on-line at http://neuma.huma-num.fr/home/corpus/composers:bach:chorals/

Let us first explain the main aspects of the integration of XQuery and ScoreAlg. The latter is represented by functions in the `scoreql` namespace, with the exception of the projection operator π which is conveniently expressed by XPath. Thus, `$o/score/soprano` corresponds to $\pi_{\text{soprano}}(\$o.score)$. The selection operator `select ($o/score/soprano, "measure() in [1,5]")` is an example of an algebraic function call. The *select* operator (Definition 7) takes as input a vScore, a Boolean expression e, and filters out the events that do not satisfy e, replacing them by a `null` event. The selection formula is sent as a string since it does not belong the XQuery scope. Note that this is different from *selecting* a score based on some property of its voice(s), an operation which can be achieved with standard XQuery expressions such as *highest()* in Query 2.

ScoreAlg expressions are evaluated lazily. This allows to decompose a complex expression in steps, assigned to XQuery variables, and to refer to the same variable several times without incurring multiple evaluations. Function *eval()* triggers the query evaluation.

A common operation is to *transpose* a score, if for instance one of the voice is too high or too low for the singer. The following query transposes down the soprano and bass voices by 2 semi-tones, if the soprano goes up higher than the F5.[8]

```
for $o in collection("Chorals")/opus
where scoreql:highest($o/score/soprano) > scoreql:frequency("F5")
let $transpS := scoreql:map ($o/score/soprano, "transpose (-2)")
let $transpB := scoreql:map ($o/score/bass, "transpose (-2)")
return   scoreql:eval (scoreql:sync ($transpS, $transpB))
```

Query 2: Synchronize, Map & User Defined Function

This second query shows also how to define variables that hold new content derived from vScores via *UDFs*. Function *highest()* that takes a score as input and returns a value (here, the highest note in the soprano voice). Operator `map` applies the transposition, whereas operator `sync` (Def. 5) creates the resulting score from transposed voices. Map is 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.

Our system may also take several vScores as input and produce a document with several vScores as output. Assume for instance that we wish to compare two chorals by inspecting pairwise combinations of their voices. The following example takes those two chorals as input, and produces two vScores, synchronizing respectively the alto and tenor voices.

[8]The whole choral should be transposed with the same mechanism.

```
for $o1 in coll("Chorals")/opus[@id="BWV250"],
    $o2 in coll("Chorals")/opus[@id="BWV320"]
return <result>
         <title>Excerpts, chorals 250, 320</title>,
         <soprani>{scoreql:eval(scoreql:sync($o1/score/alto, $o2/alto))}</soprani>
         <tenors>{scoreql:eval(scoreql:sync($o1/score/tenor, $o2/tenor))}</tenors>
       </result>
```

Query 3: Synchronize operator on two scores

Finally, our last example shows the extended concept of score as a voice synchronization which is not necessarily a "music" voice. A basic step of music analysis is to compute the gap (interval) betwen two voices, in order to determine their harmonic profile. The following query produces, for each choral, a vScore containing the soprano and bass voices, and a third voice measuring the interval between the two.

```
for $o in collection("Chorals")/opus[@id="BWV250"]
let $soprAndBass := scoreql:sync($o/score/soprano, $o/score/bass)
let $intervals := scoreql:map($soprAndBass, "intervals()")
return scoreql:eval (scoreql:sync($soprAndBass, $intervals))
```

Query 4: Sync & Map operators, generation of a new voice

5.3. Implementation

Our system has been fully implemented, integrated in an existing Digital Score Library (Neuma), and publicly released as a virtual (Docker) machine at http://hub.docker.com. It relies on off-the-shelf tools, namely a native XML database, BaseX[9] and a music notation library, music21[10][CA10].

Query evaluation

The query evaluation mechanism is illustrated by Fig. 11. First (step 1), the query is submitted to the XQuery processor, and retrieves the set of opuses matching the `where` clause. Recall that each such opus (say, the circled green node on Fig. 11) is linked to a serialized score.

During the evaluation of the `return` clause, function *eval(exp)* is called, and this triggers the execution of the ScoreAlg expression *exp* embedded in the query. The evaluation of *exp* requires three steps:

[9]http://basex.org
[10]http://web.mit.edu/music21

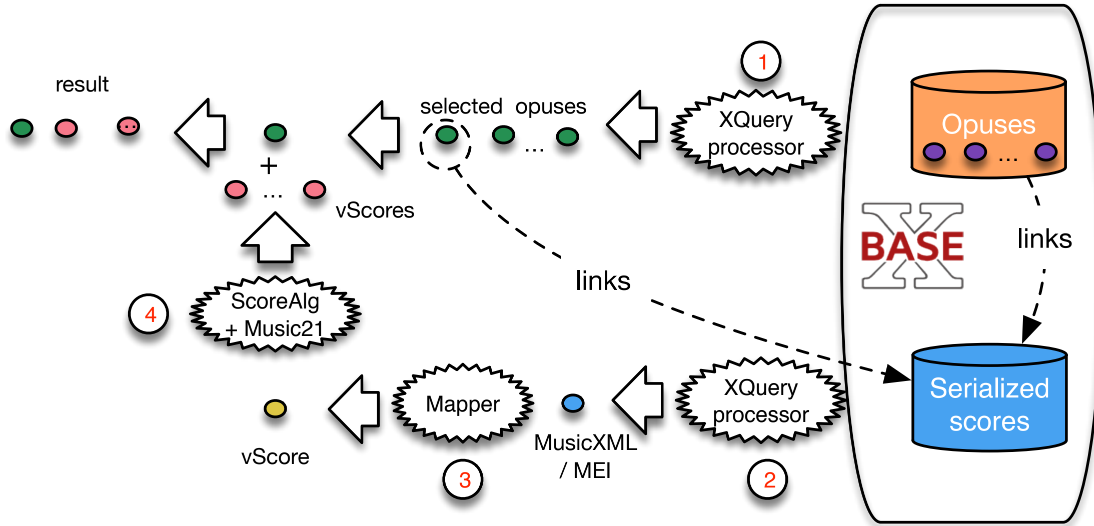


Figure 11: Anatomy of query evaluation

1. Step 2 retrieves the XML score document S (encoded in either MusicXML or MEI) associated to the current opus; this is achieved with a query q_S that simply gets the document from its reference.
2. Step 3 applies a mapping M to extract the vScore from the music content,
3. Finally (step 4), exp is evaluated on the vScore; this requires calls to the ScoreAlg library for algebraic operators, and to the Music21 toolkit for User Defined Functions.

We finally obtain one or several vScore(s), that can be associated with the opus and delivered as a result item. The whole mechanism is somehow similar to those used in mediation systems [GMUW00] with Global As View paradigm (GAV), where items are obtained from a source thanks to a local query (here, q_S on the XML database of score documents), mapped toward a global schema (here, our data model of vScores) and finally processed by the query language associated to the global schema (in our case, the score algebra). The "evaluation plan" of an algebraic expression exp is therefore a sequence $[exp \leftarrow M \leftarrow q_S]$. We examine execution strategies for such plans in the next section.

5.4. Query optimization and evaluation

Two queries are equivalent if they yield the same result on any input. Given a sequence $[exp \leftarrow M \leftarrow q_S]$, an optimizer explores the space of equivalent evaluations $[exp' \leftarrow M' \leftarrow q'_S]$. Since we do not consider alternative evaluations of M, the problem reduces to find q'_S and exp' such that $[exp' \leftarrow M \leftarrow q'_S]$ is equivalent to $[exp \leftarrow M \leftarrow q_S]$ and achieves better performances. There are three rewriting possibilities:

1. rewriting q_S to an equivalent q'_S (i.e., $[exp \leftarrow M \leftarrow q_S]$ is then equivalent to $[exp \leftarrow M \leftarrow q'_S]$;

Name	Rule	Comment
R1	$\sigma_{F_1}(\sigma_{F_2}(S)) \equiv \sigma_{F_1 \wedge F_2}(S))$	Standard selection decomposition
R2	$\sigma_F(S_1 \oplus S_2) \equiv \sigma_F(S_1) \oplus \sigma_F(S_2)$	Same comment as above.
R3	$\sigma_F(S_1 \circ S_2) \equiv \sigma_F(S_1) \circ \sigma_F(S_2)$	Same comment as above.
R4	$\mu_{v_i \dots v_n}(\sigma_F(S)) \equiv \sigma_F(\mu_{v_i \dots v_n}(S))$	NB: σ is a filter applied to *all* the events of a score, hence the equivalence
R5	$\mu_{v_i \dots v_n}(S_1 \oplus S_2) \equiv \mu_{v_i \dots v_j}(S_1) \oplus \mu_{v_{j+1} \dots v_n}(S_2)$	With $v_{i..j} \in S_1, v_{j+1..n} \in S_2$
R6	$\mu_{v_i \dots v_n}(S_1 \circ S_2) \equiv \mu_{v_i \dots v_n}(S_1) \circ \mu_{v_i \dots v_n}(S_2)$	Merge commutes with projection
R7	$(S_1 \oplus S_2) \circ S_3 \equiv (S_1 \circ \mu_1(S_3)) \oplus (S_2 \circ \mu_2(S_3))$	Synchronization can be applied on disjointed set of voices
R8	$(S_1 \oplus S_2) \oplus S_3 \equiv S_1 \oplus (S_2 \oplus S_3)$	Associativity of synchronization
R9	$(S_1 \circ S_2) \circ S_3 \equiv S_1 \circ (S_2 \circ S_3)$	Associativity of merge
R10	$[\pi_{v_i \dots v_n}(S) \leftarrow M \leftarrow q_S] \equiv [S \leftarrow M \leftarrow \pi_{v_i \dots v_n}(q_S)$	Projecting out voices can be done with XQuery
R11	$[\sigma_F(E) \leftarrow M \leftarrow q_S] \equiv [S \leftarrow M \leftarrow \sigma_F(q_S)$	Only for selection formulas F expressible in XQuery.

Table 1: Rewriting rules

2. rewriting *exp* to an equivalent *exp′* (i.e., $[exp \leftarrow M \leftarrow q_S]$ is then equivalent to $[exp' \leftarrow M \leftarrow q_S]$);
3. or finding a global rewriting that "moves" some operations between *exp* to q_S.

The first possibility resorts to the XQuery optimizer. We focus on the others. Table 1 gives a set of equivalent rules which can be applied for query rewriting, assuming that syntactic constraints are fullfilled. The first part (R1-R9) addresses local rewriting rules of ScoreAlg expressions, the second (R10 and R11) shows how we can "push" through the mapper operations from ScoreAlg to XQuery.

The last two rules deserve somes explanations: they rely on the fact that both the projection and the selection operators can be equivalently applied either in ScoreAlg and XQuery. This is always true for the projection: indeed, projecting on voices in ScoreAlg can be done with XQuery on both MusicXML or MEI documents since voices are explicitly (though intricately) encoded. Rule R10 (selection) entails more restrictions, since the equivalence depends on the selection formula F and on the ability to transpose it in XQuery.

Let us take as an example from our query 1:

```
for $o in collection("Chorals")/opus
let $incipit := scoreql:select ($o/score/soprano, "measure() in [1,5]")
return (...)
```

The score query evaluation plan is $[\sigma_{\text{measure()in[1,5]}}(\pi_{\text{soprano}}(O.score)) \leftarrow M \leftarrow q_S]$, $O \in$

Chorals. Thanks to rule R10, it can be rewritten as

$$[\sigma_{\text{measure()in}[1,5]}(O.score) \leftarrow M \leftarrow \pi_{\text{soprano}}(q_S)]$$

In other word, we can apply the mapping on a score that contains only the voices of interest to the ScoreAlg expression. This is likely to improve significantly the performance since, as measured by our experiments (see next section), the mapping is the dominant cost in the query evaluation, and depends on the size of the input score.

Pushing the selection is more difficult. In this specific case, it depends on the presence of the `measure` number in the score encoding, in such a way that it can easily be used in an XQuery filter. We did not incorporate rule R11 in our current implementation, but it would certainly be worth considering in a system dealing with large collections.

Our optimization algorithm is based on a common heuristic: limit as much as possible the size of the data items submitted to the operators.

- Push down selections (R1 to R3) and projections (R4 to R6); in particular, rule R4 should push the projection close to the input score.
- Apply R10 to push the projection, through the mapping operator M, to q_S.

The next section reports the performance of these evaluation steps in our current implementation.

5.5. Performances

	XQuery	Mapping	Alg	# result	Mapping/doc	Alg/doc
Q1	1,042	35,061	12,493	403	87	31
Q2	1,172	333	466	2	166.5	233
Q3	515	270	28	2	138.5	14
Q4	537	121	166	1	166	121

Table 2: Running time decomposition for each query (ms)

Queries 1 to 4 have been evaluated on the corpus of 403 scores of *Bach Chorales*[11]. Table 2 gives their running time (in ms) with our system. We decompose this cost according to the three steps of an evaluation plan $[exp \leftarrow M \leftarrow q_S]$, where q_S is the standard part of XQuery, M is the mapping, and exp is the algebraic expression applied to each score. Since the pair (M, exp) is evaluated for each document of the XQuery result, we also

[11] Available at http://neuma.huma-num.fr/home/corpus/composers:bach:chorals/

	Time (ms)	Ratio	# op
Mapping	87	73%	1
Select operator	26	22%	1
Result	5	5%	1
Total	118		

Table 3: Time to process a ScoreAlg expression for each score in Query 1

	Time (ms)	Ratio	# op
Mapping	166.5	41.68%	1
UDF	13	3.25%	1
Map	207	51.81%	2
Sync	3.5	0.88%	1
Result	9.5	2.38%	1
Total	399.5		

Table 4: Time to process a ScoreAlg expression for each score in Query 2

report the cardinality of this result, and the average cost of M and exp per document. For instance, the first query retrieves 403 scores, and the average unit cost of the mapping is $35,061/403 = 87$ms.

The XQuery cost is almost constant for the two first queries. The execution plan is then simply a sequential scan of the collection. Queries 3 & 4 contain a `where` clause (brackets on the *opus* path) on the document ID, and BaseX uses in that case a direct access thanks to an internal index.

Let us now examine the details of the score manipulation operators for each query. The relative costs of M and exp depend on the complexity of the latter. Table 3 shows this detail for Q1 which simply executes a selection $\sigma_{measure<5}$ to obtain the first 5 measures of the soprano voice. This is done for each document in 26 ms on average, whereas the mapping itself takes 87 ms (the **Result** line corresponds to the time spent by BaseX to built the result item, and `#op` is number of applications of the operator). Clearly, the time spent to access the score and map its content as an instance of our data model is predominant for such basic algebraic expressions.

Table 4 reports the detailed cost for each operator in Q2. The selection based on an UDF call (highest note above "F5") has a minor impact. The synchronize operator is also quite efficient because it simply builds a new structure without accessing to the values. On the other hand, the Map operator counts for half of the global cost. This makes sense if we remember that it requires an access to each individual event, in order to apply some domain function. Its cost is therefore proportional to the size of the voices, and also depends on the complexity of the function itself. We must notice that the Map is applied on two voices of each score and thus counts twice.

	Time (ms)	Ratio	# op
Mapping	138.5	90.82%	1
Sync	6.5	4.26%	1
Result	7.5	4.92%	1
Total	152.5		

Table 5: Time to process a ScoreAlg expression for each score in Query 3

	Time (ms)	Ratio	# op
Mapping	121	42.16%	1
Sync	16	5.57%	2
Map	140	48.78%	1
Result	10	3.48%	1
Total	287		

Table 6: Time to process a ScoreAlg expression for each score in Query 4

This is confirmed by the analysis of Q3 (Table 5) which does not feature a Map. The last query extracts two voices, synchronized in a new score `$SoprAndBass` that is processed to produce intervals. The new resulting voice is then resynchronized with `$SoprAndBass` to produce the final result. In spite of this apparently complex structural manipulations, the analysis of the Sync operations shows that they count for a marginal part. Computing with Map the intervals between the two voices constitutes the main part of the evaluation cost (Table 6).

Regarding the impact of Rule R10 that "pushes" projections from ScoreAlg to XQuery, we can note that it reduces significantly the running time. Query 1 uses only one voice while the three other queries projects two. For Query 1, the XML document to be parsed is therefore shorter and the Mapping step costs less, as can be seen in Table 3. This also explains the difference (130ms less) between Query 1 and Query 2 for the XQuery part.

In summary, this short performance study illustrates the main characteristics of an implementation which is not designed to achieve high speed for very large collections, but rather aims at limiting implementation efforts by relying as much as possible on off-the-shelf tools and systems. In this specific case, the system can be seen as a tight integration of a database system (BaseX) with specialized music score manipulation functions (our algebra and UDFs supplied by music21). If we leave apart the special case of the Map operator, it appears that the cost of executing algebraic operators is negligible regarding the time spent to load and parse the score's XML document. Indeed, once a vScore instance is obtained from the mapping phase, our algebra operates in RAM and yields very fast response time. Introducing a Map in a query is, as explained in Section 4, a powerful mean to extend the expressive power of the algebra to user-defined functionalities. This power has a cost: the function might be complex, and more importantly it has to be applied to

the lowest level of the score hierarchy, *i.e.,* events.

Overall, we believe that our implementation choices constitute a satisfying trade-off. On the one hand, a limited implementation effort results in a *declarative* system which allows to express concisely complex operations on score collections. This has to be compared with the time spent to write, test, run and deploy an equivalent Python or Java code in a standard programming environment. On the other hand, the overhead of our algebra has a minimal impact on the query evaluation which, as shown above, is dominated by the time spent by loading a score and applying UDFs with Map. In other words, the cost in that case is similar to that of running a dedicated program.

5.6. ScoreQL *in practice*

The ScoreQL system has been packaged as a lightweight Docker image that can be downloaed and instantiated easily in any `Docker` environment[12]. This image contains all the necessary components to store and query a collection of digital scores, via its REST interface, documented at http://cchum-kvm-scorelibneuma.in2p3.fr/ScoreQL/. This image is available on DockerHub at https://hub.docker.com/r/traversn/scoreql/.

We usw the system as part of the Neuma public digital library. Fig. 12 shows the Web querying interface. Queries such as those given as examples in the previous section can be expressed. The result are proposed as a list of document links.

6. Conclusion

Music is everywhere in the digital world, present in many forms, accessed by countless people and institutions, for all kinds of usages. Organizing and searching large collections of music documents has so far mostly focused on a generic information retrieval approach that either exploits metadata, or applies similarity-based content retrieval techniques. But music has a structure, and there exists a language to describe this structure: music notation. Our work uses this language as a basis for the description of music content.

Our approach leads to the design and implementation of a query language that operates on the music structure, extracted on the fly from the raw document encoding. 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. Finally, we showed how this algebra can be integrated at low cost in an existing database system, and provided implementation guidelines and a working implementation.

[12]https://www.docker.com/

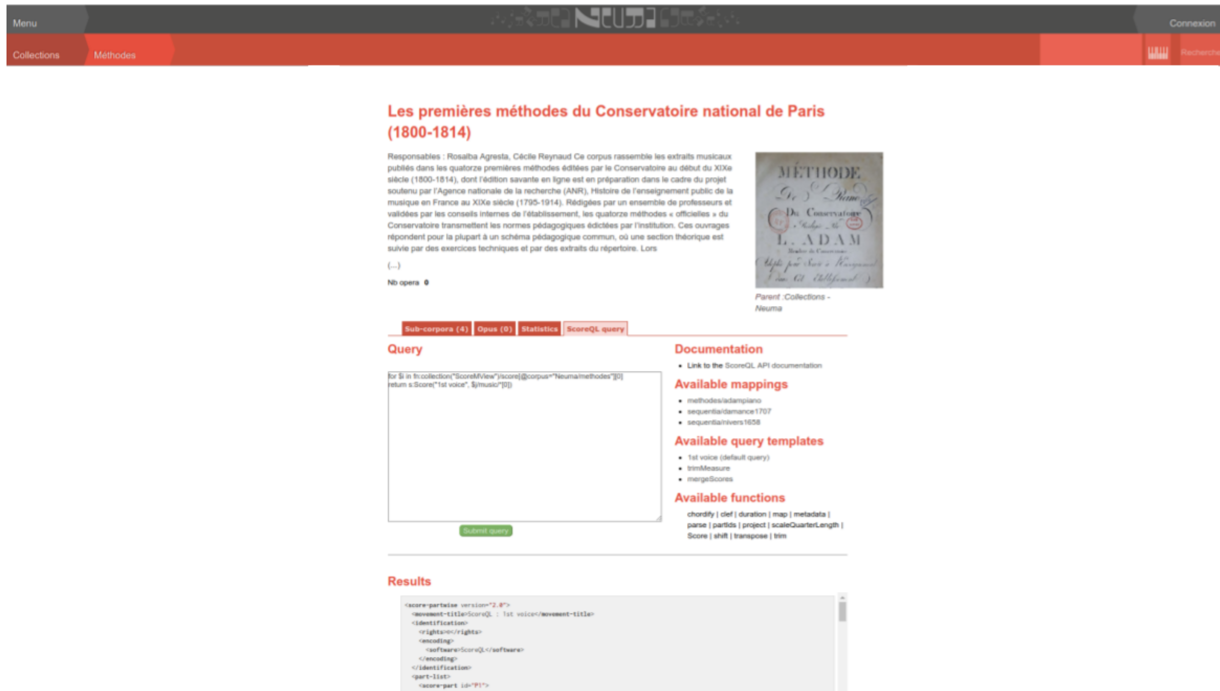


Figure 12: Screenshot of the querying interface integrated to Neuma

In principle, the approach applies to music documents in any format. In practice, it requires a mapping that transforms the raw content to instances of our model, and this mapping is, in the moment, easier and more reliable for music scores. Nothing however prevents our model in the future to be applied to other music document formats as well.

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 music documents in general. In addition to its integration as part of the Neuma system, we chose to release our system as a virtual Docker component, equipped with a REST interface, and thus directly usable by any user who wishes to enrich her music repository with data management facilities.

Finally, in a context where the volume of copyright-free music is constantly expanding, we hope that proposing a robust and well-founded language to manage large collections of music documents brings the well-known advantages of a database approach to the so-far mostly ignored field of structured data management for music collections. We also hope that our approach, that identifies and exploits implicit structures hidden in the encoding of digital documents, can serve as an inspiration in the fields of humanities. The revolution brought by Internet technologies has encouraged the production, release and publication of countless digitized archives related to history, arts or culture. Designing tools that help to figure out their implicit structure, and the data management machineries that can process

them, are important requirements to make sense of these repositories and leverage the information and knowledge they contain.

Acknowledgments This work is partially supported by the ANR MuNIR project. We are quite grateful to Florent Jacquemard for in-depth discussions and careful readings of the paper.

[AAC+08] Serge Abiteboul, T. Allard, P. Chatalic, G. Gardarin, A. Ghitescu, F. Goasdoue, I. Manolescu, B. Nguyen, M. Ouazara, A. Somani, N. Travers, and G. Vasile. WebContent: Efficient P2P Warehousing of Web Data. In *VLDB'08 Very Large Data Base*, pages 1428–1431, August 2008.

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

[ABSW04] Norman H. Adams, Mark A. Bartsch, Jonah B. Shifrin, and Gregory H. Wakefield. Time series alignment for music information retrieval. In *Proceeding of International Conference on Music Information Retrieval (ISMIR)*, pages 303–311, 2004.

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

[BB01] David Bainbridge and Tim Bell. The challenge of optical music recognition. *Computers and the Humanities*, 35(2):95 – 121, 2001.

[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.

[BDG+13] Emmanouil Benetos, Simon Dixon, Dimitrios Giannoulis, Holger Kirchhoff, and Anssi Klapuri. Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41(3):407–434, 2013.

[Bel11] Juan Pablo Bello. Measuring Structural Similarity in Music. *IEEE Transactions on Audio, Speech, and Language Processing*, 19:2013–2025, 2011.

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

[CVG+08] MA Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. Content-based music information retrieval: current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696, 2008.

[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.

[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 XML Score Databases: XQuery is not Enough! In *Proceeding of 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. *The Virtual Score: Representation, Retrieval, Restoration*, chapter "MusicXML for Notation and Analysis", pages 113–124. W. B. Hewlett and E. Selfridge-Field, MIT Press, 2001.

[GSD08] Joachim Ganseman, Paul Scheunders, and Wim D'haes. Using XQuery on MusicXML Databases for Musicological Analysis. In *Proceeding of International Conference on Music Information Retrieval (ISMIR)*, 2008.

[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.

[Kla04] Anssi P Klapuri. Automatic music transcription as we know it today. *Journal of New Music Research*, 33(3):269–282, 2004.

[LS03] A. Lerner and D. Shasha. AQuery: Query Language for Ordered Data, Optimization Techniques, and Experiments. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, pages 345–356, 2003.

[LSW+04] A. Lerner, D. Shasha, Z. Wang, X. Zhao, and Y. Zhu. Fast Algorithms for Time Series with applications to Finance, Physics, Music, Biology, and other Suspects. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 965–968, 2004.

[MEI15] Music Encoding Initiative. http://music-encoding.org, 2015. Accessed April 2019.

[RFP+12] Ana Rebelo, Ichiro Fujinaga, Filipe Paszkiewicz, André R. S. Marçal, Carlos Guedes, and Jaime S. Cardoso. Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval*, 1:173–190, 2012.

[Rol02] Perry Rolland. The Music Encoding Initiative (MEI). In *Proceedings of the International Conference on Musical Applications Using XML*, pages 55–59, 2002.

[SGU14] Markus Schedl, Emilia Gómez, and Julián Urbano. Music Information Retrieval: Recent Developments and Applications. *Foundations and Trends in Information Retrieval*, 8:127–261, 2014.

[SYBK16] Diego Furtado Silva, Chin-Chia Michael Yeh, Gustavo E. A. P. A. Batista, and Eamonn J. Keogh. SiMPle: Assessing Music Similarity Using Subsequences Joins. In *Proceeding of International Conference on Music Information Retrieval (ISMIR)*, 2016.

[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 *Proceeding of International Conference on Music Information Retrieval (ISMIR)*, 2005.

[USG14] Karen Ullrich, Jan Schlüter, and Thomas Grill. Boundary detection in music structure analysis using convolutional neural networks. In *Proceeding of International Conference on Music Information Retrieval (ISMIR)*, pages 417–422, 2014.

[Whe15] Philip Wheatland. Thoth music learning software, v2.5, Feb 27, 2015. http://www.melodicmatch.com/.

[XQu07] XQuery 3.0: An XML Query Language. World Wide Web Consortium, 2007. https://www.w3.org/TR/xquery-30/.