Vers un Traitement Algébrique de la Notation Musicale

Raphaël Fournier, Philippe Rigaux, Nicolas Travers

VERS UN TRAITEMENT ALGÉBRIQUE DE LA NOTATION MUSICALE

Raphaël Fournier-S'niehotta
CNAM
fournier@cnam.fr

Philippe Rigaux
CNAM
philippe.rigaux@cnam.fr

Nicolas Travers
CNAM
nicolas.travers@cnam.fr

RÉSUMÉ

Les partitions musicales sont des objets structurés, et nous pouvons donc envisager des opérateurs capables de modifier la structure d'une partition, d'en combiner plusieurs, et de produire de nouvelles partitions à partir de contenus pré-existants. Les encodages actuellement disponibles sont cependant conçus à des fins de rendu et d'échange, et ne peuvent pas être directement exploités pour des manipulations algébriques. Nous proposons une approche qui tire parti d'une modélisation et d'une extraction du contenu musical caché au sein de la notation, et nous définissons un ensemble d'opérateurs que l'on peut combiner pour générer de nouvelles partitions à partir d'un corpus existant. Nous montrons que cette approche fournit un outil de haut niveau permettant d'exprimer des applications simples et utiles, susceptible d'être mis en place facilement avec des composants standards et nécessitant peu de développements spécifiques.

1. INTRODUCTION

L'encodage numérique des notations musicales est une entreprise de longue date, et a donné lieu à de nombreuses propositions [19]. Aujourd'hui, les principaux encodages reposent sur le format XML, représentant la notation musicale sous forme de documents structurés. MusicXML [10] est probablement l'un des plus répandus en tant que format d'échange pour les principaux logiciels de composition (*Finale, Sibelius*, et *MuseScore*). Ce besoin d'interopérabilité a donné lieu à un codage représentant à la fois la structure, le contenu et l'information de rendu. À cela s'ajoute le fait que la syntaxe de la notation musicale varie, ainsi que son interprétation, selon les lieux, les périodes, les styles, et les contextes culturels. Concevoir un format apte à capturer cette grande variabilité dans une représentation unique et cohérente est délicat. L'initiative MEI [18, 14] tente de relever ce défi avec un format extensible [11]. Elle repose sur des composants prédéfinis tels que, par exemple, le module *Common Music Notation* (CMN). Récemment, les premières discussions lors du *W3C Music Notation Community Group* [21], lancé en septembre 2015, soulignent également la difficulté de définir un cadre d'encodage général et cohérent capable de saisir toutes les nuances de notations syntaxiques et sémantiques en fonction du contexte d'utilisation.

Si l'on considère un contexte spécifique, et si l'on suppose l'existence d'un format spécialisé capable de couvrir correctement les particularités musicales de ce contexte (par exemple, un module MEI comme indiqué ci-dessus), alors il est naturel de supposer que ce format doit encapsuler un modèle de données encodant du contenu structuré, du moins pour une partie de ce répertoire musical. En se concentrant par exemple sur le *Common Music Notation*, ce modèle de données peut en partie être identifié par les points communs de différents formats tels que MusicXML, MEI et Lilypond. Au-delà de leurs différentes motivations et approches initiales, ils partagent un ensemble de caractéristiques propres au contenu musical devant être représenté.

Bien sûr, cette notion de "modélisation du contenu" est délicate à formaliser. La plupart des éléments décrivant le rendu d'une partition peuvent, dans une certaine mesure, être considérés comme significatifs et faisant partie intégrante du sens engendré par la notation. La séparation du contenu et des éléments de rendu est d'ailleurs un sujet de discussion récurrent (voir par exemple [15]). Un des plus grands avantages résiderait dans la possibilité de "styliser" le contenu des partitions, de la même manière que ce qui a été réalisé sur le rendu de contenu HTML / XML dans le domaine des documents Web.

Dans cet article, nous développons un argument complémentaire en faveur d'une séparation stricte du contenu et du rendu des partitions musicales. Être capable d'identifier une modélisation du contenu musical permet en effet de considérer un corpus de partitions comme une collection d'objets structurés, et de définir des opérateurs capables de les manipuler afin d'en extraire des fragments, de combiner ces fragments, et donc de créer un contenu dérivé à partir de contenus existants.

Cette démarche de modélisation permet de fournir, par l'intermédiaire d'un langage de haut niveau, de nombreuses opérations utiles, telles que :

- *la gestion automatique de contenus* : découper les différentes parties d'une partition, les distribuer aux pupitres virtuels des musiciens, appliquer des transpositions et ajouter des décorations (directives) en fonction des besoins ; à l'inverse, fusionner des parties distinctes dans une seule partition.
- *des recherches et comparaisons* : chercher des partitions en fonction de certains critères ; extraire les fragments correspondants ; aligner ces fragments dans une nouvelle partition pour un complément d'investigation.
- *des analyses avancées* : obtenir des caractéristiques analytiques (par exemple, la progression harmonique) ;

annoter les partitions ; produire de nouvelles représentations soulignant les aspects structurels.

Nous proposons dans cet article d'équiper une Bibliothèque Numérique de Partitions (*Digital Score Library*, DSL) avec un langage algébrique, afin d'en extraire des partitions "intentionnelles" à partir de partitions "extensionnelles" (*e.g.,* la bibliothèque), à l'instar des bases de données relationnelles et de leur algèbre [1]. Nous nous concentrons sur les opérateurs en forme close : tout opérateur prend en entrée une ou plusieurs instances du modèle et en produit une nouvelle. La fermeture permet la composition : si s est une instance du modèle (une partition) et o_1, o_2 sont deux opérateurs, alors $o_1(s)$ est une partition, $o_2(o_1(s))$ est également une partition, et l'on obtient ainsi une structure algébrique (au sens mathématique) qui nous permet de manipuler la notation musicale afin de produire de nouvelles représentations.

Cette approche apporte les avantages bien connus des systèmes spécialisés de gestion de données à la conception et la mise en œuvre d'applications traitant la musique notée. Parmi ceux-ci, rappelons entre autres : (i) la capacité de reposer sur un modèle de données stable, bien défini et expressif, (ii) l'indépendance entre la modélisation logique et la conception physique, permettant aux développeurs de rester à un haut niveau d'abstraction et (iii) l'efficacité des opérateurs ensemblistes et des index fournis par le système de gestion de données.

Il faut souligner que nous n'envisageons pas notre algèbre dans une optique *générative* comme le font d'autres propositions (par exemple Euterpea [12]). Nous revenons sur ce que nous considérons comme la particularité de notre travail en section 5, après avoir présenté le détail de notre approche selon le plan suivant :

1. *Un cadre* : Nous décrivons dans la section 2 un cadre conceptuel où l'encodage des partitions dans une DSL peut être exploité pour les manipulations du contenu structuré.
2. *Un modèle* : La section 3 propose un modèle intégrant la plupart des caractéristiques standards du module CMN, ainsi qu'un langage d'interrogation algébrique de haut niveau.
3. *Un guide d'implantation* : Pour finir, nous fournissons en section 4 une discussion technique sur les choix d'implémentation montrant les quelques efforts nécessaires à l'accomplissement de notre vision.

La section 5 développe l'état de l'art et la section 6 présente une conclusion et expose nos perspectives de travail.

2. CADRE CONCEPTUEL

La figure 1 représente schématiquement le système que nous proposons. Les choix d'implémentation sont discutés dans la section 4.

La couche inférieure est une bibliothèque numérique de partitions (DSL) dont le rôle est de gérer un corpus de partitions sous un encodage donné, comme MusicXML,

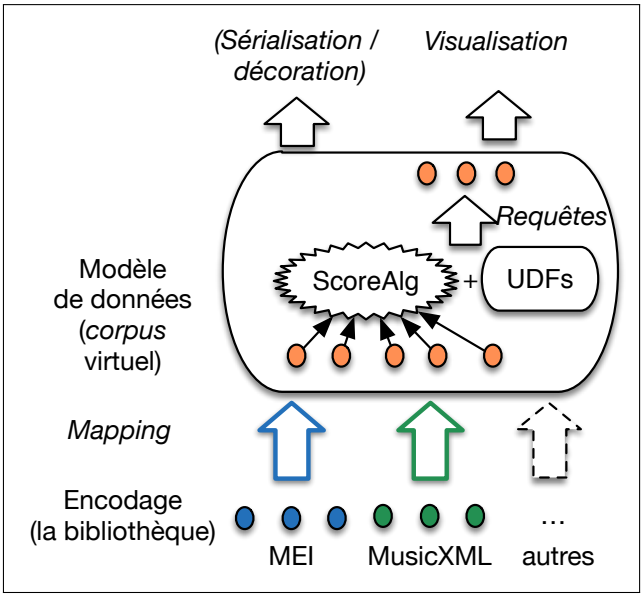


Figure 1. Abstraction de notre système

MEI, ou tout autre format dédié (par exemple, Guido ou *Humdrum*). Comme expliqué dans l'introduction, ces encodages ne sont pas conçus pour supporter des manipulations basées sur le contenu, et, de fait, il est presque impossible de le faire au-delà des opérations les plus simples. L'accès à l'information musicale *explicite* y est complexe, en raison de l'imbrication complexe des éléments de contenu et de ceux dédiés au rendu. L'extraction d'une simple séquence de notes, par exemple de MusicXML ou MEI, n'est pas une tâche triviale. Extraire les caractéristiques *implicites* (par exemple la tonalité) du matériau musical encodé est encore plus ardu.

Il nous paraît donc indispensable d'introduire une correspondance (*mapping*) entre l'encodage, vu comme une représentation purement physique propre au stockage, et la couche du modèle de données. Ce *mapping* extrait le contenu de l'encodage pour satisfaire les contraintes structurelles du modèle. Un *mapper* doit être défini pour chaque type d'encodage, comme le montre la figure 1 supposant que les documents MusicXML et MEI cohabitent dans une même DSL. Ajouter une nouvelle source de données utilisant un encodage différent demanderait uniquement d'y associer un nouveau *mapper* dédié à cet encodage. Chaque document de la DSL est traduit par le *mapper* en une instance (virtuelle) de notre modèle. Cela crée des objets intentionnels (sans matérialisation) d'un corpus de notations musicales. Nous appellerons ces instances *vScore* par la suite.

La couche "modèle" englobe à la fois la représentation des données et les opérateurs sur ces mêmes données. En outre, nous distinguons deux types d'opérateurs : les *opérateurs structurels* et les *fonctions utilisateurs* (*User Defined Functions*, UDFs). Les premiers correspondent à l'idée que la gestion de partitions structurées repose sur un ensemble restreint d'opérateurs fondamentaux, que l'on peut regrouper au sein d'une algèbre de partitions, nom-

mée SCOREALG. Ces opérateurs sont définis une fois pour toutes. Les UDFs en revanche sont là pour faire face à la richesse des possibilités de manipulation de la notation musicale. La combinaison des opérateurs de la SCOREALG et des UDFs constitue la partie opérationnelle du modèle. Elle fournit un langage de requêtes dont les expressions définissent sans ambiguïté l'ensemble des transformations produisant de nouvelles *vScore*s à partir d'un corpus de base.

Le résultat d'une requête est lui-même un nouveau corpus intentionnel. Cela donne lieu à plusieurs exploitations possibles. Tout d'abord, le résultat peut être conservé (à nouveau, intentionnellement) au sein de l'espace utilisateur, comme une "vue" (selon la terminologie des bases de données) sur le corpus de base. Un artiste pourrait par exemple garder un ensemble de pièces pour ses prochaines répétitions / concerts ; un analyste pourrait se constituer un corpus de travail. Deuxièmement, le résultat de la requête peut être visualisé, éventuellement avec des représentations qui mettent l'accent sur des aspects analytiques, ceux-ci calculés à partir des partitions.

Enfin, les partitions intentionnelles du résultat pourraient être sérialisées en mémoire persistante, dans l'un des formats d'encodage présents. C'est là que la "stylisation" du contenu des partitions intervient, dans un processus qui est l'exact opposé du *mapping* extrayant le contenu à partir de documents MusicXML ou MEI. La sérialisation des *vScore*s implique la "décoration" de leurs contenus à l'aide de fonctionnalités de rendu : les voix doivent être affectées à la portée, la clé doit être choisie pour chaque portée, les altérations doivent être affichées selon une méthode globale, etc. Nous ne nous étendrons pas sur ce processus qui, comme expliqué ci-dessus, est directement lié à des questions complexes de séparation du contenu et de la notation musicale, et va bien au-delà de la portée de notre travail actuel. Remarquons que, en un sens, la vision que nous venons de décrire constitue un cadre de travail pour explorer ces questions. Une manière de disposer d'une séparation stricte entre le contenu et le rendu serait en effet de définir des opérateurs composés d'une paire (*mapping* / sérialisation) qui produiraient un rendu alternatif d'une partition tout en préservant son contenu.

Dans les sections suivantes, nous mettons en œuvre cette approche conceptuelle avec un modèle appliqué à la CMN, et exposons nos choix d'implantation pour sa mise en pratique.

3. LE MODÈLE DE DONNÉES

Nous présentons à présent un modèle de données basé sur les concepts d'événement, de voix et de partition. Une voix est une séquence d'événements dans un domaine de valeurs qui peut être – mais pas seulement – de nature musicale (notes, silences, accords, syllabes). Dans une voix, les événements se succèdent pour une période précise, sans recouvrement. Enfin, les partitions sont modélisées comme une synchronisation de plusieurs voix.

3.1. Schéma : Voix, Partition et Opus

Le domaine du temps $\mathcal{T}$ est un ensemble ordonné et dénombrable d'estampilles, isomorphe à l'ensemble des rationnels $\mathbb{Q}$. Pour simplifier les exemples, nous supposerons l'existence d'une unité de temps représentant le plus petit intervalle possible entre deux événements musicaux.

Une partition temporelle $\mathcal{P} = \{I_1, \cdots, I_n\}$ est un ensemble d'intervalles I_i ouverts à droite tels que : $\forall i, j, I_i \cap I_j = \emptyset$ et $\bigcup_i I_i = \mathcal{P}$. Comme une partition (musicale) ne couvre jamais totalement $\mathcal{T}$ et peut contenir des "trous", nous relâcherons la seconde condition.

Nous aurons besoin de la notion de partitions temporelles compatibles pour la fusion, définie ci-dessous.

Définition 1 *Deux partitions temporelles $\mathcal{P}_1 = \{I_1, \cdots, I_n\}$ et $\mathcal{P}_2 = \{J_1, \cdots, J_m\}$ seront dites* compatibles pour la fusion *si leur union (avec élimination des doublons) est une partition temporelle, i.e., $\forall i \in [1, n], j \in [1, m]$, soit $I_i \cap J_j = \emptyset$ soit $I_i = J_j$.*

Par exemple, $I_1 = \{[0, 2[, [4, 6[, [8, 9[\}$ est compatible pour la fusion avec $I_2 = \{[2, 4[, [6, 7[, [8, 9[\}$. L'union des deux est la partition temporelle

$$I_1 \cup I_2 = \{[0, 2[, [2, 4[, [4, 6[, [6, 7[, [8, 9[\}$$

dans laquelle on retrouve bien chaque intervalle de I_1 ou I_2.

3.1.1. Événements

Définition 2 (Événement) *Étant donné un domaine de valeurs **dom**, un* événement *est un triplet $< v, t_1, t_2 >, v \in$ **dom**, $\mathrm{t}_1 < \mathrm{t}_2$, où t_1 et t_2 sont deux estampilles, représentant le fait que v persiste durant la période de temps $[t_1, t_2[$. Nous notons $\mathcal{E}$(**dom**) l'ensemble des événements sur **dom**.*

Nous ne nous restreignons pas à une liste de domaines pré-définis. Certains domaines permettent de représenter des informations de nature musicale, d'autres des informations dérivées comme les intervalles ou des données de nature analytique. Comme nous le verrons, cela permet de définir une notion étendue de "partition" combinant des événements musicaux et non-musicaux dans une même synchronisation.

Les événements musicaux prennent leurs valeurs dans les domaines suivants :

- Les sons (**dsound**) ; le domaine **dsound** représente l'émission simultanée de 0 (silence), 1 (note) ou plusieurs (accords) sons élémentaires.
- Les syllabes (**dsyll**) ; utilisés dans le cadre vocal.

Les sons sont en principe des objets complexes avec plusieurs composants (la hauteur, l'intensité, le timbre). En pratique, la notation traditionnelle se contente de représenter la (les) hauteur(s), le timbre étant suggéré indirectement par l'instrument et l'intensité, optionnellement, par des directives.

3.1.2. Voix

Les événements n'existent pas individuellement, mais sont toujours intégrés à une série temporelle, que nous appellerons simplement *voix*.

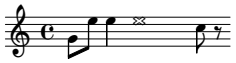
Définition 3 (Voix) *Une voix v de type **Voice(dom)** est une fonction partielle injective de $\mathcal{T}$ vers $\mathcal{E}$(**dom**) telle que $v(t) = e$ ssi $t \in [e(t_1), e(t_2)[$.*

La notion de fonctions définit implicitement une contrainte de non-recouvrement temporel des événements d'une voix. Si e_1 et e_2 sont deux événements d'une voix v, alors

$$[e_1(t_1), e_1(t_2)[\cap[e_2(t_1), e_2(t_2)[= \emptyset$$

En revanche, nous n'imposons pas qu'une voix définisse un partitionnement de $\mathcal{T}$: il existe des estampilles t telles que $v(t)$ n'est pas définie, ce que nous notons $v(t) = \perp$, $\perp$ représentant en quelque sorte l'absence d'événement. Pour simplifier les notations, nous assimilerons parfois dans ce qui suit une voix **Voice(dom)** à une fonction *totale* de $\mathcal{T}$ vers $\mathcal{E}(\textbf{dom} \cup \perp)$, et nous noterons **adom**(v) la partition temporelle obtenue en projetant v sur $\mathcal{T}$, i.e., les intervalles constitués de toutes les estampilles t telles que $v(t) \neq \perp$.

La figure suivante illustre une instance de voix, avec la représentation conventionnelle à gauche (la croix représente l'interruption de la voix pendant une noire), et la modélisation sous forme de séquence temporelle, en supposant que l'unité temporelle est la croche. Notez que la répétition d'un même son (les deux mi) est représentable grâce à deux événements, distincts, bien que partageant la même valeur.



$$v(t) = \begin{cases} < g3, 0, 1 >, t \in [0, 1[\\ < e4, 1, 2 >, t \in [1, 2[\\ < e4, 2, 4 >, t \in [2, 4[\\ < \perp, 4, 6 >, \ t \in [4, 6[\\ < c4, 7, 7 >, t \in [6, 7[\\ < r, 7, 8 >, \ \ t \in [7, 8[\end{cases}$$

Figure 2. Modélisation d'une voix musicale

Il est clair que cette représentation est présente, sous une forme syntaxique plus ou moins complexe et imbriquée avec d'autres éléments, dans les codages de la notation musicale. Notre modèle va au-delà car il n'est pas limité au domaine musical : l'ajout de nouveaux domaines permet d'appliquer le concept de voix à toute séquence de symboles. Cela couvre le texte (où le domaine est composé de syllabes) ainsi que, potentiellement, toute séquence d'information liée à la musique (par exemple, des intervalles, des annotations, ou plus généralement des données analytiques). La succession des intervalles entre deux voix peut par exemple être représentée comme une voix de type **Voice (dint)**.

3.1.3. Partition

Les voix peuvent être *synchronisées*. La synchronisation de deux voix v_1 : **Voice (dom$_1$)** et v_2 : **Voice (dom$_2$)**, notée $v_1 \oplus v_2$, est une voix v_3 : **Voice (dom$_1$ $\times$ dom$_2$)** telle que $\forall t \in \mathcal{T}, e_1 \in \mathcal{E}(\textbf{dom}_1), e_2 \in \mathcal{E}(\textbf{dom}_2)$, la propriété suivante est toujours vérifiée :

$$v_3(t) =< e_1, e_2 > \Leftrightarrow v_1(t) = e_1 \ \wedge \ v_2(t) = e_2$$

La synchronisation combine deux voix en unifiant l'interprétation de leurs domaines de temps respectifs. Nous pouvons alors définir une partition comme une synchronisation de voix identifiées par des étiquettes. Une extension simple de cette définition permet de capturer l'organisation hiérarchique des partitions en groupes, parties, etc.

Définition 4 (Partition) *Une* partition *est une structure hiérarchique définie par induction de la manière suivante :*

- *une voix est une partition ;*
- *si $s_1, \cdots, s_n$ sont des partitions, alors la synchronisation $s_1 \oplus \cdots \oplus s_n$ est une partition.*

Le type *d'une partition p est le n-uplet des types de ses composants chacun associée à un nom.*

Par exemple, le type d'un quatuor est défini par :

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

Et le type d'une partition vocale :

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

Une instance de ces types est une série temporelle de $\mathcal{T}$ vers, respectivement, $\mathcal{E}(\textbf{dsound})^4$ et $\mathcal{E}(\textbf{dsound}) \times \mathcal{E}(\textbf{dsyll})$. Conceptuellement, le premier représente une fonction qui associe à chaque instant un 4-uplet d'événements musicaux, et le dernier une fonction qui associe à chaque instant une paire *(syllabe, son)*. Essentiellement, une partition est une série temporelle dans un espace multi-dimensionnel qui étend le concept de voix (i) en permettant à plusieurs événements de se produire simultanément et (ii) en étiquetant les voix par des noms, fournissant un moyen d'y faire référence.

3.1.4. Opus et Corpus

Il reste à introduire les partitions comme attributs d'un modèle de données permettant la représentation de données "classiques", constituées de valeurs atomiques. Nous reprenons simplement le modèle relationnel en étendant les domaines de valeurs aux partitions. Un *opus* est un n-uplet de valeurs qui peuvent être aussi bien des valeurs atomiques (chaînes de caractères, entiers, flottants) que des partitions. Un *corpus* est un ensemble d'*opus* de même structure. Une base de données est un ensemble de corpus. L'exemple suivant montre un schéma possible pour un corpus contenant des quatuors.

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

3.2. L'algèbre musicale

L'algèbre est composée de trois parties : un ensemble extensible de fonctions, l'algèbre des partitions ScoreAlg, et une extension limitée de l'algèbre relationnelle.

3.2.1. Fonctions

Les fonctions temporelles opèrent directement sur le domaine temporel $\mathcal{T}$. L'ensemble des fonctions linéaires de la forme $t \rightarrow mt + n$ est une sous-classe importante dans notre contexte, et en particulier, les fonctions de déformation (*warping*) de la forme $t \mapsto mt$, et les fonctions de décalage (*shifting*) de la forme $t \mapsto t + n$. Elles correspondent respectivement à une déformation homothétique de la durée (préservant les rapports) et à une translation. Il est difficile de décider si des transformations temporelles non linéaires font sens, mais rien ne les interdit *a priori*.

Les fonctions de domaine associent les valeurs d'un ou plusieurs domaine(s) $\mathbf{dom}_1 \times \cdots \times \mathbf{dom}_n$ à une valeur d'un domaine $\mathbf{dom}_o$. La liste des fonctions *n'est pas* fixée *a priori* : le système doit être extensible pour intégrer des fonctions utilisateurs (UDFs, voir la section 2). Nous allons expliquer prochainement comment ces fonctions peuvent être intégrées dans une requête, à partir du moment où leur signature (types d'entrée et de sortie) est connue. Voici une courte liste de fonctions que nous allons utiliser dans nos futurs exemples :

- *transpose()* : **dsound** $\times$ **dint** $\rightarrow$ **dsound**, transpose un son.
- *duration()* : $\mathcal{E}$(**dsound**) $\rightarrow$ **dint**, donne la durée d'un événement.
- *interval()* : **dsound** $\times$ **dsound** $\rightarrow$ $\mathbf{dom}_{\text{int}}$, donne l'intervalle entre deux sons.

3.2.2. Algèbre des partitions, ScoreAlg

L'algèbre ScoreAlg $(\oplus, \mu, \sigma, \circ, \text{map})$ est constituée de cinq opérateurs de synchronisation, de projection, de sélection, de fusion, et d'un opérateur *map* qui applique une fonction à l'ensemble des événements d'une partition.

L'opérateur de synchronisation, $\oplus$, associe temporellement les voix de deux partitions distinctes.

Définition 5 (Synchronisation de voix, $\oplus$) *Soit S_1 et S_2 deux partitions, alors $S_1 \oplus S_2$ est une partition de type* `[type(`S_1`) ∪ type(`S_2`)]`*, définie par :*

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

Si les noms de voix de S_1 et S_2 ne sont pas distincts, un renommage est nécessaire. L'opérateur correspondant est identique à celui de l'algèbre relationnelle.

L'opérateur de projection μ se comporte de manière similaire à l'opérateur relationnel traditionnel. Il extrait une ou plusieurs voix d'une partition.

Définition 6 (Projection, μ) *Si S est une partition de type $[v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$, alors $\mu_{v_{i_1},\cdots,v_{i_m}}(S)$ est une partition définie par :*

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

La sélection ne conserve que les événements satisfaisant une expression booléenne.

Définition 7 (Sélection, σ) *Si S est une partition de type $T = [v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$, alors $\sigma_F(S)$, où F est une expression booléenne, est une partition de type T, telle que pour chaque voix $S.v_i$:*

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

L'opérateur de fusion $\circ$ combine les voix de deux partitions de même schéma, sous la condition que leurs domaines actifs respectifs soient compatibles pour la fusion (cf. Définition 1). Il permet d'aligner séquentiellement deux partitions couvrant des périodes distinctes, mais également de fusionner deux voix parfaitement homorythmiques, passant d'une vision contrapuntique à une vision polyphonique [12].

Définition 8 (Fusion, $\circ$) *Soit S_1 et S_2 deux partitions de même type $T = [v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$ telles que $\mathbf{adom}(S_1.v_i)$ et $\mathbf{adom}(S_2.v_i)$ soient fusion-compatibles, $\forall i \in [1, n]$, alors $S_1 \circ S_2$ est une partition de type T définie par :*

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

Finalement, l'opérateur map_f est la fonction de second ordre des langages fonctionnels. Elle applique une fonction f aux voix d'une partition, et retourne la partition mono-voix constituée des résultats de f.

Définition 9 (Transformation, map) *Si S est une partition de type $[v_1 : \mathbf{dom}_1, \cdots, v_n : \mathbf{dom}_n]$, et f de signature $\Pi_i^n \mathbf{dom}_i \rightarrow \mathbf{dom}_o$, alors $\text{map}_{a:f}(S)$ est une partition de type [a : $\mathbf{dom}_o$] définie par :*

$$[map_{a:f}(S)].a(t) =< f(S.v_1(t), \cdots S.v_n)(t)] >, \forall t \in \mathcal{T}$$

Une expression de ScoreAlg est définie de manière standard comme une composition des opérateurs, sous les conditions de validité évidentes relatives aux schémas d'entrée et de sortie. Voici quelques exemples.

— La sélection d'un fragment d'une partition S et son décalage dans le temps de n unités s'expriment par :

$$\mathrm{map}_{\mathrm{shift_n}}(\sigma_{\mathrm{t}\in[\mathrm{t_1},\mathrm{t_2}[}(\mathrm{S}))$$

— On peut, par composition, aligner séquentiellement le fragment précédent avec celui d'une autre partition S' (de même schéma) découpée jusqu'à n.

$$\sigma_{t\in[0,n[}(S') \circ \mathrm{map}_{\mathrm{shift_n}}(\sigma_{\mathrm{t}\in[\mathrm{t_1},\mathrm{t_2}[}(\mathrm{S}))$$

— Pour transposer une voix v_i d'une partition (monovoix), on applique avec map la fonction *transpose*.

$$\mathrm{map}_{\mathrm{v_i:tranpose_x}}(\mathrm{S})$$

— La clôture des opérateurs autorise leur composition ; l'expression suivante transpose une voix v_i dans S et la synchronise aux autres voix.

$$\mu_{type(S)-v_i}(S) \oplus \mathrm{map}_{\mathrm{v_i:transpose_x}}(\mu_{\mathrm{v_i}}(\mathrm{S}))$$

— Le résultat d'une expression est, au sens large, une série temporelle dans un espace multi-dimensionnel qui n'est pas forcément et uniquement de nature musicale ; l'expression suivante produit une "partition" synchronisant le rythme (les durées) et le texte d'une monodie de type `[lyrics: dsyll, monody: dsound]`.

$$\mathrm{map}_{\mathrm{d:duration}}(\mu_{\mathrm{monody}}(\mathrm{S})) \oplus \mu_{\mathrm{lyrics}}(\mathrm{S})$$

Il est assez facile de montrer que ScoreAlg permet d'engendrer, à partir d'un seul élément de $\mathcal{E}$(**dsound**) (une note), toutes les partitions possibles et présente donc une forme de complétude pour la notation musicale. L'expressivité précise de l'algèbre reste à formuler, mais la remarque précédente montre qu'elle capture, à un niveau abstrait, les opérations effectuées dans les éditeurs de partitions.

3.2.3. *Extension de l'algèbre relationnelle*

Finalement, pour manipuler les corpus et opus, nous utilisons simplement l'algèbre relationnelle étendue constituée des opérateurs usuels, la sélection σ, le produit cartésien $\times$, l'union $\cup$ et la différence $-$, et d'un opérateur de projection étendue Π. Nous allons nous intéresser à ce dernier.

Définition 10 (Projection étendue, Π) *Soit E une expression relationnelle valide de schéma $(t_1, \cdots, t_m)$.*
Si $e_1, \ldots, e_q$ sont des expressions valides de ScoreAlg*, alors :*

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

est une expression algébrique valide. Sa sémantique sur une instance I est définie par :

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

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

Prenons un exemple concret. Supposons que nous souhaitions extraire, à partir des quartets de Joseph Haydn, le titre et deux partitions, l'une contenant uniquement les voix du premier violon et du violoncelle, l'autre contenant le second violon et l'alto. Nous obtenons :

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

La projection étendue applique des expressions de ScoreAlg aux opus d'une base de données et permet de construire de nouvelles partitions à partir du matériau existant.

3.2.4. *Langage de requêtes*

Afin de permettre à l'utilisateur de manipuler le corpus, il est nécessaire d'avoir un langage d'interrogation de haut niveau. Pour intégrer notre modèle dans notre système, nous souhaitons limiter au maximum l'adaptation d'un quelconque système d'évaluation de requêtes existant. À cette fin, il nous faut choisir entre deux langages pour nous servir de module d'interrogation de base : SQL et XQuery. Nous avons choisi de nous baser sur XQuery qui est adapté à la nature hiérarchique de nos données, et intègre la possibilité d'incorporer des fonctions. Une implantation relativement simple du modèle consiste donc d'une part à modéliser sous forme XML (et XMLSchema) les structures de notre modèle, d'autre part à écrire des fonctions XQuery pour nos opérateurs structurels. Les fonctions utilisateurs peuvent quant à elles également être encapsulées, à partir de bibliothèques externes, sous la forme de fonctions XQuery.

Rappelons par ailleurs que notre langage s'applique à des *instances virtuelles* provenant de notre modèle de données (voix, partitions, opus), et que l'évaluation des requêtes implique un processus de *mapping* pour transformer l'encodage de la représentation physique vers le modèle virtuel.

Les exemples ci-dessous reposent sur le modèle d'un corpus de *Quartet* (cf. les exemples de la section précédente). Tout d'abord, nous créons une liste de l'ensemble des quartets composés par Haydn, réduits aux violons.

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

Rappelons que `music` est un attribut d'une partition de type **Score** dans le corpus des *Quartet*. Cette première requête montre deux opérateurs simples de manipulation de partitions : la projection d'une voix de la partition (opérateur μ) est obtenue de manière standard avec XPath, et la création d'une nouvelle partition par synchronisation (opérateur $\oplus$) s'exprime avec la fonction `Score()`. Le nommage de cette nouvelle partition se fait à l'aide d'un "`as`" de manière standard.

L'expression de l'opérateur map de ScoreAlg implique une extension de XQuery pour appliquer une fonction f aux événements d'une partition, et retourner une

partition résultant de la transformation par f de ces événements. L'exemple ci-dessous transpose la voix d'origine du premier violon ($v1$) pour une clarinette en Sib.

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

Cette deuxième requête montre l'utilisation de la clause standard **let** de XQuery pour associer à un nom de variable le résultat d'une expression de SCOREALG, ici construite syntaxiquement par un appel de fonction MAP. L'opérateur MAP est le moyen privilégié pour produire de nouvelles voix en appliquant des fonctions utilisateurs (UDFs).

Il est bien sûr possible de prendre plusieurs opus en entrée et de produire en sortie un opus contenant plusieurs partitions. L'exemple ci-dessous prend trois chorals identifiées dans la clause *where*, et produit un opus de deux partitions pour lesquelles nous associons respectivement les parties d'alto et de ténor de chaque choral.

```
for $c1 in Chorals, $c2 in Chorals, $c3 in Chorals
where $c1/id="BWV49" and $c2/id="BWV56"
        and $c3/id="BWV12"
return "Extraits des chorals 49, 56, 12" as title,
          Score($c1/music/alto,
                $c2/music/alto,
                $c3/music/alto) as altos,
          Score($c1/music/tenor,
                $c2/music/tenor,
                $c3/music/tenor) as tenors
```

Pour finir, notre dernier exemple étend le concept de partitions à l'aide d'une synchronisation de voix, qui ne sont pas forcément des voix musicales. La requête suivante créer, pour chaque quartet, une partition contenant les voix du premier violon, du violoncelle et d'une troisième voix mesurant les intervalles entre les deux premières.

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

Une telle partition ne peut être représentée dans les méthodes de rendu traditionnelles. Des travaux supplémentaires sont nécessaires pour produire des outils de visualisation capables de mettre en perspective ces fragments musicaux tout en montrant les fonctionnalités analytiques.

4. IMPLANTATION

Nous allons maintenant présenter brièvement l'implantation d'un tel système, en mettant l'accent sur les bénéfices de la réutilisation de composants existants pour limiter l'effort de développement à quelques couches d'intégration. Le lecteur intéressé trouvera des détails dans [8]. Essentiellement, nous nous reposons sur BASEX [1], une base de données dédiée à XML, aussi bien pour le stockage que l'indexation (balises et contenu). Nos partitions encodées en XML (MusicXML, MEI), sont stockées dans BaseX, de même que nos opus, représentés comme des documents XML dans une collection. Ces collections sont maintenues et gérées par la bibliothèque numérique NEUMA[17].

1. http://basex.org

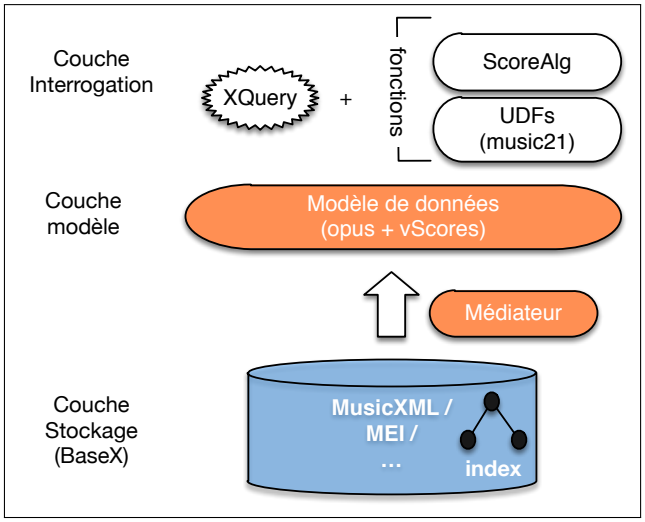


Figure 3. Architecture

BaseX fournit des possibilités d'extension par intégration de fonctions qui correspondent à nos besoins.

La figure 3 présente les principaux modules de notre architecture. La couche basse est constituée les différentes collections de partitions, encodées soit en MusicXML soit en MEI. La couche haute est une interface XQuery, ce dernier associé à un ensemble de fonctions qui correspondent soit à l'implantation des opérateurs de notre algèbre SCOREALG, soit à des UDFs. Dans ce dernier cas, il est tout à fait possible de recourir à des bibliothèques existantes pour éviter de tout réinventer. Actuellement, nous privilégions l'utilisation de l'excellent Music21 [3] pour les UDFs, sans que rien n'empêche d'intégrer d'autres modules.

C'est au niveau de la couche intermédiaire que s'effectue l'essentiel de notre effort pour intégrer le modèle d'interrogation. Rappelons que les requêtes d'interrogation s'appuient sur cette couche "modèle" qui est virtuelle pour ce qui concerne la représentation de la musique notée. Il faut donc, à la volée, transformer l'encodage sérialisé vers l'encodage du modèle abstrait.

Prenons pour être concret l'exemple de la requête suivante qui sélectionne chaque partition du corpus contenant une voix "premier violon" dont la plus haute note est supérieure au 'e6', et sans second violon. La requête produit alors, en sortie, le premier violon transposé à l'octave inférieure.

```
for $s in Quartet
let $tp := Map ($s/music/v1, transpose (-1))
where highest($s/score/v1)  > 'e6'
      and $s/P1/v2 is null
return Score ($tp)
```

L'évaluation de cette requête peut se décrire de la ma-

nière suivante :

1. Dès qu'un accès au sous-arbre (virtuel) `$s/music` est détecté, une conversion (*mapping*) du format de stockage est effectuée pour obtenir une instance du modèle.
2. Les opérateurs structurels, représentés ici par les fonctions *Map()* et *Score()*, s'appliquent à cette instance.
3. Enfin, la requête inclut des appels à des UDFs (ici *transpose()* et *highest()*).

Le principal problème technique est le déclenchement de la conversion au moment approprié. Une possibilité radicale aurait été d'intervenir au niveau du code de BaseX, mais une modification de ce genre est lourde, difficilement maintenable, et peu attractive. Nous avons adopté une solution beaucoup plus légère qui s'appuie sur le fait que l'accès à la *vScore* s'effectue obligatoirement dans le cadre d'un appel à une fonction implantant un opérateur structurel. Ces fonctions testent si la conversion a été effectuée, se chargent de la déclencher si ce n'est pas le cas, et maintiennent en *cache* la *vScore* jusqu'à ce que l'ensemble de l'expression algébrique soit effectuée.

Un processus d'évaluation de ce type pose toutefois des problèmes d'efficacité. Pour une requête, chaque fonction nécessite un accès complet à l'encodage de la partition ; autrement dit, il n'est pas possible d'anticiper les partitions qui seront vraiment traitées dans la requête. Il en résulte que pour chaque appel de fonction de la librairie (ici, *highest()*), l'ensemble du document est récupéré, parsé et envoyé à la bibliothèque. Il est de plus nécessaire de le faire pour chaque partition de la collection, ce qui représente un surcoût inacceptable qui aura pour conséquence de rendre chaque requête totalement inefficace.

Une solution consiste à matérialiser à l'avance le résultat des fonctions utilisées comme critères de recherche, sous la forme de valeurs stockées dans le document XML lui-même. Ces valeurs (comme le résultat du *highest()*) peuvent alors être indexées par le processus standard de BaseX, ce qui évite le coûteux calcul systématique évoqué ci-dessus.

En résumé, nous pensons que l'approche proposée dans cet article, bien que s'appuyant sur une modélisation assez abstraite du contenu des encodages musicaux, peut s'implanter sans efforts excessifs grâce à des systèmes standard de gestion de données et en conservant la représentation "physique" sous forme de documents XML ou autres.

5. ÉTAT DE L'ART

La communauté de la recherche d'information musicale (ou *Music Information Retrieval*) s'est essentiellement intéressée à la recherche non structurée, en particulier la recherche de similarité [20]. Ce type de recherche convient pour les utilisateurs mais évite des considérations complexes relatives à la structure de la notation musicale. Une limite importante de ce type de recherche réside dans la granularité des résultats obtenus, qui restent au niveau documentaire, sans possibilité d'avoir des résultats plus fins (comme des parties des documents). Notre travail permet des interrogations à une échelle plus fine.

Le formalisme proposé présente de nombreux points communs avec des propositions antérieures, dont les travaux de Paul Hudak [12], et d'autres modèles de calculs formels adaptés à la programmation musicale [2, 13, 7, 6]. Euterpea [12] est une approche basée sur la programmation fonctionnelle (avec Haskell) et des types abstraits de données représentant les concepts musicaux. Le système de typage, très riche, et les opérateurs polymorphes d'un langage fonctionnel offrent de larges possibilités de génération d'objets musicaux complexes et d'une étude formelle de leurs propriétés. Dans le même esprit, le T-Calculus présenté dans [13] vise à une modélisation concise et cohérente de flux musicaux représentés sous forme de *tiled streams* et synchronisés par un opérateur de *tiled group*. La notion de points de synchronisation, qui couvre des aspects musicaux importants comme les anacrouses, temps forts, temps faibles, métrique, est prise en compte, alors qu'elle est absente de notre modélisation.

Dans le domaine de la composition musicale, on peut trouver des approches utilisant des opérateurs "musicaux" tout en définissant des modèles se focalisant sur des séries de notes [7] favorisant la représentation interactive de partitions avec INScore [6]. Une autre approche s'intéresse à la définition de grammaires temporelles à base de graphes, tout en utilisant un modèle probabiliste pour favoriser la composition harmonique de partitions [16].

Les points communs avec notre travail sont bien sûr nombreux : les notions d'événements musicaux, de séquences d'événements, de concaténation (que nous avons généralisé en fusion) et synchronisation de séquences sont présents sous des formes diverses. Ce qui rend selon nous notre approche distinctive est la tentative de définir un ensemble restreint et minimal d'opérateurs manipulant uniformément des données de même type (nos *vScore*s). Une telle approche permet d'exprimer de manière concise et uniforme, par composition, des opérations sur des données existantes. Ce type de formalisme, opérant en forme close, avec un pouvoir d'expression très cadré, nous permet d'intégrer à des requêtes des opérations de transformation, extraction, sélection, combinaison sur des objets complexes issu d'un support de stockage. En contrepartie, une telle approche est sans doute faiblement adaptée à une approche plus orientée vers la *génération* de matériel musical. En particulier, la nécessité d'intégrer des fonctions externes pour modifier le matériau existant est une facilité qui rend notre approche moins attractive dans une telle perspective.

Le format *HumDrum* a constitué une approche pionnière pour représenter les partitions musicales comme des fichiers structurés et développer des fonctionnalités de recherche et d'analyse. La représentation comme les procédures y sont de bas niveau (fichiers textes bruts, commandes Unix), ce qui rend l'intégration délicate dans des applications complexes. À notre connaissance, il y a peu d'autres approches. Une tentative de transposer les prin-

cipes des bases de données à la gestion de partitions a été présenté dans [5]. Les auteurs de [9] ont étudié comment XQuery peut être utilisé directement sur le format MusicXML. Cependant, XQuery est un langage de requêtes générique qui s'adapte mal aux spécificités de la notation musicale symbolique. En outre, en ignorant le modèle de données sous-jacent, la fermeture des opérateurs devient indécidable et il manque au langage l'essentiel des propriétés qui le rendrait pleinement utilisable dans des applications de haut niveau.

Nous défendons l'idée qu'il est nécessaire d'identifier précisément le modèle de données qui sous-tend les transformations sur des "partitions". Cela permet en effet de gérer nombre de détails inutiles, apporte de la robustesse à la définition d'opérateurs fermés, et nécessite de vérifier le type de contenu que l'on souhaite manipuler. On peut toujours déplorer, à raison, que l'on perde parfois une partie du sens et que quelques caractéristiques musicales rares (par exemple : des accords avec des durées de notes variables) ne soient pas correctement capturés par un modèle abstrait. Nous avons également pour l'instant ignoré des informations comme la métrique ou la structuration par mesures qui détermine temps forts/faibles et influent fortement sur l'interprétation.

Pour finir, remarquons que notre cadre conceptuel permet d'intégrer des encodages distincts dans un même environnement. Le processus de *mapping* qui permet de le faire se rapproche des architectures de médiation utilisées pour l'intégration de différentes sources de données [4].

6. CONCLUSION

Nous proposons une tentative de traiter la notation musicale comme une source d'information structurée capable de supporter des requêtes et des transformations complexes. Un aspect discutable de cette approche réside dans le *mapping* avec pertes qui extrait l'information de la notation. Il n'y a cependant pas toujours de manière univoque de séparer le contenu d'une partition de sa présentation et cela devra évidemment faire l'objet de travaux futurs.

D'autre part, disposer d'un langage de spécification de haut niveau pour combiner, modifier et générer des partitions offre des perspectives très prometteuses, pour l'interprétation, l'enseignement et l'analyse de la musique. En particulier, nous souhaitons explorer les idées suivantes :

- comment maintenir une synchronisation précise entre les différentes parties d'une partition et l'ensemble complet, afin de refléter chaque modification telle que l'ajout d'une annotation ;
- proposer de nouvelles visualisations de la notation musicale, dictées non par les contraintes d'interprétation mais par le besoin de comprendre un aspect analytique ;
- étudier les mécanismes stylistiques qui peuvent relier une notation musicale abstraite et sa représentation sous forme de partition.

Nous travaillons actuellement sur une implémentation logicielle de notre approche afin d'en démontrer publiquement le potentiel.

Remerciements.. Ce travail est en partie financé par le Music Consortium, `https://humanum.hypotheses.org/503`. Nous remercions par ailleurs chaleureusement les relecteurs de la version soumise pour leurs commentaires et leurs recommandations qui nous beaucoup aidés à tenter d'améliorer cet article.

7. REFERENCES

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

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

[3] Michael Scott Cuthbert and Christopher Ariza. Music21 : A Toolkit for Computer-Aided Musicology and Symbolic Music Data. In *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, pages 637–642, 2010.

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

[5] Zoé Faget and Philippe Rigaux. A Database Approach to Symbolic Music Content Management. In *Intl. Symp. on Exploring Music Contents (CMMR)*, pages 303–320, 2010.

[6] 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.

[7] 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.

[8] R. Fournier-S'niehotta, P. Rigaux, and N. Travers. Querying XML Score Databases : XQuery is not Enough ! Technical Report CEDRIC-16-3612, CEDRIC laboratory, CNAM-Paris, France, 2016.

[9] Joachim Ganseman, Paul Scheunders, and Wim D'haes. Using XQuery on MusicXML Databases for Musicological Analysis. In *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, 2008.

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

[11] Andrew Hankinson, Perry Roland, and Ichiro Fujinaga. The Music Encoding Initiative as a Document-Encoding Framework. In *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, pages 293–298, 2011.

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

[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 *FARM 2013*, pages 23–34, Boston, United States, 2013.

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

[15] Laurent Pugin, Johannes Kepper, Perry Roland, Maja Hartwig, and Andrew Hankinson. Separating Presentation and Content in MEI. In *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, 2012.

[16] Donya Quick and Paul Hudak. Grammar-based automated music composition in haskell. In *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling & design*, FARM '13, pages 59–70, New York, NY, USA, 2013. ACM.

[17] Philippe Rigaux, Lylia Abrouk, H. Audéon, Nadine Cullot, C. Davy-Rigaux, Zoé Faget, E. Gavignet, David Gross-Amblard, A. Tacaille, and Virginie Thion-Goasdoué. The design and implementation of neuma, a collaborative digital scores library - requirements, architecture, and models. *Int. J. on Digital Libraries*, 12(2-3) :73–88, 2012.

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

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

[20] Rainer Typke, Frans Wiering, and Remco C. Veltkamp. A Survey Of Music Information Retrieval Systems. In *Proc. Intl. Conf. on Music Information Retrieval (ISMIR)*, 2005.

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