ÉLARGIR L'ACCÈS À L'INTERPRÉTATION MUSICALE AVEC WEB MIDIFILE PERFORMER

Joseph Larralde, Bernard Paul Serpette, Jean Haury, Raphaël Blard, Myriam Desainte-Catherine

ÉLARGIR L'ACCÈS À L'INTERPRÉTATION MUSICALE AVEC WEB MIDIFILE PERFORMER

Joseph Larralde
SCRIME / LaBRI / Université de Bordeaux
joseph.larralde@u-bordeaux.fr

Bernard Paul Serpette
INRIA Bordeaux
bernardpaulserpette@gmail.com

Jean Haury
SCRIME
jeanhaury@gmail.com

Myriam Desainte-Catherine
LaBRI
myriam@labri.fr

Raphaël Blard
SCRIME
raphael.blard.2023@enseirb-matmeca.fr

RÉSUMÉ

Web Midifile Performer est une application en ligne permettant, à partir d'une ou de quelques touches d'un clavier d'ordinateur ou d'un contrôleur MIDI, l'interprétation expressive de tout morceau de musique pré-encodé dans un fichier au format MIDI. Il s'agit d'une adaptation pour le Web du logiciel *MidifilePerformer*, dont le développement a lui-même été motivé par le *Metapiano* de Jean Haury.

Elle est concrètement formée de deux entités logicielles : *libMidifilePerformer*, une librairie écrite en langage C++ implémentant le modèle et les algorithmes mis en œuvre dans *MidifilePerformer*, et une application Web s'appuyant sur cette librairie pour proposer une interface utilisateur adaptée au public ciblé. La librairie, portable dans de multiples environnements, a également pour vocation d'être déployée en tant qu'extension logicielle de programmes tels que Max/MSP et OSSIA Score, ou intégrée comme système de lecture manuelle dans des logiciels tels que MuseScore. L'application, quant à elle, est accessible depuis un site Internet dédié qui en propose une prise en main progressive.

1. INTRODUCTION

L'application *Web Midifile Performer* et la librairie *libMidifilePerformer* répondent à une volonté de diffuser plus largement les fonctionnalités offertes par le logiciel *MidifilePerformer* [1], écrit à l'origine par Bernard P. Serpette en langage Java et constituant la mise en pratique de l'article théorique *MidifilePerformer : a case study for chronologies* [1]. D'une part, le développement de ces fonctionnalités dans une librairie C++ sous licence premissive [2] favorise leur dissémination sous forme d'extensions logicielles, d'autre part, la création d'une application Web basée sur cette librairie [3] a permis d'offrir un accès direct à ce dispositif d'aide à l'interprétation musicale, et a également représenté l'opportunité de distribuer la librairie pour le Web [4].

Le formalisme décrit dans [1] vise à généraliser les principes du notage pianotechnique [2] définis par Jean Haury et implémentés dans son dispositif, le *Metapiano*. Ce dispositif se compose d'un patch Max/MSP conçu pour fonctionner avec, en entrée, un piano réduit à neuf touches seulement. Les touches agissent comme des commutateurs déplaçant un curseur à travers le morceau pour le réinterpréter selon le rythme, la dynamique et l'articulation désirés.

Cependant, le *Metapiano* se trouve limité par la nécessité pour ses utilisateurs de retranscrire manuellement, avec l'aide d'un utilitaire dédié au notage pianotechnique, tout morceau qu'ils souhaitent intérpréter sous forme de fichier texte que le curseur du logiciel peut ensuite parcourir. *MidifilePerformer*, et par suite *Web Midifile Performer*, ont pour but de résoudre ce problème en permettant l'automatisation de cette retranscription par l'analyse de fichiers MIDI, et de rendre ainsi directement accessible à l'interprétation le vaste répertoire musical constitué par ce type de fichiers, largement disponibles sur le Web et à l'utilisation très répandue dans le domaine de la création musicale.

2. MODÈLE

Nous exposons dans la section 2.1 de manière aussi synthétique que possible le formalisme défini par Bernard P. Serpette dans [1] sur lequel est basé *MidifilePerformer*, puis nous détaillons dans la section 2.2 son implémentation sous la forme d'une librairie C++ réutilisable.

1. https://github.com/scrime-u-bordeaux/MidiFilePerformer
2. https://github.com/scrime-u-bordeaux/libMidifilePerformer
3. https://scrime-apps.labri.fr/web-midifile-performer
4. https://github.com/scrime-u-bordeaux/midifile-performer-js

2.1. Formalisme

2.1.1. Abstractions

L'article [1] introduit notamment les notions abstraites de *triplet modèle-commande-rendu*, d'*évènement*, de *chronologie*, et de *fonction de rendu*.

- Le *triplet modèle-commande-rendu* est le patron de conception selon lequel est structuré l'algorithme d'interprétation : dans notre cas, le *modèle* représente la partition du morceau à jouer, la *commande* représente les actions effectuées par l'utilisateur au travers de l'interface physique, et le *rendu* représente le résultat musical obtenu en croisant le modèle choisi et les commandes fournies.
- L'*évènement* est l'atome permettant de décrire les objets constituant une représentation temporelle. Il peut représenter le début ou la fin d'un tel objet si celui-ci possède une durée, ou son occurrence s'il est instantané. Dans le cas d'un objet m se déroulant sur l'intervalle temporel (t_1, t_2), les évènements correspondant à son début et à sa fin seront respectivement notés $\underline{m}^{t_1}$ et $\overline{m}^{t_2}$. Dans le cas d'un objet m ponctuel, son occurrence à l'instant t sera notée $\dot{m}^{t}$.
- La *chronologie* est un ensemble d'événements successifs ordonnés par dates croissantes, noté sous la forme $\underline{m_1}^{t_1}\underline{m_2}^{t_2}\overline{m_1}^{t_3}\overline{m_2}^{t_4}\dot{m_3}^{t_5}...$, avec $t_n <= t_{n+1}$.
- La *fonction de rendu* est un algorithme prenant en entrée le modèle et la suite des commandes, tous deux décomposés sous forme de chronologies, et produisant, comme son nom l'indique, le rendu en sortie.

Le cadre théorique utilisé par la suite pour décrire et manipuler les chronologies est celui des *S-langages* [3], basés sur des *S-mots*, eux-mêmes constitués de *S-lettres*. Le préfixe *S* signifie *Set* : une *S-lettre* est un ensemble d'évènements simultanés, et un *S-mot* une suite de *S-lettres* ordonnées par dates croissantes.

2.1.2. Application au contexte musical

L'article [1] envisage ensuite l'application de ces notions au cas de l'interprétation expressive de fichiers MIDI et, dans le but de prévenir un comportement contre-intuitif pour un système musical interactif, pose les contraintes suivantes :

- L'ordre des débuts des notes de la partition doit être respecté par la fonction de rendu, mais l'ordre des fins des notes peut être modifié.
- Le relâchement d'une commande ne doit pas déclencher le début d'une note, mais l'enfoncement d'une commande peut déclencher la fin d'une note.
- Chaque geste d'enfoncement doit déclencher au moins le début d'une note, et chaque geste de relâchement doit, dans la mesure du possible, déclencher un évènement.

Notons que les évènements ponctuels, bien que pris en compte par le formalisme, ont ici un rôle secondaire. La commande est exclusivement constituée d'enfoncements et de relâchements de touches. Dans *MidifilePerformer*, les évènements ponctuels du modèle sont traités comme des évènements de fin par commodité. Dans *Web Midifile Performer*, seuls les messages MIDI `NOTE_ON` et `NOTE_OFF` sont intégrés au modèle.

2.1.3. Création du modèle de partition dégradée

Nous allons à présent détailler l'algorithme permettant de générer la chronologie du modèle à partir d'un fichier MIDI, tel que présenté dans l'article [1]. Cette chronologie est parfois qualifiée de *partition dégradée* car certaines informations sont perdues lors de sa création, et le modèle obtenu ne permet pas de rejouer le morceau à l'identique du fichier MIDI original.

Cet algorithme fonctionne en plusieurs étapes, la première consistant à regrouper les évènements synchrones en ensembles d'évènements. La chronologie suivante (illustrée par la figure 1) :

$$\underline{m_1}^{t_1}\underline{m_2}^{t_1}\overline{m_1}^{t_2}\underline{m_3}^{t_2}\dot{m_4}^{t_3}\overline{m_2}^{t_4}\overline{m_3}^{t_4} \tag{1}$$

devient donc :

$$\{\underline{m_1}, \underline{m_2}\}^{t_1}\{\overline{m_1}, \underline{m_3}\}^{t_2}\{\dot{m_4}\}^{t_3}\{\overline{m_2}, \overline{m_3}\}^{t_4} \tag{2}$$

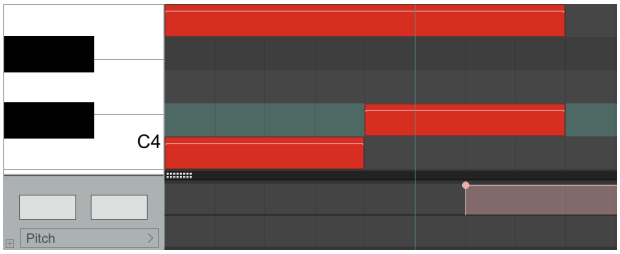

Figure 1. illustration de l'équation 1 sous forme de *piano roll*

Cette opération équivaut à la création d'un S-mot. En définissant comme ensemble de début tout ensemble contenant au moins un début de note, et comme ensemble de fin tout autre ensemble, la même chronologie peut se réécrire sous la forme :

$$\underline{M_1}^{t_1}\underline{M_2}^{t_2}\overline{M_3}^{t_3}\overline{M_4}^{t_4} \tag{3}$$

L'étape suivante consiste à altérer le résultat obtenu pour obtenir une alternance d'ensembles de début et d'ensembles de fin. Pour cela, on introduit des ensembles vides entre les ensembles de début consécutifs et on fusionne les ensembles de fin consécutifs. L'équation 2 devient alors :

$$\{\underline{m_1}, \underline{m_2}\}^{t_1}\{\}^{t'_1}\{\overline{m_1}, \underline{m_3}\}^{t_2}\{\dot{m_4}, \overline{m_2}, \overline{m_3}\}^{t'_2} \tag{4}$$

et peut être exprimée sous la forme :

$$\underline{M_1}^{t_1}\overline{M'_1}^{t'_1}\underline{M_2}^{t_2}\overline{M'_2}^{t'_2} \tag{5}$$

Une dernière étape optionnelle est mentionnée, consistant à éviter les ensembles vides lorsque ceux-ci sont précédés par un ensemble contenant des débuts de notes et suivis par un ensemble contenant les fins des notes correspondantes. Dans ce cas, on migre ces fins de notes dans l'ensemble vide les précédant. En appliquant ce principe, l'équation 4 devient à présent :

$$\{\underline{m_1}, \underline{m_2}\}^{t_1}\{\overline{m_1}\}^{t'_1}\{\underline{m_3}\}^{t_2}\{\dot{m_4}, \overline{m_2}, \overline{m_3}\}^{t'_2} \quad (6)$$

Dans la pratique, cette transformation laisse la possibilité à une articulation encodée comme un *esatto legato* dans un fichier MIDI d'être jouée *staccato* par l'interprète. Elle est implémentée dans *MidifilePerformer* et *Web Midifile Performer* sous l'appellation *unmeet*.

2.1.4. Fonction de rendu

Enfin, l'article [1] propose une étude comparative de différentes fonctions de rendu. Au terme de cette comparaison, la fonction $\texttt{combine}_3$ est identifiée comme celle offrant le plus de possibilités en termes d'expressivité dans l'interprétation musicale, et se trouve par conséquent utilisée dans le code source de *MidifilePerformer* et de la librairie lui faisant suite. Cette fonction de rendu se distingue des autres dans le cas suivant : considérons le modèle de partition $\{\underline{a}\}\{\overline{a}\}\{\underline{b}\}\{\overline{b}\}$, représentant deux notes successives a et b séparées par un silence, que l'on souhaite interpréter à partir de deux touches de commande x et y. Alors $\texttt{combine}_3$ est la fonction de rendu permettant de retranscrire au mieux l'intention gestuelle exprimée par l'interprète. Dans les termes de l'algèbre d'intervalles d'Allen [4] :

- Un geste *staccato* $\{\underline{x}\}\{\overline{x}\}\{\underline{y}\}\{\overline{y}\}$ (relation d'intervalles *Before*) produira un résultat *staccato* $\{\underline{a}\}\{\overline{a}\}\{\underline{b}\}\{\overline{b}\}$.
- Un geste *legato* $\{\underline{x}\}\{\underline{y}\}\{\overline{x}\}\{\overline{y}\}$ (relation d'intervalles *Overlaps*) produira un résultat *legato* $\{\underline{a}\}\{\underline{b}\}\{\overline{a}\}\{\overline{b}\}$.
- Le geste $\{\underline{x}\}\{\underline{y}\}\{\overline{y}\}\{\overline{x}\}$ (relation d'intervalles *During*) produira le résultat escompté $\{\underline{a}\}\{\underline{b}\}\{\overline{b}\}\{\overline{a}\}$.

Ce comportement est obtenu grâce à l'utilisation d'un dictionnaire de correspondance entre les touches de l'interface et les événements de modèle qu'elles déclenchent. Lorsqu'une touche de commande est enfoncée, les évènements d'un ensemble de début sont déclenchés, et les évènements de l'ensemble de fin lui faisant suite dans le modèle sont réservés pour être déclenchés lors du relâchement de cette même touche de commande.

2.2. API C++ générique

2.2.1. Implémentation de l'existant

L'article [1] s'achève par la description de l'architecture *push-pull* choisie pour l'implémentation de *MidifilePerformer*, qui permet de traiter les fichiers MIDI comme des flux d'évènements entrants et laisse ouverte la possibilité d'applications «live». Le concept de chronologie

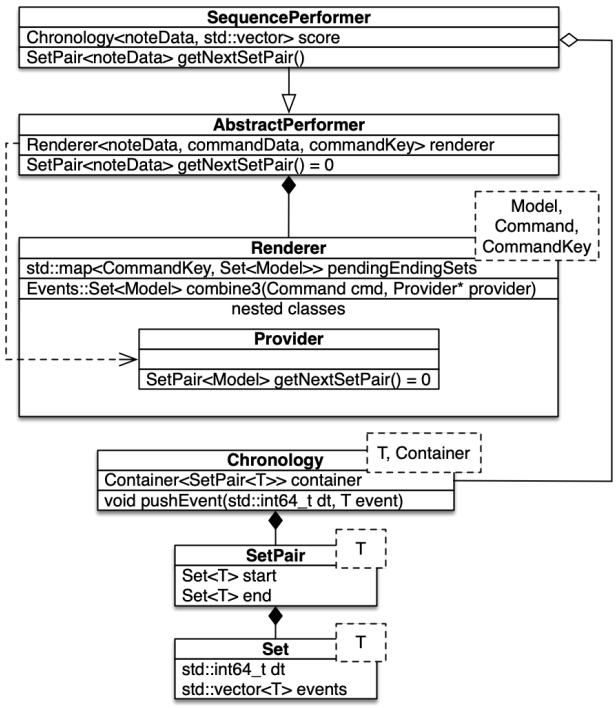

Figure 2. schéma UML simplifié de la librairie C++ *libMidifilePerformer*

y est défini comme une interface abstraite transformant un flux de *S-lettres*, et la création du modèle de partition dégradée est opérée par une chaîne de classes implémentant cette interface, chacune effectuant successivement une des transformations correspondant aux étapes décrites dans la section 2.1.3. La librairie *libMidifilePerformer* fonctionne, pour l'heure, sur la base d'une analyse *post-mortem* uniquement, les fichiers MIDI parcourus n'étant pas modifiables en cours d'interprétation. Cependant son architecture est également prévue pour pouvoir traiter les évènements comme des flux. Elle implémente l'intégralité de la chaîne de chronologies dans une unique classe `Chronology` permettant de spécifier certaines options, notamment l'activation de la transformation *unmeet*, et le modèle de partition est créée itérativement par une succession d'appels à la méthode $\texttt{pushEvent}(\texttt{dt}, \texttt{event})$.

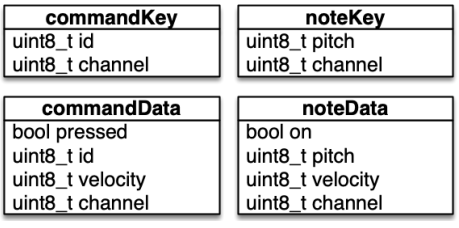

Figure 3. types d'évènements et clés associées utilisés dans *libMidifilePerformer*

Comme résumé dans le schéma UML simplifié de *libMidifilePerformer* (figure 2), les concepts abstraits d'*ensemble d'évènements* (*S-lettre*), de *chronologie* (*S-mot*) et de *fonction de rendu* exposés dans la section 2.1.1

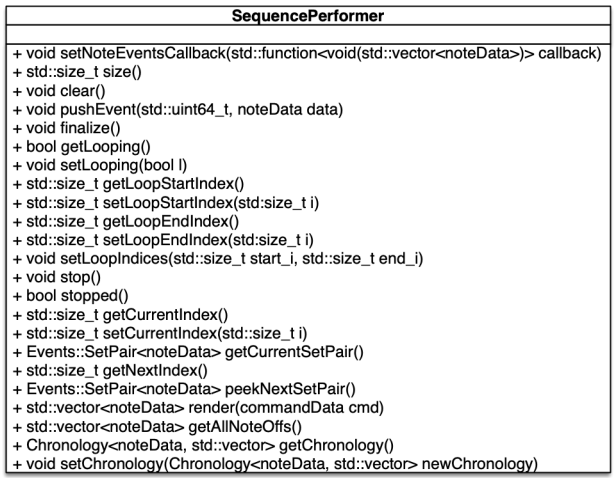

Figure 4. API de *libMidifilePerformer* : méthodes publiques de la classe `SequencePerformer`

sont définis comme des patrons de classes. La classe abstraite `AbstractPerformer` définit le fonctionnement de base de la librairie en spécifiant les types d'évènements `commandKey`, `commandData`, `noteKey` et `noteData` mis en œuvre (figure 3), et en introduisant la fonction virtuelle `getNextSetPair`. Il est ensuite possible de créer des classes dérivées d'`AbstractPerformer` selon différentes approches afin de fournir des API adaptées aux cas d'usage. Dans notre cas, la classe `SequencePerformer` implémente l'analyse *post-mortem* en utilisant une classe `Chronology` basée sur un conteneur de type `std :: vector`, et fournit un ensemble de méthodes publiques sur lesquelles s'appuie *Web Midifile Performer*. Une API alternative orientée «live» pourrait par exemple être implémentée sous l'appellation `StreamPerformer`, mais le cas d'usage étant encore mal défini, cette tâche est pour le moment non prioritaire. Il est également possible qu'en recueillant suffisamment de retours utilisateurs, la spécification d'une API plus «universelle» puisse un jour émerger, l'architecture de la librairie devant alors être partiellement repensée.

2.2.2. Fonctionnalités additionnelles

La classe `Chronology` conserve un horodatage des ensembles d'évènements issu de l'analyse du fichier MIDI le plus proche possible de celui des messages originaux, sans introduire de dérive temporelle, et les types `noteData` et `commandData` conservent les vélocités originelles des messages `NOTE_ON`. Tout ceci permet non seulement de rejouer le morceau au plus proche de la version originale (seuls les évènements de fins de notes successifs sont groupés en un seul évènement), mais également de décider de la stratégie à adopter pour le choix des vélocités à appliquer individuellement aux notes d'un accord en fonction de la vélocité de la commande, ce qui est implémenté dans la classe `ChordVelocityMapping` selon un *design pattern* de *Stratégie*.

La volonté d'offrir la possibilité de travailler certains passages en boucle a également donné lieu à l'ajout de nouvelles fonctionnalités. Tout d'abord, la classe `AntiVoiceStealing` a été créée pour résoudre le cas dans lequel une note serait déclenchée deux fois avant d'être arrêtée, ce qui est rendu possible par l'utilisation de $\texttt{combine}_3$. Prenons le cas simple d'une chronologie correspondant à un mordant joué *legato* sur les notes a et b. Les transformations successives appliquées pour obtenir une alternance d'ensembles de début et d'ensembles de fin produisent la chronologie suivante : $\{\underline{a}\}\{\}\{\underline{b}\}\{\overline{a}\}\{\underline{a}\}\{\overline{b},\overline{a}\}$. Si l'on interprète ce mordant avec le geste original *legato* ayant produit cette chronologie, à savoir $\{\underline{x}\}\{\underline{y}\}\{\overline{x}\}\{\underline{x}\}\{\overline{y}\}\{\overline{x}\}$, on obtiendra en sortie le résultat suivant : $\{\underline{a}\}\{\underline{b}\}\{\}\{\underline{a}\}\{\overline{a}\}\{\overline{b},\overline{a}\}$. Cette suite d'évènements, une fois traduite en messages MIDI `NOTE_ON` et `NOTE_OFF` et transmise à un instrument MIDI, va jouer a, puis b, puis jouer à nouveau a sans avoir arrêté son occurrence précédente, puis arrêter a au relâchement de y, et enfin tenter de l'arrêter à nouveau en même temps que b au second relâchement de x, ce qui, suivant le comportement de l'instrument MIDI, n'aura potentiellement aucun effet. On s'attendrait plutôt à obtenir la sortie suivante : $\{\underline{a}\}\{\underline{b}\}\{\}\{\overline{a}\,\underline{a}\}\{\}\{\overline{b},\overline{a}\}$. Ce comportement, plus intuitif car une touche relâchée n'arrête pas une note déclenchée par une touche enfoncée plus récemment, est assuré par l'utilisation de la classe `AntiVoiceStealing`. Cette classe garde en mémoire le compte des occurrences actives d'une même note. Elle incrémente le compte et envoie automatiquement un message `NOTE_OFF` avant le message `NOTE_ON` produit par toute nouvelle occurrence, néglige l'envoi de tous les messages `NOTE_OFF` qu'elle devrait envoyer et décrémente à la place le compte tant que celui-ci est supérieur à 1, puis autorise enfin l'envoi du dernier message `NOTE_OFF` lorsque le compte est égal à 1 avant de le décrémenter. Cela permet de généraliser le comportement souhaité sur l'exemple du mordant à n'importe quel nombre de notes identiques simultanées.

Ce principe a ensuite été étendu pour autoriser l'interprétation en boucle de n'importe quelle partie d'un morceau ainsi que la modification des points de bouclage en cours d'interprétation sans se retrouver confronté au même type de problème. L'analyse *post-mortem* permet de connaître à tout moment de l'exécution d'un morceau quelles notes sont censées être actives et prédire ainsi l'apparition d'évènements de fins de notes devant être ignorés si on ne commence pas l'interprétation du morceau par son début. Cette suite d'états a été implémentée sous la forme d'un vecteur de dictionnaires nommé `avsHistory` (pour `AntiVoiceStealing` history). En combinant l'état de la classe `AntiVoiceStealing` et l'état du dictionnaire à l'indice correspondant à un endroit précis du morceau, on obtient la possibilité de tenir une note de fin de boucle déclenchée par une certaine touche jusqu'au relâchement de cette touche, tout en prévenant les fins de notes indésirables après avoir sauté à un autre endroit du morceau. Comme nous le verrons par la suite, l'interface de l'application Web restreint ces possibilités car les points de bouclage doivent être modifiés manuellement avec la sou-

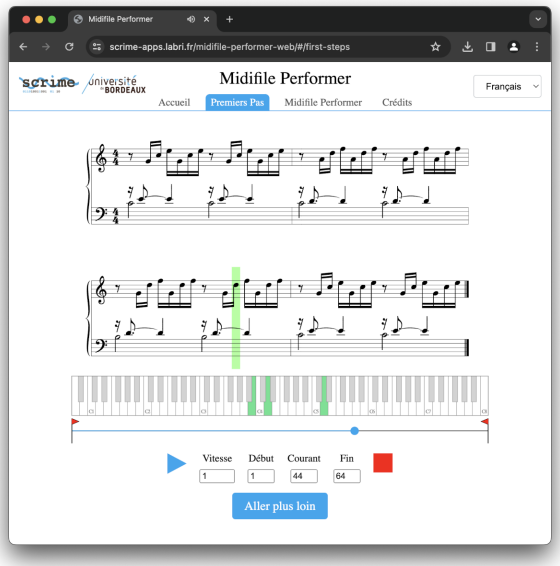

Figure 5. Interface de familiarisation avec l'application

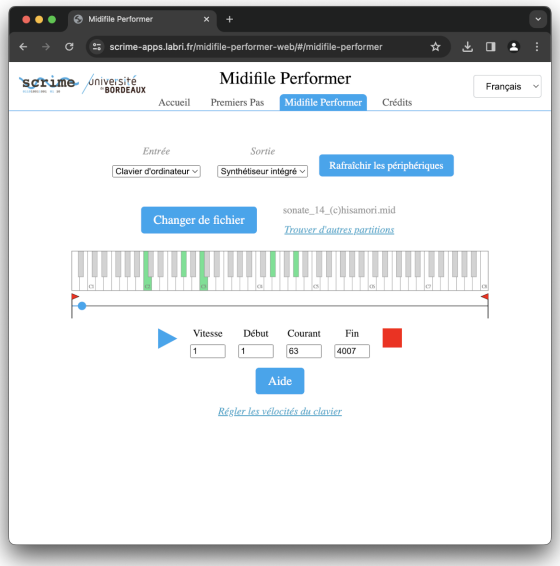

Figure 6. Interface d'utilisation libre

ris, mais ce principe serait capable de garantir une cohérence de comportement totale si l'on envisageait une gestion plus algorithmique des sauts dans une partition, ou même d'une partition à une autre, moyennant quelques modifications dans le code source.

L'API de la librairie est déterminée par les méthodes publiques de la classe `SequencePerformer` (Figure 4). Il est également possible de manipuler directement des instances de la classe `Chronology`.

3. APPLICATION WEB

En compilant la librairie C++ *libMidifilePerformer* vers WebAssembly, un paquet NPM [5] est généré qui permet son usage dans un environnement Web, moyennant l'ajout d'une couche d'API supplémentaire définissant des classes ESM5 servant d'abstraction aux entités C++ sous-jacentes. Ce paquet, distribué sous licence BSD-3-Clause, sert alors de base à l'application *Web Midifile Performer*.

3.1. Interface

Web Midifile Performer étant un site Internet, son interface physique première est le clavier d'ordinateur. Celui-ci peut faire place à tout type de contrôleur MIDI capable de produire des événements `NOTE_ON` et `NOTE_OFF`, tout particulièrement les claviers. C'est au moyen de ces entrées physiques que l'utilisateur peut émettre des commandes, par l'intermédiaire de l'API WebMidi prise en charge par la plupart des navigateurs à l'exception notable de la version iOS de Safari.

L'interface utilisateur visuelle de *Web Midifile Performer* se divise quant à elle en deux onglets : un onglet d'entraînement ou de prise en main (figure 5), où le morceau est une courte séquence fournie avec l'application (constituée des huit premières mesures du Prélude en do majeur de Bach), et un onglet «libre» (figure 6), où l'utilisateur peut interpréter un fichier MIDI de son choix en le déposant sur la page du site.

L'interface graphique offre alors les éléments suivants pour le support visuel de la performance musicale :

- Une barre de lecture du morceau, permettant son écoute passive afin de se familiariser avec son rythme de jeu ;
- Un piano dont les touches s'éclairent lorsque la note correspondante est jouée ;
- Dans le cas de la page de prise en main, la partition de la séquence pré-chargée, accompagnée d'une visualisation du curseur d'interpétation au sein de celle-ci.
- Dans le cas de la page «libre», la possibilité de régler la vélocité associée à chaque rangée de touches du clavier d'ordinateur

3.2. Fonctionnalités

Une fois le fichier MIDI à interpréter fourni par l'utilisateur, le clavier, ou tout autre appareil de commande dont il souhaite faire usage, deviennent actifs. Tout appui sur l'une de leurs touches provoque l'avancée du curseur d'un pas, déclenchant l'ensemble de début ainsi rencontré, et tout relâchement déclenche l'ensemble de fin correspondant. C'est ainsi le rythme des commandes qui dicte le rythme de l'interprétation, et l'ordre des relâchements qui dicte l'articulation des notes. La vélocité (ou force) du jeu dépend, quant à elle, de la force d'appui, émulée dans le cas d'un clavier d'ordinateur par un slider de vélocité manuellement réglable.

5. `https://www.npmjs.com/package/midifile-performer`

L'interprétation immédiate d'un morceau, même connu, est cependant difficile dans un premier temps ; en effet, le principe de jeu précédemment décrit s'oppose au paradigme classique d'un clavier, où chaque touche ne produirait bijectivement qu'une seule note. Ainsi, pour permettre à l'utilisateur d'apprivoiser notre système, la lecture passive du morceau et son interprétation active peuvent se succéder sans transition.

A l'appui de la barre d'espace ou du bouton de lecture de l'interface, la lecture passive a lieu ; tout appui interrompt cette lecture et déclenche le prochain ensemble (ou maintient l'ensemble actuel dans le cas où il n'aurait pas été joué assez longtemps, afin d'obtenir une expérience empiriquement fluide). Par usage successif de lecture et d'interprétation, l'utilisateur peut alors apprendre le rythme de jeu nécessaire à la reproduction du morceau, et se servir ensuite de cette base pour filer à partir d'elle l'interprétation désirée.

Pour faciliter l'étude du morceau, la lecture passive peut être ralentie ou accélérée (l'interprétation ne changeant aucunement de rythme, puisque celui-ci n'est dicté que par la commande), mais le morceau étudié peut également être réduit à une section délimitée par deux marqueurs visuels de la barre de lecture, prenant la forme de deux drapeaux. Lecture et interprétation décriront alors une boucle dans la section choisie.

4. USAGES

Le mode de jeu inhabituel présenté dans cet article se résume à manipuler une ou quelques touches d'un clavier d'ordinateur, d'un clavier MIDI ou d'un autre contrôleur MIDI pour interpréter la rythmique, la dynamique, l'articulation, le tempo et l'expression d'un fichier de notes d'une partition ou d'une œuvre. Cette approche du jeu a été développée sur le *Metapiano* et le *Bao-Pao* de La Puce à l'Oreille [6] pendant plus de 20 ans pour concilier les exigences de virtuosité d'un côté et les défis liés au handicap de l'autre. Ainsi, le *Bao-Pao* a permis de dépasser les limites physiques, offrant à tous l'accès à l'exécution et à l'interprétation musicale. Sur le *Metapiano*, le musicien professionnel se trouve dans l'instant en situation de performance d'une pièce musicale. Il peut expérimenter toutes sortes d'interprétations avec la virtuosité requise.

Entre ces deux extrêmes de performance, le SCRIME a ouvert un volet pédagogique sur plusieurs années. Il a été offert aux élèves du primaire l'opportunité de déchiffrer à vue et d'interpréter immédiatement une courte partition. Sur l'écran d'un vidéoprojecteur, la partition imprimée est projetée, avec un curseur mobile suivant précisément les mouvements de l'enfant sur un clavier réduit à 9 touches. En utilisant une seule touche, l'enfant n'a plus besoin de fixer des yeux le clavier, concentrant ainsi toute son attention sur la partition, les notes, les indications expressives et le point précis de sa progression.

Ces différentes pratiques d'interprétation au *Metapiano* et au Bao-Pao ont nécessité la création de vastes

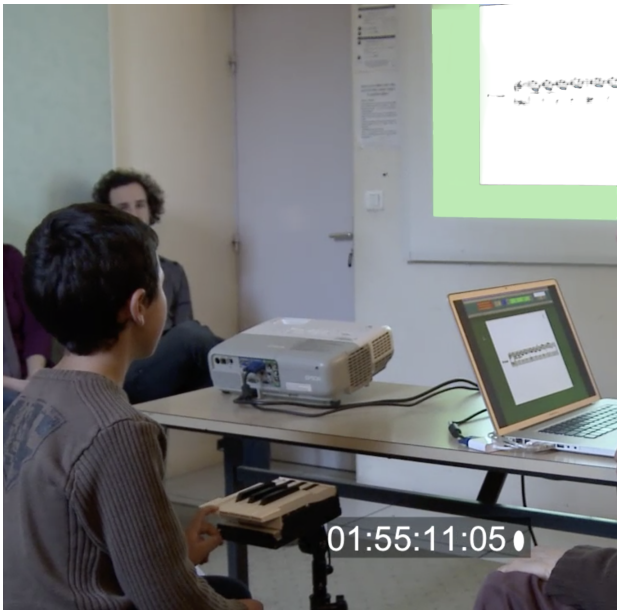

Figure 7. utilisation du Métapiano comme outil de déchiffrage de partitions

répertoires de musique qui soient adaptés au handicap, à la pédagogie, à la virtuosité. Les partitions choisies ont été codées, numérisées manuellement en fichiers interprétables selon les principes pianotechniques cités précédemment. Parallèlement, sur le Web se sont développés des sites dédiés aux partitions du répertoire de musique classique. Ainsi, pour ne citer qu'eux, IMSLP [7] propose de télécharger gratuitement des fichiers en format PDF de partitions imprimées et Kunstderfuge [8] de télécharger gratuitement les partitions sous la forme de fichiers MIDI. Ces plateformes proposent actuellement l'accès à l'ensemble du répertoire musical gratuitement. Lire et suivre une partition téléchargée en PDF tout en écoutant cette même partition en fichier MIDI exécutée automatiquement est aisé. Il était tentant d'ajouter à la lecture et l'écoute, la pratique et l'interprétation d'une pièce. C'est l'objectif de *Web Midifile Performer*.

La profusion de contrôleurs MIDI augmente les possibilités d'interactions avec le système. Par exemple, dans une expérience avec Pierre Lacroix, de l'équipe Systèmes et Réseaux du LaBRI [9], nous avons interfacé le système avec un pédalier d'orgue MIDI. Cette expérience nous a permis d'ajuster quelques paramètres de la fonction de rendu comme, notamment, la possibilité d'ignorer l'enfoncement d'une touche si il se produit relativement synchrone avec l'enfoncement d'une touche à proximité sur le pédalier. L'usage des pieds est plus délicat que celui des doigts et il arrivait fréquemment que deux touches soient enfoncées simultanement. Ainsi, l'ordinateur peut servir à gommer certaines difficultés relevant de la dextérité.

6 . https://www.bao-pao.com

7 . https://imslp.org/

8 . https://www.kunstderfuge.com/

9 . Laboratoire Bordelais de Recherche en Informatique

5. CONCLUSION ET PERSPECTIVES

5.1. Vers un lecteur manuel générique de fichiers MIDI

Le format MIDI est un standard de la production musicale, pris en charge dans virtuellement tous les environnements du domaine. Il existe de très nombreuses archives de fichiers MIDI ainsi qu'une large gamme de logiciels comprenant une fonction de lecture passive desdits fichiers.

Dans chaque environnement de ce type, il serait ainsi possible d'utiliser *MidifilePerformer*. Une fonction d'import de fichier à partir d'une URL pourrait grandement faciliter la prise en charge par *Web Midifile Performer* de morceaux disponibles en ligne ; et grâce à la bibliothèque *libMidifilePerformer*, des plugins pourraient être développés pour de multiples logiciels extensibles, avec comme objectif le plus immédiat MuseScore, permettant l'accès direct depuis l'interface des logiciels en question à toutes les fonctionnalités d'interprétation des autres versions du *MidifilePerformer*.

5.2. Affichage de partitions

Actuellement, *Web Midifile Performer* ne permet pas de générer de manière automatisée une représentation graphique d'un fichier MIDI sur laquelle il serait possible de faire défiler un curseur par évènements successifs, comme c'est le cas avec la partition des huit premières mesures du prélude de Bach dans l'onglet «premiers pas» de l'application. Dans ce dernier, la partition a été générée au format SVG grâce à l'éditeur de partitions en ligne Guido [5], puis intégrée manuellement dans l'application. Les versions récentes de MidifilePerformer offrent une fonctionnalité de curseur défilant sur une représentation graphique de type *piano roll*, ce qui sera implémenté prochainement dans *Web Midifile Performer* car la génération d'une telle représentation à partir d'un fichier MIDI est relativement simple.

Par ailleurs, les auteurs souhaitent s'inscrire dans la dynamique Music Encoding Initiative [6], et visent à moyen terme l'implémentation d'un module permettant, à partir d'un fichier au format MusicXML, de générer simultanément la partition avec curseur défilant et la représentation au format MIDI du morceau de musique encodé dans ce fichier. Cela permettra d'automatiser la création de représentations similaires à l'onglet «premiers pas» sur l'ensemble des morceaux disponibles en format MusicXML. Les morceaux disponibles uniquement en format MIDI pourront toujours bénéficier d'un affichage de type *piano roll*. Pour atteindre ce but, on évaluera le potentiel d'applications web musicales open source telles que Dezrann [7] ou CosmoNote [8] à pouvoir fournir des composants de code ré-utilisables dans notre cadre.

5.3. Déploiement logiciel

Comme évoqué dans le résumé, la librairie C++ *libMidifilePerformer* a pour vocation non seulement d'être intégrée comme principe universel de lecture manuelle de fichiers MIDI dans tout programme en proposant déjà une lecture automatique, mais également d'être déployée dans une variété de briques logicielles. Les environnements ciblés à court terme sont principalement les environnements de programmation visuelle Max/MSP et Pure Data, ainsi que le séquenceur intermédia OSSIA Score pour lequel un prototype fonctionnel existe déjà. A cette fin, le framework C++ déclaratif Avendish [9] est actuellement en cours d'évaluation. La création d'un plugin de type VST, qui pourrait intégrer l'affichage et le suivi par curseur de partitions au format MusicXML dans son interface, est également envisagée.

5.4. Evolutions possibles

Le *Metapiano*, *MidifilePerformer* et *Web Midifile Performer* s'incrivent dans la lignée des outils d'aide à l'interprétation musicale tels que le Radio Baton [10] ou le Digital Baton [11]. Ils requièrent cependant l'exécution d'un geste par évènement musical (note ou groupe de notes), et sont donc destinés à être pilotés par des gestes de musicien plutôt que par des gestes de chef d'orchestre. Les derniers développements de *MidifilePerformer* explorent la piste d'un contrôle gestuel de plus haut niveau permettant de s'affranchir de cette contrainte mais n'ont pas encore été publiés au moment de la rédaction de cet article.

6. REMERCIEMENTS

Ces travaux de recherche se sont déroulés au sein de la plateforme de recherche SCRIME de l'Université de Bordeaux (Studio de Création et de Recherche en Informatique et Musiques Expérimentales) et ont été financés par le ministère de la culture.

7. REFERENCES

[1] Chabassier, J., Desainte-Catherine, M., Haury, J., Pobel, M., and Serpette, B. P. "MidifilePerformer : a case study for chronologies", *26th ACM SIGPLAN International Conference on Functional Programming (ICFP '21)*, pp. 13-22, South Korea, 2021

[2] Haury, J. "La pianotechnie ou notage des partitions musicales pour une interprétation immédiate sur le métapiano.", *Journées d'Informatique Musicale*, 2009

[3] Durand, I., and Schwer, S. R. "A Tool for Reasoning about Qualitative Temporal Information : the Theory of S-languages with a Lisp Implementation.", *J. Univers. Comput. Sci., Vol. 14, no 20*, pp. 3282-3306, 2008.

[4] Allen, J. F. "Towards a general theory of action and time.", *Artificial intelligence, Vol. 23, no 2*, pp. 123-154, 1984.

[5] Daudin, C., Fober, D., Letz, S., and Orlarey, Y. "The Guido Engine-a toolbox for music scores rendering." *Linux Audio Conference*, pp. 105-111, 2009

[6] Roland, P. "The music encoding initiative (MEI)", *Proceedings of the First International Conference on Musical Applications Using XML, Vol. 1060*, 2002

[7] Giraud, M., Groult, R., and Leguy, E. "Dezrann, a web framework to share music analysis.", *International Conference on Technologies for Music Notation and Representation (TENOR 2018)*, pp. 104-110, 2018

[8] Fyfe, L., Bedoya, D., and Chew, E. "Annotation and Analysis of Recorded Piano Performances on the Web.", *Journal of the Audio Engineering Society, Vol. 70, no 11*, pp. 962-978, 2022.

[9] Celerier, J-M. "Rage Against The Glue : Beyond Run-Time Media Frameworks with Modern C++", *Proceedings of the International Computer Music Conference (ICMC)*, Limerick, Ireland, 2022.

[10] Mathews, M. V. "The radio baton and conductor program, or : Pitch, the most important and least expressive part of music.", *Computer Music Journal, Vol. 15, no 4*, pp. 37-46, 1991

[11] Marrin, T., and Paradiso, J. A. "The Digital Baton : a Versatile Performance Instrument.", *ICMC*, September 1997