Université Pierre et Marie Curie

Master Informatique, Spécialité STL

Semestre 2, UE Projet STL

Un analyseur syntaxique pour MusicXML

Auteurs :
Sébastien DUCHENNE
Alexandre GASPARD CILIA

Encadrant :
Pr. Carlos AGON

Soutenue le 22 mai 2017

Table des matières

1 Remerciements

Carlos Agon : idées et conseils pour la réalisation du projet, cours de musique et sur le fonctionnement de OpenMusic, relecture du rapport, visite de l'IRCAM, découverte d'un domaine qui était totalement obscur pour nous

Karim Haddad : explications sur la musique et OpenMusic

Table des figures

2 Introduction

L'IRCAM [1], Institut de Recherche et Coordination en Acoustique/Musique, est un centre de création et de recherche scientifique sur la musique. Il a été fondé en 1969 par Pierre Boulez à la demande du président Georges Pompidou.

En 1995, le CNRS et le ministère de la Culture et de la Communication s'associe et créée l'UMR 9912 STMS. Cette unité mixte de recherche, hébergée à l'IRCAM, s'intéressent aux sciences et aux technologies de la musique et du son. En 2010, elle est rejoint par l'UPMC.

Cette UMR est composé de nombreuses équipes de recherche. L'une d'elles s'intitule "Représentation musicale", et réalise des outils de compositions musicales. Elle a notamment créée OpenMusic [2], un environnement de composition musicale assisté par ordinateur.

Les chercheurs de l'IRCAM souhaitent travailler sur des partitions musicales sous formes d'arbres rythmiques. Actuellement, ils utilisent OpenMusic. Cependant, ce logiciel — (mettre inconvénients) —. C'est pourquoi, ils souhaitent une nouvelle version.

Ce programme sera écrit en langage Java et sera similaire à OpenMusic. Il permettra donc d'éditer graphiquement des morceaux de musique. Son implémentation comportera différents modules, dont celui consistant à construire les arbres rythmiques à partir d'un fichier au format MusicXML. C'est ce module que notre tuteur de projet Carlos Agon, enseignant-chercheur à l'UPMC et membre de l'équipe de représentation musicale, nous a demandé de réaliser.

Le module que nous avons développé est déposé sur un compte GitHub [3] public. Il est donc open source. Il a été réalisé en anglais pour favoriser son internationalisation.

3 Technologies utilisées

3.1 OpenMusic

OpenMusic [2] est un langage de programmation visuel basé sur le langage Lisp qui permet d'écrire graphiquement des compositions musicales. Il a été conçu par les chercheurs de l'IRCAM Carlos Agon, Gérard Assayag et Jean Bresson. Les programmes sont constitués d'éléments reliés entre eux et représentant des structures de données ou des fonctions. La figure suivante montre l'interface du logiciel.

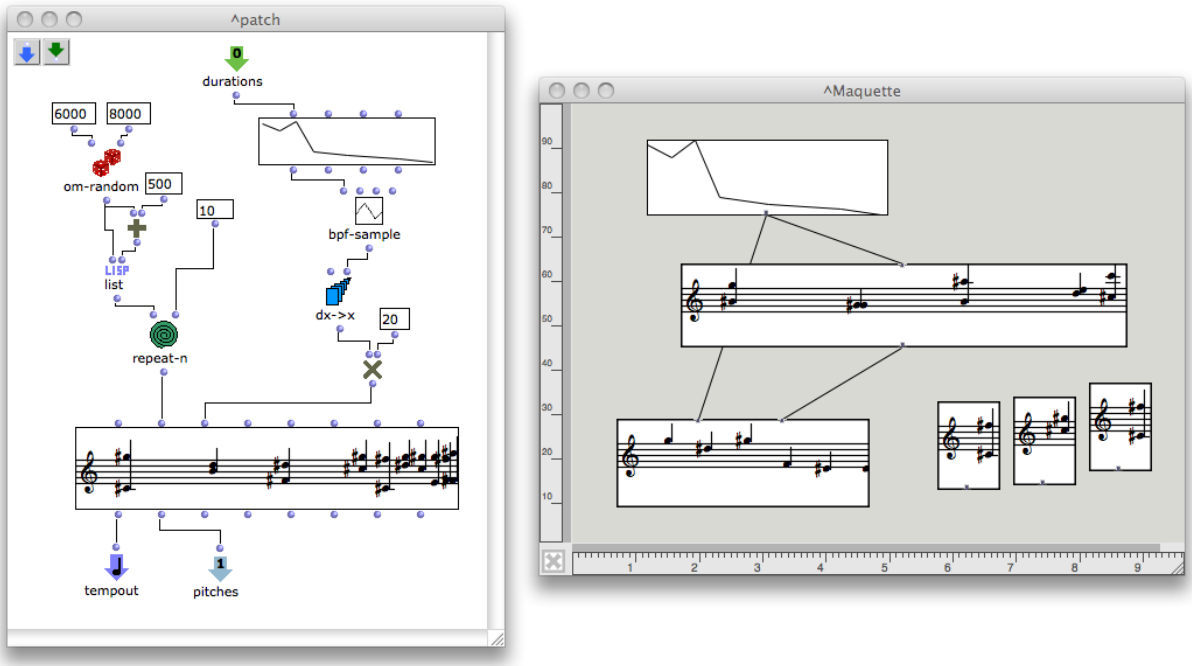


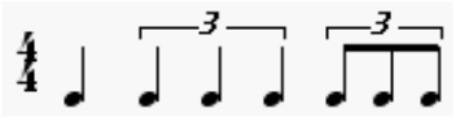
FIGURE 1 – OpenMusic

Dans ce logiciel, les morceaux de musique sont constitués d'une suite d'arbres rythmiques. Comme décrit dans l'article [4], « un arbre rythmique est défini comme un couple (D S) où D est une fraction (< 0) et S est une liste de n-éléments définissant n-proportions de D. Chaque élément de S peut-être soit un entier, soit un arbre rythmique. »

Les images suivantes sont des exemples d'arbres rythmiques, en haut, et la partition correspondante, en bas.

(3/4 (1 1 1))

(4/4 (1 (2 (1 1 1)) (1 (1 1 1))))



3.2 Le langage Java

Java [5] est un langage de programmation **orienté objet** fortement typé développé par **Sun Microsystems** à partir de **1995**. La société sera plus tard rachetée par **Oracle** en 2009 qui possède et maintient Java encore aujourd'hui.

Java se détache de la masse des autres langages de programmation notamment grâce à sa portabilité et sa facilité d'utilisation.

Listing 1 – Hello world en java

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Ci-dessus, un classique "Hello world" en Java. Nous pouvons y voir la définition de la classe *HelloWorld* ainsi que la méthode principale du programme nommé *main* effectuant un affichage sur la sortie standard.

3.3 Le langage XML

Le langage XML [6, 7], acronyme de e**X**tensible **M**arkup **L**angage, est langage de balisage générique spécifié par le W3C. Il permet de définir différents espaces de noms, c'est à dire des langages avec leur propre grammaire et vocabulaire. Il permet l'échange d'information entre des programmes très différents à condition d'utiliser la même grammaire.

Il a l'avantage de pouvoir être compris par les êtres humains et les machines. Cependant, c'est un langage verbeux et qui peut donc prendre beaucoup de place s'il contient beaucoup d'information.

Un document XML est constitué de balises pouvant contenir d'autres balises ou une valeur simple. Une balise peut aussi contenir des attributs donnant des informations supplémentaires sur le contenu.

Listing 2 – Exemple d'un document XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<racine>
    <balise attribut="valeur">Contenu</balise>
    <baliseunique />
</racine>
```

Ci-dessus un exemple de XML simpliste mais qui met en avant les bases du langage. La première ligne annonce le type de document et la version dans lequel le document va être rédigé. *<racine>* est le nœud racine du document, celui qui en somme va contenir tout le document. On peut ici, facilement remarquer que le document peut être représenté sous la forme d'un arbre.

XML permet à l'utilisateur de définir lui même la grammaire de son document grâce notamment aux **DTD** et au **Schéma XML**. Ces outils nous permettent de disposer de format d'échange de données tel que **MusicXML**.

3.4 Le format MusicXML

MusicXML [8] est un format de fichier permettant de représenter la notation musicale occidentale (notation classique, accords en notation anglo-saxonne, tablatures et percussions). Basé sur le langage XML, il est propriétaire mais il peut librement être utilisé avec une licence publique.

Dans ce projet, nous nous intéresserons aux différents symboles de la notation classique, à savoir les éléments de base, les ornements, les articulations et les répétitions. Nous nous sommes documentés sur cette notation [9] afin de bien comprendre les différentes balises du format.

Il y a plus de 20 ans, le format MIDI était très utilisé. Cependant, il n'est pas très adapté pour représenter toutes les caractéristiques de la musique, on perd donc en informations avec ce format. Pour pallier à cela, les formats SMDL et NIFF ont été créés. Cependant, le format SMDL était complexe et donc peu compréhensible. Il était donc très peu utilisé. Le format NIFF était un format peu pratique à utiliser et n'a donc pas été adopté par certains logiciels. Ces formats n'ont donc pas eu le succès souhaité.

En 2004, la société Recordare LLC s'inspire des 2 formats universitaires MuseData et Humdrum pour créer la première version du format MusicXML. Ses avantages sont qu'il est facile à manipuler. Il permet le transfert de morceaux de musique d'une application à une autre. Il peut représenter beaucoup de caractéristiques de la musique. Cependant, il est verbeux, puisqu'il utilise le format XML, et ne donc permet pas de représenter la musique non occidentale.

Il est de plus en plus utilisé puisque plus de 200 logiciels de musique l'ont adopté. Il est donc possible de travailler finement sur un morceau de musique en utilisant différents programmes.

Comme le format XML est verbeux, le fichier prend de la place. La version 2.0, sortie en 2007, apporte donc la compression du fichier au format xml en un fichier au format mxl, et permet de diviser sa taille de façon importante. La version 3.0, sortie en 2011, permet le support des instruments virtuels.

On voit, dans le code correspondant à la partition suivante, que les informations sur la

partition sont placées dans la balise "measure" et celle concernant la ronde sont contenues dans la balise "note".

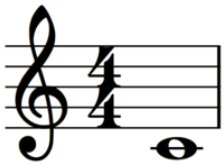


FIGURE 2 – Hello World en MusicXML

Listing 3 – Document XML d'un Hello World en MusicXML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD␣MusicXML␣3.0␣Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.0">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

3.5 Le langage Relax NG

Relax NG (**Re**gular **La**nguage for **X**ML **N**ext **G**eneration) [10] née de la fusion de TreX de James Clark et de Relax de Murata Makoto. C'est un langage qui permet de définir la grammaire d'un document XML. Relax NG ne s'intéresse qu'à la structure du document et non à sa valeur.

C'est ce que nous utiliserons afin de s'assurer de la validité du document à traiter.

3.6 Les librairies

Dans cette section, nous aborderons les librairies utilisées pour réaliser ce projet. Cela ira de la validation du XML en passant par le parsing de ce dernier, jusqu'à la récupération des information stockées dans le fichier MusicXML.

3.6.1 Trang et Jing

Trang [11] et **Jing** [12] sont deux librairies développées par **Thai Open Source**. Elles permettant de générer des grammaires Relax NG et de valider des documents XML à partir de cette même grammaire.

Trang est une librairie qui permet de traduire un fichier de description grammaticale en fichier Relax NG. En effet XML n'est pas facilement lisible pour un esprit humain, c'est pour cela que Trang nous permet de créer notre grammaire dans un langage plus compréhensible. Une fois la grammaire écrite dans un fichier en *.rnc*, nous pouvons générer notre fichier Relax NG en *.rng*.

Jing, quant à lui, est une librairie Java qui permet de valider un document XML à l'aide d'un fichier Relax NG.

Listing 4 – Code java permettant de vérifier la validation d'un document XML

```
final ValidationDriver vd = new ValidationDriver();
vd.loadSchema(rngFile);

if (!vd.validate(inputTextStream)) {
        throw new ParseException("Invalid xml :(");
} else {
    System.out.println("Valid xml :)");
}
```

Le code ci-dessus est une utilisation simplifiée de Jing. Nous commençons tout d'abord par créer un objet *ValidationDriver* de Jing dans lequel nous chargeons notre fichier Relax NG. Nous n'avons ensuite plus qu'à lancer la méthode *validate(InputSource in) : boolean* qui nous indiquera si le document est valide.

3.6.2 L'API SAX

SAX [13, 14] est une API créée par David Megginson en 1998 et est l'acronyme de **S**imple **A**PI for **X**ML. Elle permet de manipuler des documents XML en utilisant des événements envoyées à chaque rencontre d'un élément.

3.6.3 Le DOM

Le DOM [15] , ou **D**ocument **O**bject **M**odel, est une interface de programmation normalisée par le W3C et est indépendant de tout plateforme et langage. Il voit les documents à balises comme des arbres dont le contenu et la structure peuvent être accédés et mis à jour dynamiquement.

La figure suivante montre un exemple de DOM.

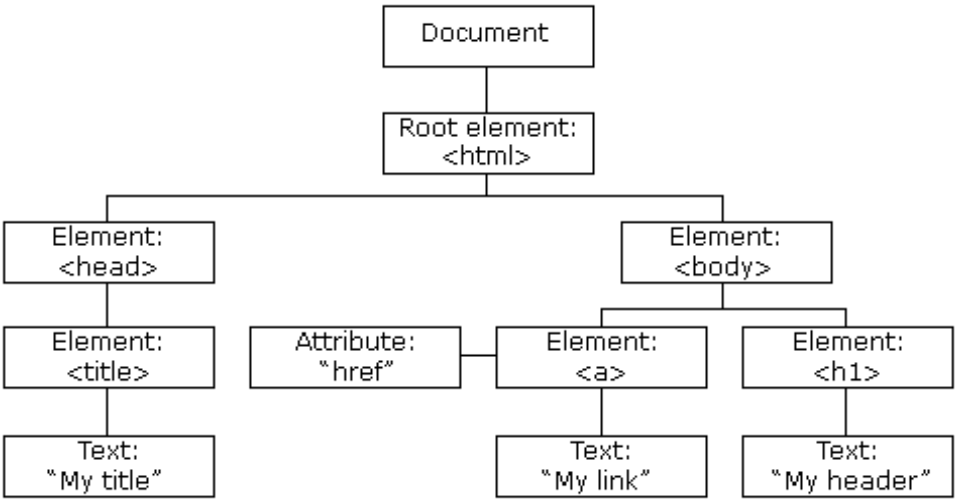


FIGURE 3 – Exemple d'un DOM d'une page HTML

3.6.4 Affichage d'un graphe avec GraphStream

Une fois le fichier XML parsé, nous disposons d'un *DOM* sous forme d'un objet *document*. Nous pouvons ensuite afficher l'arbre grâce à la librairie GraphStream [16]. Pour cela nous appelons un méthode qui affiche un nœud de l'arbre sur la racine du *Document*. Cela nous permet donc d'afficher tout l'arbre.

4 Architecture du programme

Dans cette section nous aborderons l'architecture du programme. C'est-à-dire la façon dont est organisé notre code et notamment les choix d'implémentation que nous avons effectués. Pour rappel, notre objectif est de générer un arbre rythmique et de récupérer la liste des accords et symboles musicaux pour réaliser l'affichage de la partition musical.

4.1 Architecture générale

Le programme se divise en trois parties. La première étant le parser. Cette partie permet au programme de disposer d'un *DOM* à partir d'un fichier MusicXML. La seconde partie est la représentation objet des données contenues dans le *DOM*. Et enfin la troisième partie est l'arbre rythmique qui pourra être généré à partir de la représentation objet de la partition.

4.2 Parsing du fichier MusicXML

La première étape est de parser le document afin d'extraire les informations qu'il contient. C'est là qu'interviennent les éléments contenus dans le package *pstl.musicxml.parsing*. Nous avons décidé d'encapsuler le parser XML natif de Java dans une classe *XMLParser*. Ce choix est motivé pour plusieurs raisons.

Tout d'abord pour ne laisser apparent que les fonctions du parser de Java que nous allons réellement utiliser afin de limiter le risque que nous ou un utilisateur tiers fasse usage du parser d'une mauvaise façon. Nous avons aussi fait ce choix pour simplifier l'utilisation de la classe pour que la création du parser qu'elle encapsule se fasse de la même façon à chaque fois. Et enfin ce choix a été rendu nécessaire à cause de *Relax NG*. En effet comme nous ne faisons appel ni aux fichiers *DTD* ni aux fichiers *XML Schema*, il nous faut effectuer un prétraitement pour éliminer les références aux *DTD* du fichier à parser. Vous pourriez vous demander pourquoi ne pas simplement utiliser les *DTD* pour valider le document ? La raison est simple, la plupart des fichiers font référence aux *DTD* en ligne fournis par MusicXML. Or ces derniers n'autorisent que les navigateurs web à accéder à de tels fichiers. Les possibilités qui s'offraient à nous étaient les suivantes : se faire passer pour un navigateur en modifiant quelques variables d'environnement. Cela aurait eu pour désavantage tout d'abord de ne pas être très rapide, l'accès à aux *DTD* par réseau n'est pas très rapide comparé à un accès local. Nous jugions d'autre part la méthode peu honnête. En effet si l'organisation derrière MusicXML ne permet pas cela pour des raisons que j'imagine financières (par cela j'entends le coût engendré par la maintenance des serveurs) il n'aurait pas été juste d'outrepasser leurs instructions. Et enfin le dernier choix non retenu était celui de faire appel à des *DTD* stockées localement. Cette méthode a été écartée bien que conseillée dans la documentation du format MusicXML car elle impliquait des redirections d'URI ce qui aurait fortement complexifié la création du parser.

C'est donc pour toutes ces raisons que nous avons choisi d'utiliser *Relax NG*. De plus, les fichiers décrivant la grammaire de MusicXML sont disponibles en ligne et l'auteur, qui les a déposés sur un projet GitHub [17], a fait preuve d'une certaine exhaustivité lors de la rédaction de ces derniers.

Ce parser nous permet de disposer d'un *DOM* (qui a pour nom de classe *Document* avec Java) qui pourra être parcouru plus tard. Le schéma suivant récapitule les étapes pour passer du document XML au DOM.

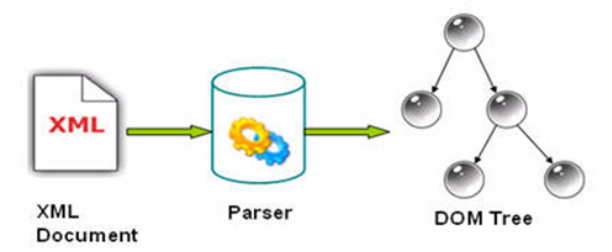


FIGURE 4 – Parsing d'un document XML en DOM

4.3 Représentation objet de la partition

Nous aurions pu nous satisfaire du *Document* retourné par le parser pour générer notre arbre rythmique mais cela aurait posé plusieurs problèmes. Tout d'abord la manipulation du *DOM* n'est pas aisée. Cette structure de données basé sur des nœuds qui contiennent des fils, attributs et leur contenu textuel, doit être exploré d'une façon assez lourde à cause en partie du fait qu'un nœud n'est pas nativement un objet itérable comme les *Collections* de Java par exemple. D'autre part, toutes les données contenues dans ces nœuds sont considérées comme des chaînes de caractère qui nécessitent un parsing et donc une manipulation assez verbeuse. De ce fait, il est plus simple pour nous de parcourir ce *Document* une seule fois et d'extraire les données qu'il contient dans une structure de données plus aisément manipulable dans Java. Cela permet par exemple d'éviter des erreurs lors du développement des autres fonctionnalités de l'application qui se basent sur ces informations. Nous utilisons donc un ensemble de méthodes contenus dans la classe *ScoreUtils* pour convertir notre *Document* en instance de *Score*.

ScoreUtils intègre aussi une partie très importante, la récupération des symboles contenus dans la partition. En effet, une partition n'est pas seulement un ensemble de notes il contient aussi un grand nombre de symbole ayant tous des significations très differentes pourvant aussi bien influancer la tonalité de la note ainsi que sa durée.

La structure de données que nous avons élaborée se compose de la façon suivante : l'élément qui va contenir toutes les informations est la classe *Score*. Une *Score* contient une liste de *Parts*

qui l'on peut qualifier de *Voix* en français. Chaque *Part* contient un nom, un identifiant ainsi qu'une liste de *Measures* (mesure) qui contient elle-même un numéro, une *Signature*, une liste de *IMusicalItem* et un *Metronome* correspondant au tempo.

IMusicalItem est la super classe de bon nombre d'élément que nous pouvons qualifier de bout de chaine comme les *Note*, *Rest* (silence) ou encore les *Tie* (liaison). Mais ce n'est pas tout, les groupes de *IMusicalItem* sont également des *IMusicalItem*. Cela nous permet entre autre de pouvoir imbriquer des groupes de *IMusicalItem*. Dans les premières version de cette partie du code, il y n'y avait pas de lien entre les mesures et les groupes. Nous nous sommes rapidement rendu compte que la mesure et le groupe étaient a peu de chose près la même chose. Nous avons donc décidé qu'il serait mieux que le mesure soit aussi un groupe pour limiter la duplication de code. Un groupe peut représenter des croches liées ou encore un *triolet*.

Le tempo contient un type (une noire par exemple) et un nombre par minute. Ainsi, le tempo *60 à la noire* signifie qu'il y a, dans une minute, 60 noires ou bien 30 blanches. Un élément musical possède aussi une liste de symboles musicaux. Un symbole musical représente une notation qui peut être une nuance comme un point ou encore un répétition par exemple.

Pour chaque symbole musical, une classe lui est associée. Le symbole peut être unaire ou binaire. Les symboles unaires sont liés à une seule note, tel que les points d'orgues. Les symboles binaires sont liés à 2 notes, comme les liaisons. Pour chaque note, on récupère tous les symboles qui lui sont associés, on créé les objets intermédiaires et on les ajoute à l'objet *Note*.

La création de l'objet *Score* se fait en deux étapes. Tout d'abord nous récuperons les données brutes dans le *Document* que nous mettons dans l'objet *Score* final. Une fois toutes ces données à disposition nous pouvons créer les groupes en fonction des symboles que nous rencontrons lors du parcours des mesures. Prenons comme exemple la balise *<beam>* qui peut être présente dans une balise *<note>* et qui contient les informations nécessaires pour représenter un groupe de croches dans *MusicXML*. Cette balise a principalement un attribut *number* qui designe le numéro du groupe actuel dans le cas où il a plusieurs groupes dans une mesure et qui contient l'énumeration suivante : *begin, end, continue* qui nous permet de connaitre la position de la note dans le groupe. Lors de notre premier passage nous stockons ces informations dans un objet *Beam*. Lors de notre second passage nous utilisons ces données pour créer nos groupes.

On pourrait penser que ce modèle de données est lourd, mais cela est largement compensé entre autres par l'aisance d'utilisation qu'il procure ainsi que la possibilité notamment de déduire les "coordonnées" des éléments qui le composent. En effet rien de plus facile que de dire qu'une note se trouve dans le *Chord* 4 de la *Measure* 2 de la *Part* 2 de telle *Score*. Ainsi nous pouvons par exemple utiliser ces coordonnées pour placer certains symboles à des endroits précis.

4.4 Construction des arbres rythmiques

Une fois la structure objet construite, les arbres rythmiques pour chaque mesure sont construits. Cette étape est réalisée par les classes du package *rhythmicstructures*. La classe factory *RhythmicTreeFactory* parcours tout l'arbre des objets intermédiaires, et pour chaque *Measure* rencontrée, un nouvel objet de la classe *RhythmicTree* est créé. Ce dernier contient une *Signature*, une *Fraction*, un *ItemType*, une liste d'*ExtraSymbols* et une liste de *RhythmicTree* correspondant aux fils.

4.5 Test du programme

Afin de vérifier que notre programme fonctionne correctement, une base de test a été créée dans le dossier *test-data*. Elle est constituée de nombreux fichiers tests au format MusicXML. Chaque fichier contient quelques notes avec un symbole musical (signe crescendo, staccato, etc.). Chaque classe associée à un symbole peut ainsi être testée. Une partition complète de Bach est également présente pour tester sur un grand nombre de mesure.

5 Conclusion

Ce projet nous a permis de découvrir le domaine de la musique qui était obscur pour nous. De plus, nous ne pensions pas que l'informatique pouvait être utile à cet art.

Nous avons implémenté le cœur du module et l'avons testé avec quelques symboles musicaux. Si nous avions eu plus de temps, nous aurions pu gérer la totalité des nombreux symboles pris en compte dans la grammaire de MusicXML.

Références

[1] IRCAM, "Accueil | ircam," 2017. [Online]. Available : https://www.ircam.fr

[2] C. Agon, G. Assayag, and J. Bresson, 2016. [Online]. Available : http://repmus.ircam.fr/openmusic/home

[3] S. Duchenne and A. Gaspard, "Github - screachfr/pstl_musicxml," 2017. [Online]. Available : https://github.com/ScreachFr/pstl_musicxml

[4] C. Agon, K. Haddad, and G. Assayag, "Representation and Rendering of Rhythmic Structures," in *WedelMusic 2002*, NA, France, 2002, pp. –, cote interne IRCAM : Agon02b. [Online]. Available : https://hal.archives-ouvertes.fr/hal-01106194

[5] Oracle, "The java tutorials," 2015. [Online]. Available : https://docs.oracle.com/javase/tutorial/

[6] W3C, "Extensible markup language (xml)," 2016. [Online]. Available : https://www.w3.org/XML/

[7] L. Roland, "Structurez vos données avec xml," 2017. [Online]. Available : https://openclassrooms.com/courses/structurez-vos-donnees-avec-xml

[8] MakeMusic, "Musicxml for exchanging digital sheet music," 2017. [Online]. Available : https://www.musicxml.com/

[9] C. Loup, "Apprendre le solfège - apprendre la musique," 2017. [Online]. Available : http://www.apprendrelesolfege.com/

[10] J. Clark and M. Makoto, "Relax ng home page," 2014. [Online]. Available : http://relaxng.org/

[11] T. O. Source, "Trang," 2008. [Online]. Available : http://www.thaiopensource.com/relaxng/trang.html

[12] ——, "Jing," 2008. [Online]. Available : http://www.thaiopensource.com/relaxng/jing.html

[13] D. Megginson, "Sax," 2004. [Online]. Available : http://www.saxproject.org/

[14] Oracle, "Lesson : Simple api for xml," 2015. [Online]. Available : https://docs.oracle.com/javase/tutorial/jaxp/sax/index.html

[15] W3C, "Document object model (dom)," 2005. [Online]. Available : https://www.w3.org/DOM/

[16] T. G. Team, "Graphstream - a dynamic graph library," 2015. [Online]. Available : http://graphstream-project.org/

[17] V. Biragnet, "Relaxng-for-music-xml," 2012. [Online]. Available : https://github.com/VincentBiragnet/RelaxNG-for-Music-XML