Sorbonne Université

Rapport du PSTL

Visualisation de la trace d'un langage synchrone pour la musique.

Réalisé par :
Merouane ALOUI
Mohamed BOUTOUGOUMAS

Tuteurs : Pierre Donat-Bouillud
Jean-Louis Giavitto

Année 2018-2019

Table des matières

1 Introduction **1**
1.1 Présentation d'Antescofo 2
1.1.1 Langage Antescofo 3
1.1.2 Machine d'écoute 4
1.1.3 Puredata 4
1.2 État de l'art 5
1.3 Planification du projet 7

2 Développement **7**
2.1 Choix d'implémentation 8
2.2 Parsing d'une partition 8
2.2.1 Spécification 9
2.2.2 Parsing 10
2.3 Communication entre *Wasco* et *Antescofo* 11
2.3.1 OSC 11
2.3.2 Serveur intermédiaire (*NodeJs*) 12
2.4 Réalisation 12
2.4.1 Interface 12
2.4.2 Structure des communications 15
2.5 Problèmes rencontrés 19

3 Conclusion **20**

4 Annexe **21**

Table des figures

1 Interactions entre Antescofo et un musicien [1] 2
2 Capture d'un patch puredata pour Antescofo 5
3 Interface d'AscoGraph 6
4 Planification du projet 7
5 Spécification des événements musicaux[2] 9
6 Spécification d'un pitch (hauteur)[2] 10
7 Suivi de la partition musicale Ravel. 14
8 Schéma représentant la structure des communications. 16

1 Introduction

La musique et l'informatique sont deux domaines disjoints, mais pourtant on aperçoit une forte implication de cette dernière dans les étapes de création, modification, et publication musicale.
Antescofo entre dans la catégorie des outils utilisés dans la branche des systèmes musicaux interactifs (IMS), ainsi il offre un moyen pour les musiciens de se synchroniser en temps réel avec des instruments électroniques.

Ce projet consiste à proposer un outil de visualisation de la trace d'une partition musicale écrite dans le langage Antescofo, en utilisant des technologies web récentes (HTML5, CSS3 et SVG) afin de rendre l'affichage compatible avec toutes les plateformes Mac, Linux, Windows grâce à la probabilité des navigateurs web, mais aussi car ces technologies ne risquent pas d'être déprécié dans le courant des années à venir (4 à 5 ans) ce qui permet à notre application de persister dans le temps et faciliter ainsi sa maintenabilité.

Une partition musicale écrite dans le langage Antescofo est composée d'évènements liés aux notes musicales telles que les Note, Trill, Chord et de Multi, ainsi que des actions liées à ces évènements.

La trace d'une partition augmentée[1], est stocké dans notre projet sous forme d'un tableau Json, celui-ci contient toutes les notes, chrod, trill et multi de la partition avec leur durée supposée.
Cette trace nous l'obtenons suite au parsing de la partition écrite en langage Antescofo, qui est ensuite utilisé pour représenter graphiquement les éléments (notes musicales) de la partition.

Antescofo disposée auparavant d'une application nommée Ascograph, qui permettait de visualiser la trace d'une partition, mais cet outil n'est plus maintenable, ce qui a poussé l'équipe d'Antescofo à vouloir en développer un nouveau. Nous nous somme donc inspiré d'Ascograph,et en particulier comment il communique avec Antescofo via des messages OSC.

1. Une partition musicale augmentée, est l'espace permettant de représenter, composer et manipuler des objets musicaux hétérogènes, aussi bien dans le domaine graphique que temporel.

1.1 Présentation d'Antescofo

> Antescofo est un suiveur de partition automatique et un langage synchrone et temporisé. Il permet de synchroniser un musicien avec des instruments électroniques ; le suivi, la synchronisation et les actions à effectuer sont précisées par le langage Antescofo. Le langage Antescofo permet donc de décrire les événements à suivre, et les actions en réaction aux événements, et peut indiquer les durées des actions et des événements en temps physique (en secondes) ou en temps musical (en pulsations).[1]

D'après cette définition l'outil Antescofo est composé de deux composants qui sont le langage Antescofo qui permet de synchroniser le musicien avec l'ordinateur, et un mécanisme qui permet d'extraire le tempo et la position dans la partition qu'on appellera par la suite machine d'écoute.
Pour que cet outil puisse fonctionner on aura besoin d'un outil de suivi de partition [2] comme Puredata ou Max qui intègre un objet Antescofo.

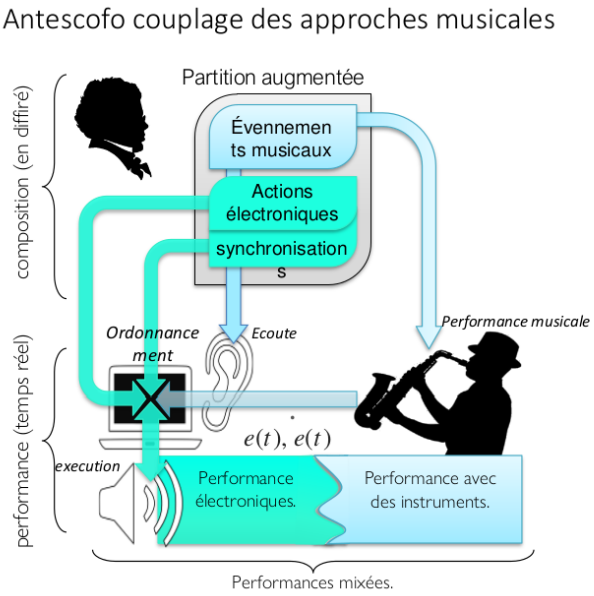

FIGURE 1 – Interactions entre Antescofo et un musicien [1]

2. Le suivi de partition est le processus consistant à écouter automatiquement une exécution musicale live et à suivre sa position dans la partition.

1.1.1 Langage Antescofo

> Le langage Antescofo permet à un compositeur de décrire une partition augmentée correspondant à un scénario musical complexe où les actions électroniques sont générées en fonction du jeu du musicien.

Le langage Antescofo permet à un compositeur de décrire une partition augmentée correspondant à un scénario musical complexe où les actions électroniques sont générées en fonction du jeu du musicien.
C'est un langage impératif avec des variables de type flots de données, comparable à d'autres langages synchrone tel que Lustre ou Esterel, qui réagissent à des événements reçus, et y répondent par des actions ordonnancées en utilisant une notion de temps logique spécifique et une politique d'ordonnancement déclarer dans le scheduler (cas d'Antescofo).
Nous donnons ci-dessus un exemple de partition écrite en langage Antescofo.

```
1  BPM     65
2  NOTE    C4 1.0 Measure1
3  CHORD   (C4 E4) 2.0
4  NOTE    64 1/2
5  CHORD   (D4 F4) 1.0
6  NOTE    0  1.0  ; a silence
7  NOTE    G4 0.0  ; a grace note with duration zero
8
9  @fun_def @midi2hz($x)  {$diapason * @exp(($x-69.0) * @log(2.0)/12)}
10
11      GROUP Solo2
12      {
13                    ASCOtoCS_SYNTH4 c (@midi2hz($tabFreq[0]))
14                    ASCOtoCS_SYNTH4 c (@midi2hz($tabFreq[2]))
15                    ASCOtoCS_SYNTH4 c (@midi2hz($tabFreq[4]))
16      }
17
18 TRILL   (A4 B4) 1.0
19 NOTE    0  1.0     ; a silence
20 TRILL   ( (C5 E5) (D5 F5) ) 2.0
21 MULTI   ( (F4 C5) -> (D4 A4) ) 4.0
22 1.0s
23 print "A message to print !"
24 TRILL   ( (A4  A2) (B4  A2) )        1/2
25 TRILL   ( (A4  D3) (B4  D3) )        1/2
26 TRILL   ( (A4  C3 E3) (B4  C3 E3) )  1/2
27 TRILL   ( A4 B4 )    2.0
```

1.1.2 Machine d'écoute

Une machine d'écoute artificielle qui capte des signaux audio de la part des musiciens en temps réel, et les différents paramètres nécessaires à l'interprétation de la partition par un ordinateur comme le tempo actuel et la position de la note courante dans la partition.
Cette machine s'appuie dans son fonctionnement sur un bagage machine learning et sur le traitement de signal tout en faisant coopérer le processus de détection de tempo avec celui qui calcule la position courante mais pas que car ces deux derniers peuvent se concurrencer.
Pour donner un exemple de traitement de la machine d'écoute, nous citons le processus de détection de tempo qui se base sur un modèle cognitif de la synchronisation musicale qui se base sur le phénomène de sympathie des pendules.[3]

1.1.3 Puredata

Est un outil de suivi de partition. Son fonctionnement général est de la sorte ; l'outil est relié aux instruments musicaux et reçoit des signaux audio qui lui sont transmis, ces signaux sont ensuite convertis depuis un format audio analogique vers un format numérique. Les signaux convertis sont ensuite utilisés dans l'interface de puredata et manipulés via des patchs (un patch, c'est une feuille de travail (ou un espace de travail), au commencement vide, sur laquelle nous allons pouvoir « composer », « tisser» ou « écrire » notre programme)[4].

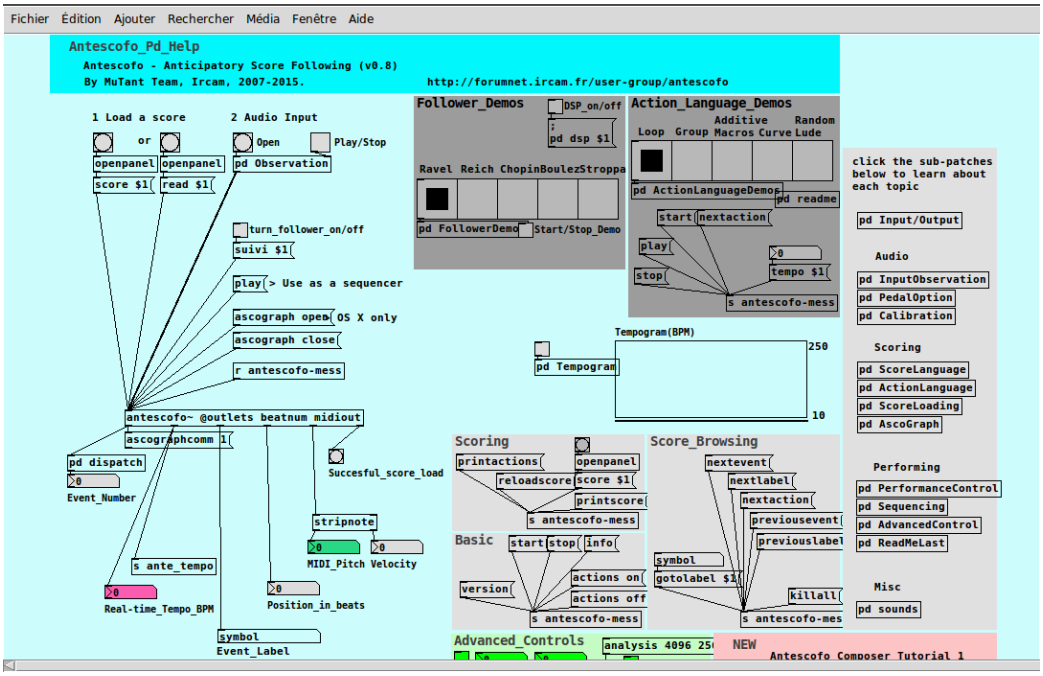

FIGURE 2 – Capture d'un patch puredata pour Antescofo

1.2 État de l'art

AscoGraph est le logiciel qui permet la conversion, l'écriture et la visualisation d'une partition musicale. Les partitions sont écrites dans des fichiers au format MusicXML [3] puis à l'aide d'un drag and drop de ces fichiers là sur l'interface d'AscoGraph celui-ci les convertit au format de notes d'Antescofo.[5]

3. MusicXML est un format de fichiers ouvert basé sur XML pour la notation musicale.

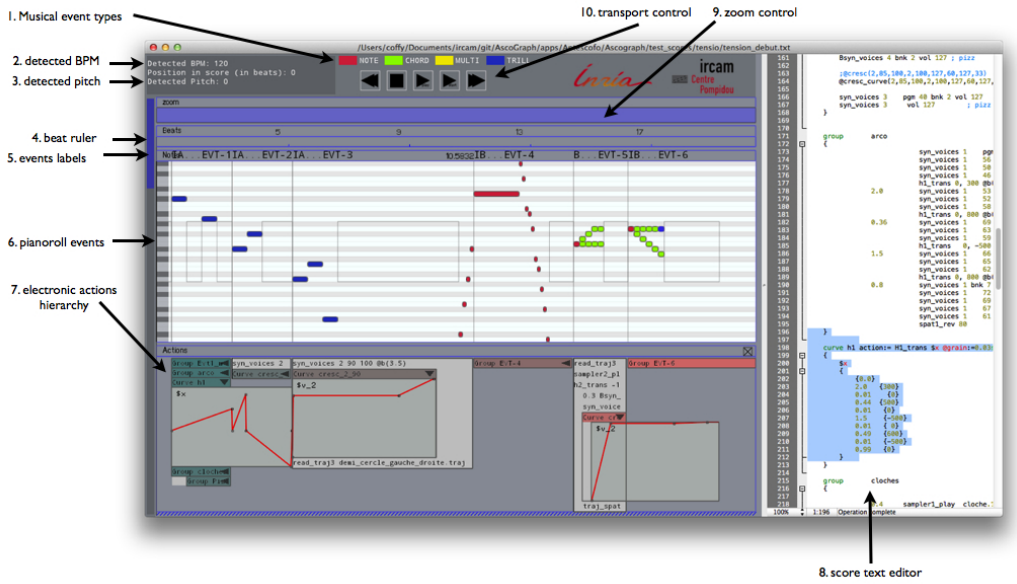

FIGURE 3 – Interface d'AscoGraph

Son interface propose différentes fonctionnalités, les plus importantes sont l'affichage du pianoroll représenté par la flèche 6, l'éditeur texte de la partition représenté par la flèche 8, et finalement les buttons de lancement et de navigation dans la partition musicale, représenté par la flèche 10. Il propose aussi des fonctionnalités secondaires telles que l'affichage du tempo courant ainsi que de la note courante.

Critique

Un des points négatifs de ce logiciel est le fait qu'il soit trop dépendant d'Antescofo. En effet pour produire Ascograph, il faut auparavant compiler une librairie Antescofo, qu'il faudra ensuite lier à Ascograph, et ce procédé doit être réitéré à chaque modification ou changement dans le programme d'Antescofo.
Cela montre clairement que l'outil n'est pas très flexible.

Un autre point négatif et cette fois un peu plus contraignant est la difficulté de maintenir l'outil, car la timeline (curseur) de la partition, s'appuyait sur un addon OpenFrameworks [4], nommé ofx Timeline, qui n'est plus maintenu depuis maintenant des années.
Ces deux défauts majeurs (manque de flexibilité et difficulté à maintenir), sont les raisons qui ont entraîné l'arrêt de ce projet, et ont poussé l'équipe d'Antescofo à se diriger vers une autre solution, celle de l'application WEB.

1.3 Planification du projet

Notre projet a été réalisé en suivant le plan suivant :

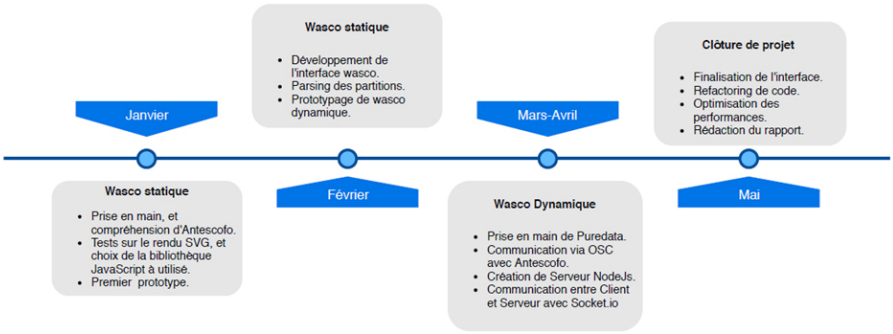

FIGURE 4 – Planification du projet

2 Développement

Dans ce chapitre du rapport nous allons parler des technologies et méthodes de développement utilisées pour mener à bout ce projet.

Notre application est basée sur une architecture trois tiers ce qui signifie que le client (qui est l'interface web *wasco*) demande une ressource au serveur (intermédiaire), cette ressource représente le beatpos [5] (position en beat) de

4. OpenFrameworks est une boîte à outils open source conçue pour le "codage créatif" fondée par Zachary Lieberman, Theo Watson et Arturo Castro en 2005, écrite en C++ et construit sur OpenGL.

5. Le beat (*battement*) est le temps de la mesure ou pulsation.

la note musicale courante, mais aussi de la position dans la partition en général, puis ce dernier part la chercher auprès d'un autre serveur (serveur d'Antescofo).

2.1 Choix d'implémentation

Nous avons utilisé les technos web afin de rendre le projet plus accessible et indépendant d'Antescofo mais aussi profiter des avantages que présentent certains langages web comme le javascript et ses frameworks.

Les partitions musicales sont affichées sur l'interface wasco en format SVG à l'aide d'une bibliothèque SVG.js. Une Note est dessinée avec un rectangle coloré avec la couleur associée à sa ligne dans le pianoroll, un Chord [6] (formée de plusieurs note) est dessiné en superposant toutes ses notes avec le même principe de dessin et faire la même chose pour les Trill [7] et les Multi [8]. Une fois la partition est dessinée, manque plus qu'une TimeLine (curseur) pour pouvoir défiler sur la partition dans le but de simuler le fait qu'un musicien est en train de la jouer, nous l'avons dessiné avec une ligne animée grâce à une fonction Animate() de la bibliothèque SVG.js [6].

L'avantage que le format SVG offre en plus de ce que peut offrir le Canvas par exemple c'est la qualité excellente de l'affichage qui vient du fait que SVG dessine ses objets au format vectoriel donc ne se déforme pas suite à un redimensionnement de l'interface, à autre avantage est le fait que les objets SVG sont directement intégrés au DOM [9], ce qui facilite la manipulation de ces objets (style, redimension, ajout d'effets d'animation, etc).

2.2 Parsing d'une partition

Pour dessiner notre trace nous avons besoin, auparavant de construire un tableau contenant toutes les notes musicales et leur durée dans la partition.

6. CHORD est un vecteur de notes.

7. TRILL est un conteneur composé soit d'élément simple comme des notes ou bien de chord.

8. MULTI définit un glissement "glissando" entre 2 évènements comme des trill ou des notes.

9. Document Object Model. Le DOM définit la structure d'une page et le moyen d'interagir avec elle : il s'agit d'une interface de programmation.

Ce tableau sera ensuite parcouru et chaque note sera dessinée.
Dans cette partie, nous allons donc présenter la spécification des évènements musicaux dans le langage Antescofo, ainsi que l'outil utilisé pour parser une partition écrite dans Antescofo, et en extraire les données qui nous intéressent.

2.2.1 Spécification

La figure 5 donne une vue globale sur comment les évènements musicaux sont définis dans le langage Antescofo.

Ainsi une *note* est représenté par le mot clés ***NOTE*** suivie par un ***pitch*** et d'un flottant représentant la ***durée***, suivie d'un ***attribut*** qui est soit un commentaire soit un label et a comme type une chaîne de caractères.
Un *chord* à l'inverse d'une *note* peut avoir plusieurs *pitch* qui partagent la même durée.
Un *trill* ou un *multi* peuvent quant à eux contenir une succession *chord*.

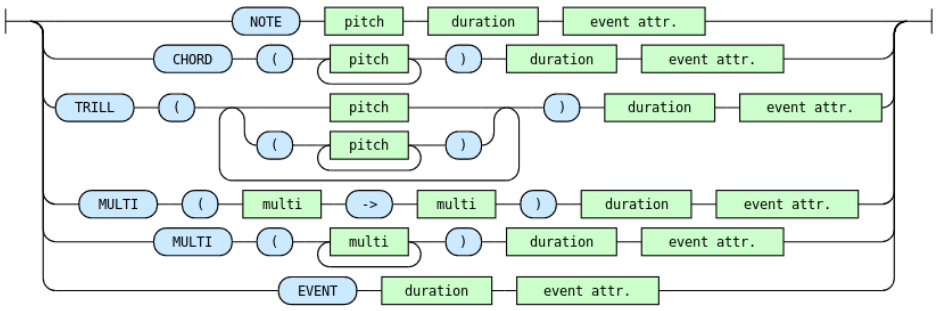

FIGURE 5 – Spécification des événements musicaux[2]

Un ***pitch*** est la hauteur [10] d'une note musicale. Il est généralement représenté par un **entier** ou un **flottant**, comme il peut aussi être représenté par son **nom** (ex : *A4*). (cf : figure 6)

10. Sa position dans le solfège.

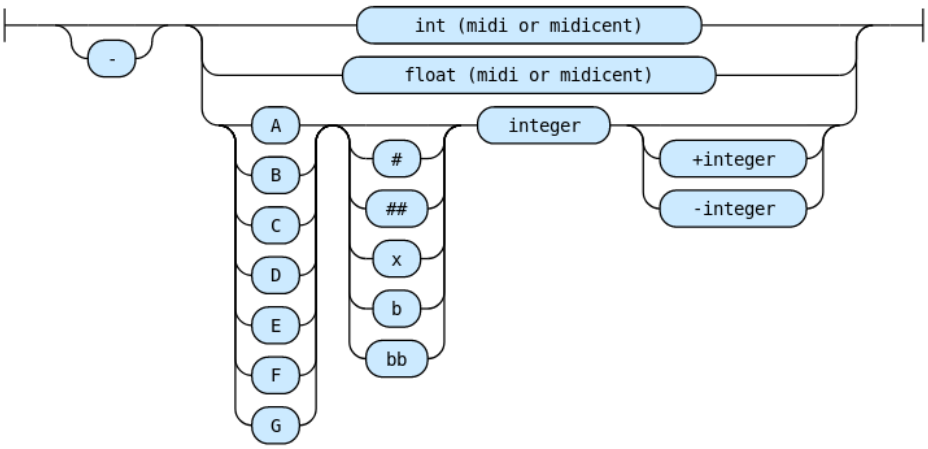

FIGURE 6 – Spécification d'un pitch (hauteur)[2]

2.2.2 Parsing

Pour récupérer les données qui nous intéressent (les notes et leur durée) dans une partition écrite dans Antescofo, nous avons utilisé un outil nommé *peg.js*[7] qui génère un parseur écrit en javaScript, cela est plus rapide que faire cette analyse nous-même en la codant avec des expressions régulières.

Cet outil est lui aussi écrit en javaScript, donc facile à être intégré dans notre application.

Peg.js génère un parseur basé sur le formalisme à la PEG [11], qui est un type d'analyseur syntaxique de la famille des parseurs du haut vers le bas. *Peg.js* offre un éditeur en ligne, sur lequel il suffit d'écrire notre grammaire, et ensuite télécharger le parseur écrit en javaScript. Notre grammaire suit les spécifications décrites dans la section 2.2.1, et donc pour reconnaître un chord, il faut que la ligne lue commence par le mot-clé *CHORD* suivie par un *pitch* et une *durée*, notre grammaire pour un chord est donc comme suit :

```
chord
 =  "CHORD" _ "(" c:pitch+ ")" _ d:duration _{return {chord : c, nb : c.length};}
```

11. Parsing Expression Grammar

2.3 Communication entre *Wasco* et *Antescofo*

La communication dans notre application est assurée grâce à Socket.io et OSC. Dans cette section nous donnons le détail concernant ces deux technologies.

2.3.1 OSC

> **O**pen **S**ound **C**ontrol (OSC) est un protocole de communication entre ordinateurs, synthétiseurs sonores et autres périphériques multimédias optimisés pour les technologies de réseau modernes. Ce protocole simple mais puissant fournit tout le nécessaire pour le contrôle en temps réel du traitement du son et des autres supports, tout en restant flexible et facile à mettre en œuvre.[8]

Notre application communique avec Antescofo au travers des messages OSC, en se connectant via une socket UDP. Ainsi Antescofo envoie les informations nécessaires (tempo, pitch, beatpos, ...) à notre application en passant par un serveur intermédiaire, qui se charge d'interpréter les messages OSC et de les formater en objets javaScript.

Pour recevoir ou envoyer des messages OSC, depuis notre application nous avons utilisé une librairie nommée osc.js.Cette librairie comme son nom l'indique, est écrite en javaScript, et peut facilement être reliée à notre serveur.

Exemple de message OSC :

```
"/carrier/freq" ",f" 440.4
```

Que *osc.js* traduit en objet Json de la sorte :

```
{
        address: "/carrier/freq",
        args: [
            {
                type: "f",
                value: "440.4"
            }
        ]
}
```

Dans le champ *address* on spécifie l'adresse à la quelle le message sera envoyé, dans notre application l'adresse à utilisé est ***/antescofo/cmd***, le champ *args* quant à lui représente les données à envoyé.

2.3.2 Serveur intermédiaire (*NodeJs*)

NodeJs est l'une des technos web les plus appréciées du moment, pour ses différentes fonctionnalités, et son côté bas niveau qui offre aux développeurs une meilleure gestion de leur application. L'un de ces aspects phares est le fait qu'il permet d'exécuter du javaScript côté serveur, ce qui permet aux développeurs de coder le backend et le frontend avec un seul langage, ainsi plus besoin d'avoir deux équipes chacune codant avec un langage.
Mais le point le plus important pour nous, est le fait que son fonctionnement soit basé sur les évènements, il permet donc de lancer des traitements à la réception d'un événement spécifique et de lancer d'autres traitements à la réception d'un autre événement.[9]

Un autre avantage pour nous est qu'il permet aussi de créer ses propres événements. C'est ce que nous avons utilisé pour gérer la synchronisation avec Antescofo.

2.4 Réalisation

Dans cette section nous donnons les détails de notre implémentation.

2.4.1 Interface

Traitement

L'interface Wasco permet d'afficher une partition musicale au format SVG, une partition est généralement composée de note, trill, chord, et multi mais pas que car dans les notes il y a ce qu'on appelle grace note (une note avec une durée de 0 pulsations) et les silences des notes sans midi (pitch) mais avec une durée, vu cette diversité nous avons dû faire des choix de représentation comme pour les graces notes nous les avons dessiné avec des cercles noirs, les silences eux, nous les dessinons avec un rectangle blanc qui prend toute la hauteur de l'interface pour dire que le musicien s'est arrêté de jouer pendant ce moment.

Pour réaliser l'interface nous somme passé auparavant par un traitement statique et puis nous avons introduit le traitement dynamique en prenant en compte cette fois les messages OSC.

Wasco statique

Le suivi peut se faire en statique avec un tableau de beatpos et à chaque durée aléatoire on va chercher un beatpos de ce tableau puis faire avancer la TimeLine à la note suivante dans la partition en utilisant cette valeur de beatpos, de cette manière nous avons simulé l'arrivée d'un événement qui a comme valeur le beatpos de la note suivante. Cette approche nous a permis de coder l'application indépendamment des événements reçus de la part d'Antescofo.

Wasco dynamique

Pour rendre l'affichage vif et dynamiquement avec des réactions suite à des évènements Socket.io [10] nous avons représenté une TimeLine qui défile sur la note courante tant que nous n'avons pas encore reçu un nouvel évènement désignant la nouvelle note à défiler et tant que la durée de défilement n'a pas expiré (la durée de défilement d'une note se calcule en multipliant sa durée supposée fois 1000 ms).

Problème lié aux graces notes

Petit souci avec les graces notes car leurs beatpos (leurs temps d'arrivée) arrive au même temps que celui de la vraie note qui arrive juste après donc on ne sait pas s'il s'agit d'une note normale ou d'une grace note, comme solution à ce problème, nous construisant au départ (quand on dessine les notes) une map qui associe à chaque grace note un beatpos (durée d'arrivée estimée ; une sorte de timestamp) comme ça quand son évènement beatpos arrive on va la chercher dans la map et il suffit de l'esquiver et passer à la note suivante.

Nous faisons la même chose pour les notes, trill, corde, et multi que ce qu'on fait pour les graces notes c'est-à-dire que nous créons une map quand nous dessinons la partition qui associe la note à son beatpos, et une fois que nous recevons l'événement beatpos c'est facile de trouver quelle note faire

défiler, mais ce n'est pas toujours le cas, car les durées qui servent à dessiner les notes sont estimées et ça ne doit pas forcément être les mêmes que les réelles, dans ce cas nous lançons une recherche dichotomique sur l'ensemble de la partition pour trouver la bonne note.

Fonctionnalités

Cette application via son interface utilisateur offre de multiples fonctions et options, dans cette section nous allons les décortiquer une à une.
Comme fonction de base l'application propose de charger une partition écrite dans le langage Antescofo pour qu'elle soit parsée et transformée en un tableau json utilisé par la fonction qui s'en charge de l'affichage, la partie parsing sera détaillée en dessous.
Bouton *pause* (respectivement *resume*) qui fait référence au fait d'arrêter de jouer une partition (respectivement relancer la partition).

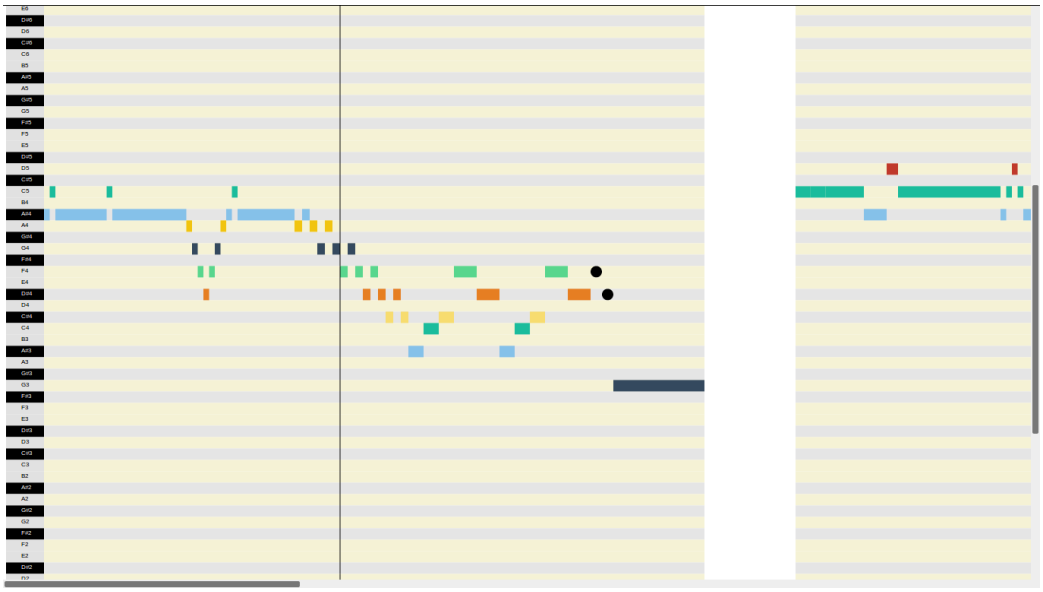

FIGURE 7 – Suivi de la partition musicale Ravel.

Parsing

Pour parser la partition (code) écrite en langage Antescofo, nous lisons le code ligne par ligne, et nous ignorons toutes celles qui ne commencent pas par

une note musicale, cela peut se faire grâce à l'expression régulière suivante :

```
var regex = /^(BPM|NOTE|CHORD|TRILL|MULTI)+/y;
```

Ensuite, si une ligne respecte l'expression régulière ci-dessus, on fait appel au parseur en lui passant le contenu de la ligne, cela ce fait grâce à l'instruction suivante :

```
parser.parse(lines[i]);
```

Ci-dessus nous donnons un exemple du résultat obtenu :
Soit la ligne lue dans la partition (code) est :

```
CHORD (A4 B1) 3/2
```

Le résultat serait :

```
[
   {
      "chord": [
         69,
         35
      ],
      "nb": 2
   },
   {
      "duration": 1.5
   }
]
```

2.4.2 Structure des communications

Dans cette partie nous expliquant et donnons les détails de la structure des communications, entre *wasco* et *Antescofo*.

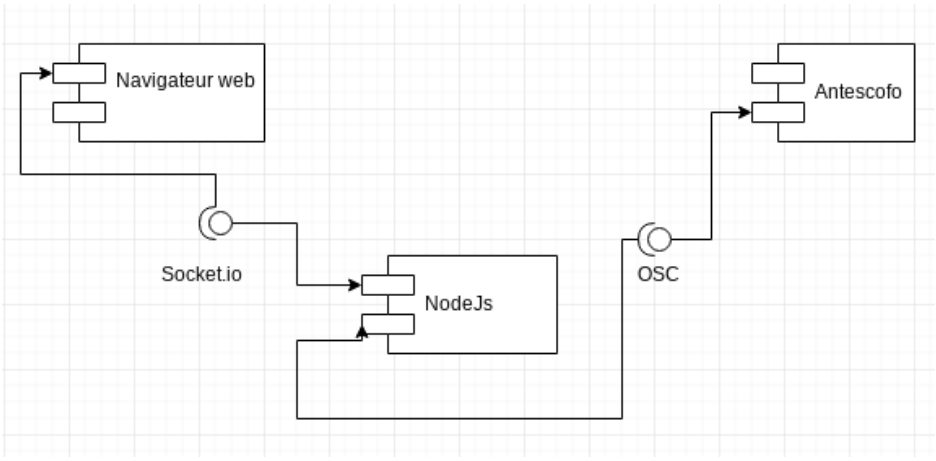

FIGURE 8 – Schéma représentant la structure des communications.

Antescofo utilise deux ports, l'un est le port : 5678 sur lequel il reçoit des messages OSC, et l'autre le port : 6789 sur lequel il envoie des messages.

Notre serveur quant à lui, initialise une socket udp en utilisant la méthode osc.UDPPORT fourni par osc.js, et la paramètre en donnant une adresse et un numéro de port, en occurrence le port 6789, pour écouter les messages qu'il perçoit depuis Antescofo, ensuite pour envoyer des messages à ce dernier il utilise la méthode send, paramétré avec le port 5678.
D'autre part notre interface web wasco communique quant à elle avec le serveur Node, via des WebSocket en utilisant la librairie Socket.io [ref site officiel] écrite en javaScript et relié à Node, et qui permet un échange bidirectionnel (depuis le serveur vers le client web et inversement) en temps réel (synchrone), ce qui n'est pas le fonctionnement habituel des serveurs web qui fonctionne généralement de manière asynchrone. Sachant que notre application doit être réactive et fonctionner de manière synchrone, l'utilisation du serveur Node combiné à Socket.io semble être une bonne solution.

Pour demander à Antescofo d'effectuer une tâche (charger une partition, lancer le suivi d'une partition, sauter à la position suivante ou commencer à une position précise, etc), il faut lui envoyer un message *OSC* correspondant à une des commandes internes.

Antescofo possède de nombreuses commandes internes, et celle qui nous ont été utiles sont les suivantes :

- *antescofo*::*score(string)* : charger une partition dont le chemin et passé en paramètre.
- *antescofo*::*play()* : permet de simuler une partition.
- *antescofo*::*startfrombeat(float)* : permet de lancer la simulation depuis la position en beat passé en paramètre.
- *antescofo*::*nextevent()* : saute au prochain événement dans la partition (les notes représentent des événements).
- *antescofo*::*previousevent()* : saute au précédent événement.
- *antescofo*::*stop()* : arrête la simulation et les tâches en-cours.

Au lancement de notre application, l'utilisateur choisit depuis l'interface *wasco* la partition écrite en langage Antescofo à jouer, ainsi le chemin absolu de celle-ci est envoyé en paramètre de l'évènement *FileEvent* que nous créons en utilisant l'instruction suivante :

```
socket.emit('FileEvent', 'chemin/partition.asco');
```

Puis le serveur Node reçoit l'évènement *FileEvent* grâce à l'instruction suivante :

```
socket.on('FileEvent', function (chemin) {/* suite des traitements */});
```

Ensuite ce dernier, envoie un message OSC à Antescofo, contenant le chemin de la partition, grâce à l'instruction suivante :

```
udpPort.send(osc_message, 'localhost', 5678);// 5678 le port d'ecoute d'Antescofo
```

Le message envoyé au format *json*, est comme suit :

```
{ //osc_message
        address: "/antescofo/cmd",
        args: [
            {
                type: "s",
                value: "score" // La commande interne - antescofo::score(string)
            },
            {
                type: "s",
                value: chemin // recus depuis l'interface wasco
            }
        ]
}
```

Ce message sera ensuite réécrit par *osc.js* au format OSC, et sera donc comme suit :

```
// osc_message au format OSC
"/antescofo/cmd" ",ss" "score" "chemin"
```

Une fois qu'Antescofo reçoit ce message, il exécute la commande interne *antescofo*::*score(chemin)*, et charge ainsi la partition en question, cela a aussi pour effet de lancer la simulation.

Une fois que la simulation est lancée, Antescofo envoie sur son port de sortie différents messages OSC, contenant chacun une information particulière.

Parmi les informations que partage par Antescofo on a :

- Le message *stop*, qui nous informe que la simulation est arrêtée.
- Le message *tempo float*, qui nous informe d'un changement de tempo.
- Le message *pitch float*, qui nous informe de la hauteur de la note courante.
- Le message *event_ beatpos float*, qui nous informe de la position courante dans la trace de la partition.

Celles qui nous intéressent ici sont *tempo* et *event_ beatpos*.

Notre serveur Node qui écoute sur le port de sortie d'Antescofo, reçoit ces messages OSC, grâce à l'instruction suivante :

```
udpPort.on("message", function (message) { /* suite des traitements */ });
```

Ce dernier, envoie ce message vers *wasco* via *socket.io* grâce à l'instruction suivante :

```
socket.emit('OSC-beatpos', osc_message);
```

Les messages OSC reçus sont comme suit :

```
"/antescofo/tempo" ",f" float
"/antescofo/pitch" ",f" float
"/antescofo/event_beatpos" ",f" float
```

Le serveur Node filtre ces messages, et envoie celui qui correspond à des évènements *beatpos*.

wasco reçoit ce message grâce à l'instruction suivante :

```
socket.on('OSC-beatpos', function (message) { /* suite des traitements */ });
```

Suite à ce message, un curseur (timeLine) lié au *beatpos* courant est créé, et son animation est lancé, grâce à l'instruction :

```
timeLine.animate( /* parametres */ );
```

Ainsi à chaque nouveaux message concernant un *beatpos* est reçu, un curseur associé est créé, et le suivi de la partition s'effectue.

Le message concernant le *tempo* est utilisé pour faire accélérer ou ralentir l'animation du curseur. Un changement de tempo correspond au fait que le musicien joue plus vite ou plus lentement.

2.5 Problèmes rencontrés

Au cours de ce projet nous avons rencontré beaucoup de difficultés notamment celles relatives à la compréhension des notations musicales comme les notes, les trill, les chord et multi, et ce qu'elles représentent réellement pour pouvoir les dessiner. Puis il fallait travailler avec un outil qui n'est pas open source ce qui nous a poussé à effectuer beaucoup de recherches documentaires qui aboutissent rarement à des ressources fiables et à jour.

En fin la plus grosse difficulté était les performances car notre premier prototype était très coûteux environ 30 % CPU donc nous avons dû repenser la manière dont on dessine la TimeLine pour finir avec une consommation inférieure à 8 % CPU.

3 Conclusion

Pour conclure ce projet, nous avons développé une application de visualisation de la trace d'une partition musicale écrite dans Antescofo qui s'appelle Wasco pour (Web Ascograph), ce projet a été proposé par l'équipe Antescofo dans le cadre des projets PSTL.

Au cours de ce projet nous étions en mesure de mettre en pratique les connaissances que nous avons acquis pendant les années de formation à la l'université notamment les connaissances relatives à la programmation web et réseaux mais pas que car nous avons exploré le langage Antescofo qui est un langage synchrone donc une abstraction de temps.

Ce projet est très important car il nous permet de relier l'informatique à la musique et ainsi interagir avec des clients qui n'ont probablement pas des connaissances dans l'informatique, donc nous étions dans l'obligation de faire des choses claires et compréhensibles par les non-informaticiens, ce qui nous mène à la problématique de l'ergonomie de l'interface homme machine avec la règle des trois cliques par exemple (un clic pour charger la partition un autre pour faire pause et un dernier pour resume).

Comparé à ce que Ascograph peut faire, Wasco ne peut pas faire grand-chose, donc nous avons décidé de noter les perspectives de notre application dans cette section afin d'expliquer aux lecteurs de ce rapport l'étendue du projet. Nous citons l'amélioration de l'affichage en ajoutant une représentation musicale classique avec une clef de Sol et des notes dessinées à la main avec la bibliothèque javascript *vexFlow*[11], ainsi que l'intégration d'une autre interface pour afficher les actions qu'Antescofo envoie au fil de son exécution.

4 Annexe

— Lien vers le dépôt git du projet
https ://github.com/programLyrique/wasco

Références

[1] "Introduction : the Antescofo System - AntescofoDoc." [Online]. Available : http ://antescofo-doc.ircam.fr/UserGuide/intro/

[2] "Event Specification - AntescofoDoc." [Online]. Available : http ://antescofo-doc.ircam.fr/Reference/event$_ref$/

[3] A. Cont, "ANTESCOFO : Anticipatory Synchronization and Control of Interactive Parameters in Computer Music." p. 9.

[4] "Site officiel Puredata." [Online]. Available : https ://fr.flossmanuals.net/puredata/introduction/

[5] T. Coffy, J.-L. Giavitto, and A. Cont, "AscoGraph : A User Interface for Sequencing and Score Following for Interactive Music," p. 6.

[6] "SVG." [Online]. Available : https ://developer.mozilla.org/fr/docs/Web/SVG

[7] "PEG.js – Parser Generator for JavaScript." [Online]. Available : https ://pegjs.org/

[8] "Introduction to OSC | opensoundcontrol.org." [Online]. Available : http ://opensoundcontrol.org/introduction-osc

[9] "Index | Node.js v9.11.2 Documentation." [Online]. Available : https ://nodejs.org/docs/latest-v9.x/api/

[10] "Socket.IO." [Online]. Available : https ://socket.io/index.html

[11] "VexFlow - HTML5 Music Engraving." [Online]. Available : http ://www.vexflow.com/