Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

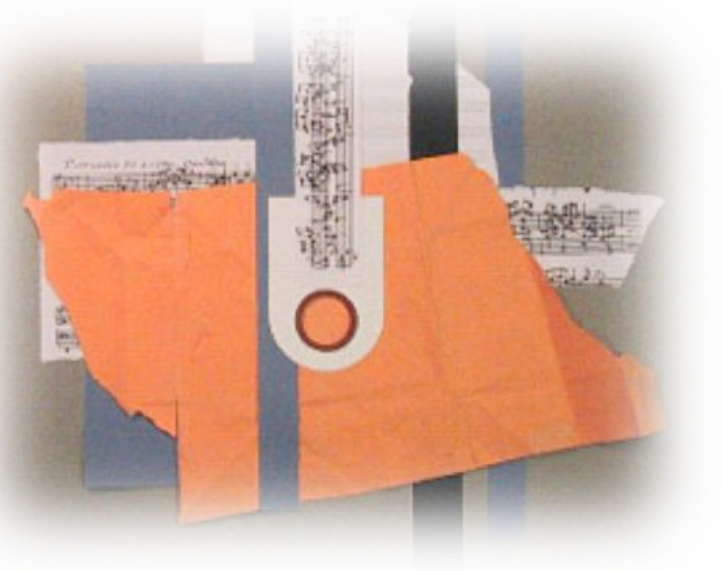
Bouche Dimitri
3A SyM 2012/2013

Ircam
1, place Igor-Stravinsky
75004 Paris
Département R&D
Equipe Représentations Musicales

Rapport de Projet de Fin d'Etudes

Dynamisation de l'architecture audio du logiciel de composition assistée par ordinateur

OpenMusic



Février – Août 2013

Responsable de stage : M. **Bresson** Jean

Remerciements

Je tiens tout d'abord à remercier tout le personnel de l'Ircam, avec qui j'ai eu le plaisir de travailler et d'échanger au cours de ces six mois pour leur accueil, le temps qu'elles m'ont consacré ainsi que leurs précieux conseils.

A Hugues Vinet et Gérard Assayag, respectivement Directeur du département Recherche et Développement et Directeur de l'UMR STMS (Science et Technologies de la Musique et du Son), qui m'ont permis d'intégrer l'Ircam.

A toute l'équipe Représentations Musicales avec qui j'ai pu collaborer et cohabiter, pour leur gentillesse et leur professionnalisme.

A mon maître de stage, Jean Bresson, de m'avoir proposé ce poste, qui a toujours su se rendre disponible et m'a énormément appris durant mon stage. Je le remercie également pour sa patience et pour les différentes rencontres qu'il a pu organiser pour me permettre de m'intégrer au mieux dans le projet de recherche.

Je remercie également les équipes de recherches du LaBRI et du GRAME, avec qui j'ai pu découvrir l'émulation générée par un projet de recherche, et plus particulièrement Stéphane Letz avec qui j'ai pu être en étroite collaboration durant mon stage, et qui a toujours répondu à mes demandes.

Table des Matières

Table des illustrations

Introduction

Le présent rapport résume les travaux réalisés au cours de mon projet de fin d'études, études réalisées au sein de l'ENSEA (**E**cole **N**ationale **S**upérieure de l'**E**lectronique et de ses **A**pplications) de Cergy-Pontoise. Etudiant issu de l'option SyM (**Sy**stèmes **M**ultimédias), j'ai souhaité de par ma formation et mes centres d'intérêts, effectuer un stage liant programmation informatique, temps réel, traitement audio et musical. C'est pourquoi j'ai intégré l'équipe RepMus (**Rep**résentations **Mus**icales), du département Recherche et Développement de l'Ircam (**I**nstitut de **R**echerche et **C**oordination **A**coustique/**M**usique), pour le stage intitulé : « Dynamisation de l'architecture audio du logiciel OpenMusic ».

1.1.Présentation de l'Ircam

L'Institut de recherche et coordination acoustique/musique est l'un des plus grands centres de recherche publique au monde se consacrant à la création musicale et à la recherche scientifique. Lieu unique où convergent la prospective artistique et l'innovation scientifique et technologique, l'institut (dirigé depuis 2006 par Frank Madlener) réunit plus de cent soixante collaborateurs.

L'Ircam oriente ses travaux sur trois axes principaux : création, recherche et transmission. Le centre propose des cursus et des résidences pour les compositeurs, ce qui permet aux chercheurs de confronter leurs travaux aux artistes auxquels ils se destinent.

Fondé par Pierre Boulez, l'Ircam est associé au Centre Pompidou sous la tutelle du ministère de la Culture et de la Communication. Depuis 1995, le ministère de la Culture et de la Communication, l'Ircam et le CNRS sont associés dans le cadre de l'unité mixte de recherche STMS (Sciences et technologies de la musique et du son - UMR 9912) rejoints, en 2010, par l'Université Pierre et Marie Curie (UPMC).

La figure 1 représente l'organigramme du département R&D de l'Ircam :

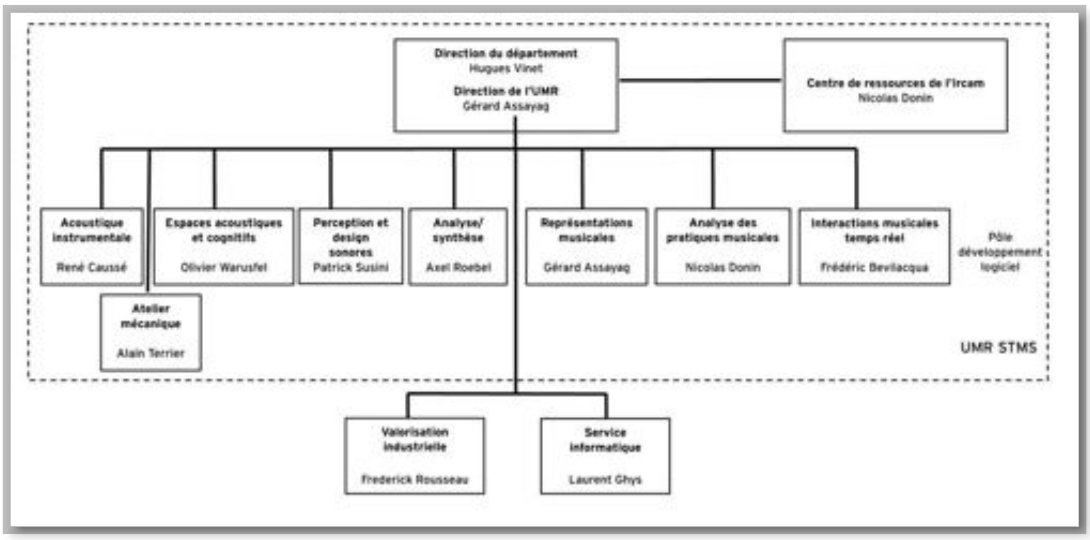


Figure 1 : Organigramme du département Recherche et Développement de l'Ircam.

1.2.Présentation de l'équipe RepMus

L'équipe Représentations Musicales conduit des recherches et développements dans le domaine de la représentation symbolique, des langages et paradigmes informatiques appliqués à la musique. Ces travaux trouvent application à la fois dans la Composition Assistée par Ordinateur (CAO) et dans la musicologie. La représentation haut niveau des concepts et structures musicales soutenues par des langages informatiques originaux développés par l'équipe, conduit à la conception de modèles qui peuvent être utilisés à la fois pour la production et l'analyse musicale.

Du côté de la création et depuis sa diffusion à une grande communauté de musiciens, le logiciel en CAO a provoqué de nouvelles manières de penser, exploitant les possibilités informatiques pour simultanément représenter et exécuter la partition, ses structures formelles sous-jacentes et leurs processus génératifs. Il est important de noter qu'un travail dirigé de cette manière contient, dans une large mesure, sa propre analyse structurelle (d'un point de vue musicologique, les outils de représentation et de modélisation permettent une approche réellement expérimentale, en injectant un dynamisme considérable à une discipline encore sous-représentée en France).

1.3.Contexte

Ce stage ingénieur orienté recherche s'inscrit dans le cadre du projet ANR INEDIT (**IN**teractivité dans l'**E**criture **D**e l'**I**nteraction et du **T**emps). L'objectif de ce projet est de fonder scientifiquement l'interopérabilité des outils de création sonore et musicale, afin d'ouvrir la voie à de nouvelles dimensions créatives couplant écriture du temps et écriture de l'interaction. Il lie trois équipes de recherche, issues de trois centres : le GRAME CNCM (**C**entre **N**ational de **C**réation **M**usicale) de Lyon, le LABRI (**LA**boratoire **B**ordelais de **R**echerche en **I**nformatique) de Bordeaux, et l'IRCAM.

La motivation principale de ce projet est de réconcilier la division actuelle entre les aspects « compositionnels » et « performatifs » des outils de création sonore. Cette division est apparente dans la catégorisation des outils en « Composition Assistée par Ordinateur » et « Temps réel ». INEDIT a pour objectif de permettre l'utilisation de différentes approches stylistiques grâce à la coopération d'outils au sein d'un environnement supportant toutes les phases du *workflow* musical, de la composition à la performance. Des logiciels existants créés par les acteurs de ce projet constituent une base pour l'aboutissement de ces objectifs : *OpenMusic* [1] et *iscore* [2] pour la phase de composition, *INScore* [3] pour la visualisation interactive, *Antescofo* [4] pour l'articulation signal/événement et *FAUST* [5] et *LibAudioStream* [6] pour la gestion des flux audio synchrones. Rendre interopérables ces outils permet de mettre en place des *workflows* riches et originaux qui seraient impossibles avec une application intégrée et un formalisme unique forcément plus rigide.
Le défi principal posé par ces problématiques est la formalisation des relations temporelles musicales et en particulier le développement d'une approche GALS (Globally Asynchronous, Locally Synchronous) permettant de concilier contraintes temporelles synchrones et asynchrones dans un contexte où il faut gérer plusieurs échelles de temps hétérogènes.

Mon rôle dans ce projet est de dynamiser l'architecture audio du logiciel OpenMusic de manière à ouvrir de nouveaux horizons de composition et d'interaction avec le temps. Les consignes sont larges, et le travail et la réflexion qu'elles impliquent sont très riches. En effet, je fus libre quant à la manière de réaliser ces objectifs, ce qui conduit à une mise en question permanente et à des décisions souvent sujettes à révision.

<u>Remarque :</u> Etant donné la taille du code développé au cours du stage, seules les fonctions principales figurent dans ce rapport, et les logiques d'exécution ne sont que partiellement détaillées. Pour plus de détails, se référer au code source.

2. Etat de l'art

2.1.Introduction au logiciel OpenMusic

2.1.1. Résumé

OpenMusic (OM) est un langage de programmation visuel basé sur le langage CommonLisp/CLOS (CommonLisp Object System) [7]. Dans ce logiciel de composition assistée par ordinateur Open Source [8], les programmes visuels sont créés en assemblant et connectant boîtes et icônes représentant fonctions et structures de données. Les structures de contrôle basiques contenues dans le langage Lisp sont fournies par OpenMusic sous forme de blocs visuels.
OM peut être utilisé comme un langage de programmation visuel fonctionnel ou orienté objet (d'autres paradigmes de programmation ont été implémentés dans l'environnement, comme la programmation par contraintes). D'un point de vue plus spécifique, une palette de classes et de bibliothèques en font un environnement adapté à la composition musicale. Autour du noyau d'OpenMusic se trouvent des projets spécialisés implémentant des données et comportements musicaux. Les données sont associées à des éditeurs graphiques et peuvent être modifiées par l'utilisateur pour satisfaire des besoins spécifiques. Différentes représentations des structures musicales sont supportées, parmi lesquelles on retrouve des notations courantes comme le piano-roll MIDI, la partition ou encore le signal sonore. Une organisation temporelle du matériel musical est proposée grâce à des outils tels que la Maquette ou l'OMSheet [9].
Il est possible de programmer en CommonLisp/CLOS dans OpenMusic, aussi bien textuellement que visuellement (par blocs).

Ce logiciel est destiné à la composition d'œuvres plutôt qu'à la performance. En effet, il n'est pas temps réel. En revanche, il permet des applications spécifiques uniques et souvent non adaptables en temps réel. La composition d'une œuvre dans OpenMusic peut se distinguer en deux phases interopérables:

- La préparation du matériel musical dans un « Patch »,
- Son organisation temporelle dans une « Maquette ».

2.1.2. Le « Patch »

Le Patch [10] est un espace de création libre qui représente un programme visuel sous forme de graphe, où l'on dispose et connecte des boîtes représentants des processus musicaux. Ici, il n'y a pas de notion d'échelle temporelle. On peut concevoir et écouter indépendamment divers matériaux, et les faire interagir entre eux. On peut également, en plus des multiples processus intégrés nativement dans le logiciel, en créer de nouveaux soit en agençant différentes boîtes entre elles, soit en créant ses propres boîtes et en définissant leur rôle en code Lisp. Ci dessous, un exemple de patch montre que l'on peut manipuler différentes représentations musicales (figure 2) :

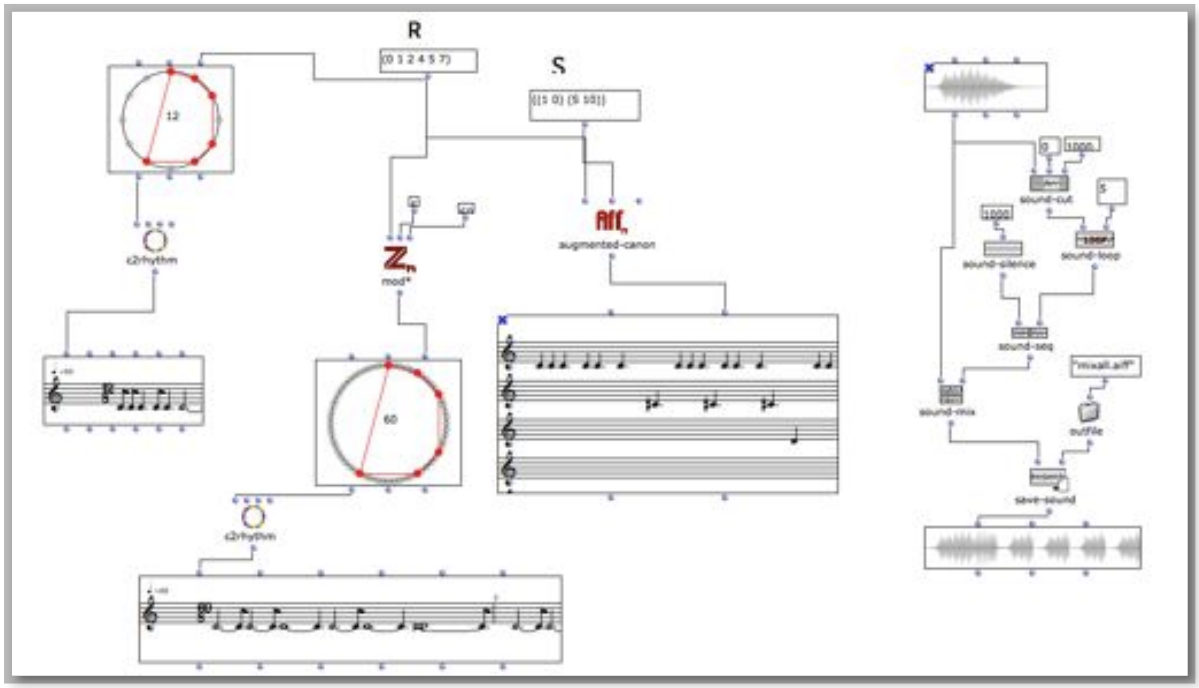


Figure 2 : Exemple de Patch.

Ensuite, il est possible d'organiser temporellement ces patchs, ou les boites qui les composent.

2.1.3. La « Maquette »

La Maquette est un donc un espace d'organisation temporelle des objets, ou même des Patches. Il est également possible de connecter ces objets et Patches entre eux, de manière à transmettre l'information pour du calcul relatif des données. Cet espace intègre donc structures fonctionnelles et temporelles au sein d'un même programme visuel. La figure 3 représente un exemple de maquette, mêlant différentes représentations musicales et incluant une transmission de données entre deux Patch (les deux boîtes connectées représentent en réalité des patches implémentant des fonctions définies par l'utilisateur et produisant des données musicales) :

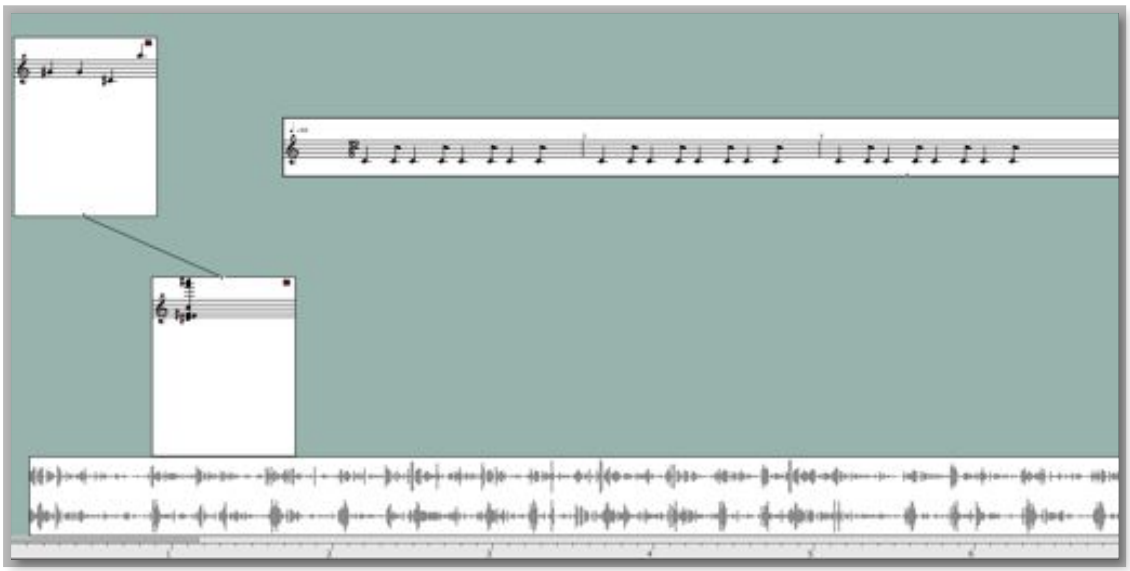


Figure 3 : Exemple de Maquette.

2.2.Architecture Audio/Midi actuelle

Actuellement, le logiciel OpenMusic s'articule autour de plusieurs moteurs de rendu audio/MIDI :

- Midishare (bibliothèque) pour le rendu du protocole MIDI [11],
- *LibAudioStream* (bibliothèque) pour le rendu et la gestion de l'audio,
- *Multiplayer* (application externe) pour le rendu de l'audio multi canal ou spatialisé [12],
- *Microplayer* (application externe) pour le rendu du Midi micro-intervallique [13].

Pour communiquer avec des applications externes (ex : Multiplayer, Microplayer etc...) on utilise le protocole OSC [14].

Ces différents moteurs sont gérés par une classe nommée *General Player*. Ce player unique fait office d'ordonnanceur. OpenMusic ne contient qu'une unique instance de ce player, elle même gérée par une unique horloge. Ainsi, il est impossible d'avoir une lecture dynamique des fichiers : en effet, si un fichier est déjà en lecture, le statut du *General Player* sera occupé et donc il est nécessaire de repasser dans un état libre (en arrêtant la lecture) afin d'en relancer une autre. L'ordonnancement est géré indépendamment par les différents moteurs, via la construction préalable de séquences statiques, ce qui entraine une perte de contrôle totale sur les objets en lecture. C'est pour cette raison que le logiciel dispose d'une seule barre de transport (nommée *Palette*) empêchant des appuis multiples sur les boutons de lecture et d'arrêt.
La figure 4 résume ce système :

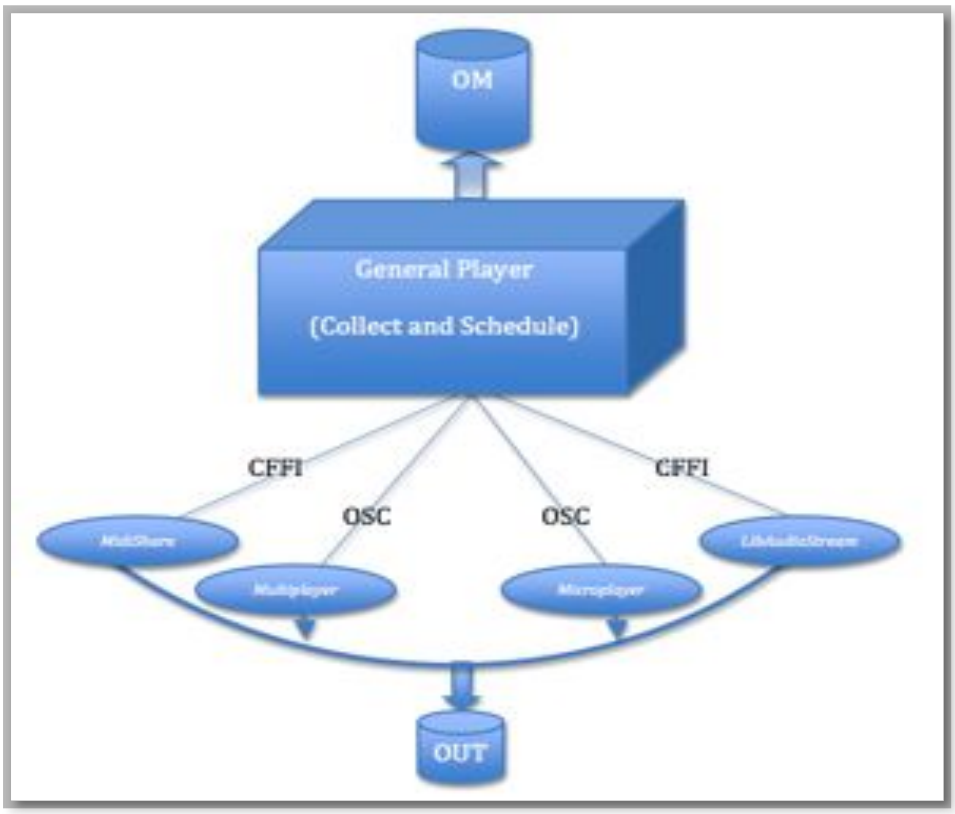


Figure 4 : Architecture audio actuelle schématisée.

Pour ce qui est de la synthèse audio, les moyens utilisés dans OM sont entièrement basés sur des outils externes contrôlés par des modules rassemblés en lignes de commande ou OSC.

2.3.Fonctionnalités audio dans une application musicale

Nous allons ici dresser un bref état de l'art des fonctionnalités existantes dans les logiciels audio classiques. Il n'est pas dans l'optique d'OpenMusic de devenir un logiciel audio commun, mais il peut être intéressant d'intégrer des fonctionnalités basiques, de manière à améliorer le *workflow* et l'horizon compositionnel proposé.
Voici une liste des fonctionnalités présentes dans les logiciels audio classiques, qui peuvent potentiellement trouver leur place au sein d'OpenMusic :

- La récupération des informations sur un fichier audio (format, fréquence d'échantillonnage etc…)
- L'affichage performant de la forme d'onde (waveform),
- Des fonctions d'édition basiques (coupe/colle etc...),
- Un système de pistes efficace,
- Un gestionnaire d'effets performant,
- Une large palette d'automations possibles (courbes de contrôle dynamiques).

Il est à noter qu'OpenMusic intègre déjà certains de ces attributs, mais parfois de manière non optimale. Il sera donc nécessaire de repenser (ou non) ces acquis en fonction des nouvelles intégrations que je réaliserai durant mon stage.

3. Objectifs

3.1.Objectifs de dynamisation

L'objectif de ce stage est d'envisager un mode d'intégration dynamique entre l'environnement de CAO et les signaux sonores qu'il est susceptible de produire ou manipuler. Ainsi la définition et le contrôle des procédures de synthèse et de traitement des sons pourront s'insérer plus finement au sein de processus musicaux mis en œuvre dans cet environnement. On se dirigera donc vers la création d'un moteur audio, voire d'une API utilisable par le logiciel, distinguant les processus de contrôle et de rendu sonore, rarement conciliables.
Comme dit précédemment, l'architecture actuelle ne permet pas de contrôler les données en cours de lecture (modifier, arrêter etc...). La dynamisation que nous envisageons permettra d'interagir avec celles-ci quel que soit leur état. Les principales fonctionnalités que nous visons sont les suivantes :

- Manipulation des ressources audio,
- Moteurs de production sonore programmables (synthétiseurs),
- Effets audio (DSP) programmables,
- Contrôle dynamique.

Cette implémentation pourra passer par l'étude des différentes librairies existantes et des environnements ou langages de programmation comme Faust, SuperCollider [15] ou encore Max [16]. Le but ne sera pas forcément d'utiliser tous ces outils pour en tirer le maximum, mais plutôt de définir au fur et à mesure les besoins d'OpenMusic de manière à établir une démarche claire et éviter de créer trop de nouvelles dépendances.
Toutes les intégrations et modifications apportées devront s'intégrer pleinement dans l'architecture logicielle actuelle et ne supprimer aucune possibilité offerte par OM.

3.2.Objectifs liés au projet ANR

Dans le cadre du projet ANR INEDIT, je me dois durant mon stage d'essayer de faire interagir au maximum les outils mis à ma disposition par les équipes de recherche. C'est ainsi qu'il sera possible, en plus de dynamiser l'architecture audio du logiciel, d'ouvrir de nouvelles perspectives compositionnelles et proposer un environnement performant et original aux utilisateurs. Dans ces outils, nous retiendrons LibAudioStream et Faust.

3.2.1. Présentation de LibAudioStream

LibAudioStream (LAS) est une bibliothèque C++ créé par le GRAME CNCM, permettant de manipuler des ressources audio à travers le concept de « streams » qui peuvent être manipulés et combinés dans une structure fonctionnelle arborescente. Elle permet alors le développement de *players* audio et de traitements rapides des ressources. Cette bibliothèque dispose d'une API comprenant les fonctions d'accès aux fichiers audio, de mixage, séquençage, application d'effets. Elle embarque également la technologie LLVM [17,18], qui permet la compilation de code à la volée. Cette technologie est utilisée pour compiler du code Faust, langage également créé par le GRAME CNCM, tout comme LAS.
La version de LibAudioStream utilisée pendant ce stage est nouvelle comparée à celle utilisée précédemment dans OpenMusic. Nous utiliserons la version 1.276.

3.2.2. Présentation de Faust

Faust est un langage de spécification pour la description de processeurs audio basé sur une sémantique fonctionnelle, permettant la création de traitements ou de synthétiseurs sonores fonctionnant en temps réel. Les programmes Faust peuvent être déployés sur un grand nombre de plateformes et de formats, et s'articulent notamment avec la bibliothèque LAS.
Faust permet donc à l'utilisateur de créer ses propres effets et synthétiseurs, puis de les contrôler en temps réel. De plus, un grand nombre de bibliothèques sont fournies de manière à ce que des fonctions basiques soient directement accessibles.
La figure 5 montre un exemple de code Faust :

```
declare name        "karplus32";
declare version     "1.0";
declare author      "Grame";
declare license     "BSD";
declare copyright   "(c) GRAME 2006";

//-----------------------------------------------
//              karplus-strong
//      with 32 resonators in parallel
//-----------------------------------------------

import("music.lib");

// Excitator
//---------

upfront(x)  = (x-x') > 0.0;
decay(n,x)  = x - (x>0)/n;
release(n)  = + - decay(n);
trigger(n)  = upfront : release(n) : >(0.0) : +(leak);
leak        = 1.0/65536.0;

size        = hslider("excitation (samples)", 128, 2, 512, 1);

// Resonator
//----------

dur         = hslider("duration (samples)", 128, 2, 512, 1);
att         = hslider("attenuation", 0.1, 0, 1, 0.01);
average(x)  = (x+x')/2;

resonator(d, a) = (+ : delay(4096, d-1.5)) ~ (average : *(1.0-a));

// Polyphony
//----------

detune      = hslider("detune", 32, 0, 512, 1);
polyphony   = hslider("polyphony", 1, 0, 32, 1);

output      = hslider("output volume", 0.5, 0, 1, 0.1);

process =  vgroup("karplus32",
        vgroup("noise generator", noise * hslider("level", 0.5, 0, 1, 0.1))
         : vgroup("excitator", *(button("play")) : trigger(size))
        <: vgroup("resonator x32", par(i,32, resonator(dur+i*detune, att) * (polyphony > i)))
        :> *(output),*(output)
        );
```

Figure 5 : Exemple de code Faust implémentant un synthétiseur Karplus-Strong.

Le compilateur Faust analyse et traduit le code en C++ optimisé et l'intègre dans une architecture cible (Max, Puredata, VST etc…).
De plus, Faust permet une visualisation du processus créé, sous forme de bloc-diagramme exporté en fichier SVG. Ce format de fichier permet un affichage à plusieurs niveaux dans les navigateurs web : on peut alors naviguer à l'intérieur et à l'extérieur des blocs, en affichant le contenant et le contenu de ceux ci. Ainsi il devient très facile de

se représenter le traitement en cours de réalisation. La figure 6 montre un exemple de fichier SVG généré lors de la compilation d'un effet Faust (ici un Flanger) :

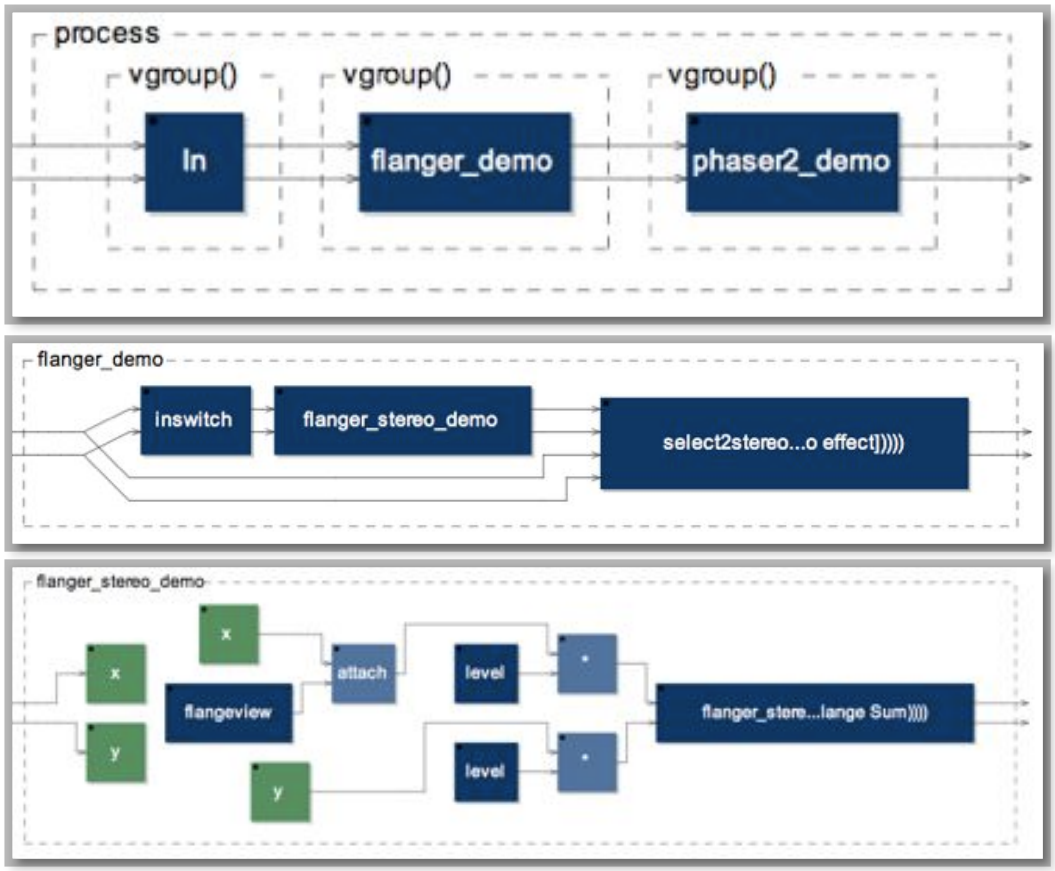


Figure 6 : Exemple de fichier SVG généré par Faust (3 niveaux présentés ici).

<u>Remarque</u> : La sémantique de Faust permet un traitement à l'échantillon. Il n'est pas possible de passer dans le domaine spectral (les fonctions de filtrage ou d'égalisation doivent se faire de manière discrète).

4. Réalisations

4.1.Récupération des métadonnées

Mon premier travail a consisté en la récupération des métadonnées des fichiers audio. Ce travail m'a surtout permis de me familiariser avec la programmation en Lisp, langage qui m'était inconnu lors de mon arrivée à l'Ircam.

OpenMusic était déjà capable de récupérer le titre du fichier, son format (Wav, Aiff etc…) et sa fréquence d'échantillonnage. Seulement, les fonctions récupérant ces métadonnées se basaient sur un mélange de différentes bibliothèques, en allant récupérer une partie de ces données grâce à une, le reste grâce à d'autres.
Mon but fut donc de récupérer ces données de manière claire et uniforme. J'ai également souhaité récupérer le type de données (32bit float, 24bit int etc…) en plus du format.

Pour ce faire, j'ai centralisé les opérations autour de *libsndfile* (une bibliothèque développée en C, pour la lecture et l'écriture de fichiers audio) [19]. Il faut pour cela créer une interface entre le code C et le code Lisp, de manière à pouvoir utiliser en Lisp les fonctions de l'API écrites en C.
L'interfaçage se fait grâce à CFFI (the **C**ommon **F**oreign **F**unction **I**nterface) qui permet de charger des librairies, fournit des opérateurs pour allouer et libérer de la mémoire et permet d'appeler des fonctions externes écrites dans un autre langage [20].

Il fut alors possible de récupérer les métadonnées et de les afficher comme présenté ci-dessous, dans l'éditeur audio (voir figure 7) :

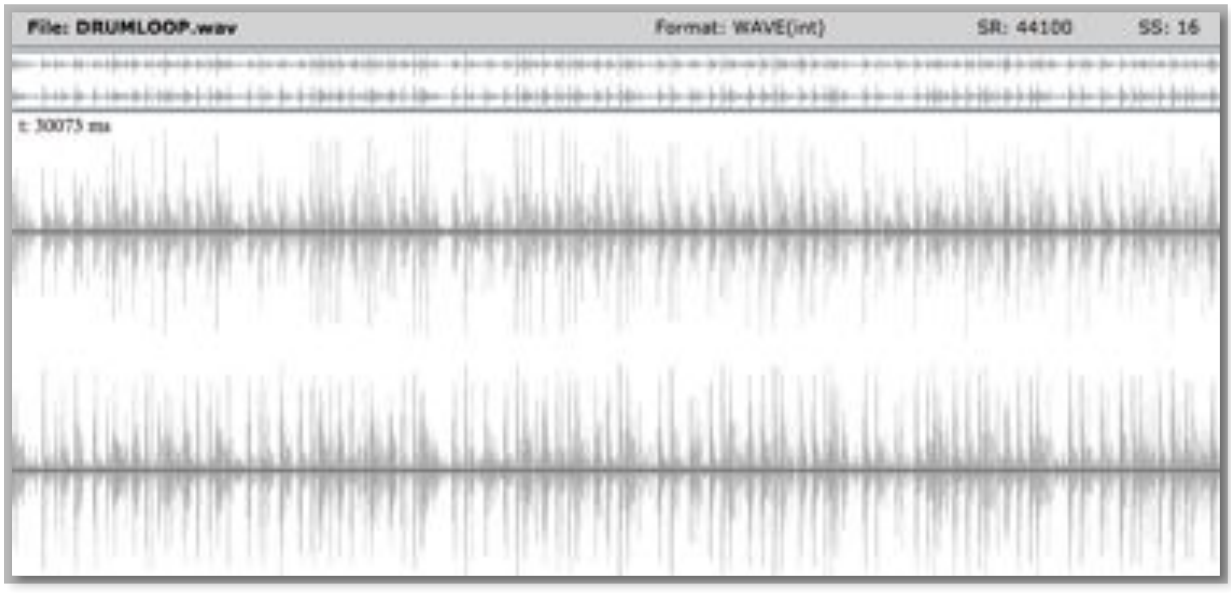


Figure 7 : Affichage des métadonnées dans l'éditeur audio.

La fonction incluant la récupération de ces métadonnées est disponible en annexe 1.

4.2.Affichage de la forme d'onde

Dans la continuité de la récupération des métadonnées d'un fichier audio, mon travail suivant portant sur l'édition audio fut de revoir l'affichage de la forme d'onde des fichiers audio.

Jusqu'ici, l'affichage de cette forme d'onde était disponible grâce à la création d'une image à partir du fichier, puis l'on pouvait zoomer sur celle-ci. Ce système, présentant l'avantage de ne nécessiter aucun calcul suite à la création de l'image, est très peu précis : en effet, la résolution de l'image étant fixe, si l'on zoom de manière trop prononcé, il est impossible d'avoir accès à une information visuelle fiable.

Remarque : la fonction créant l'image d'un fichier audio a une résolution égale à x/npix *,* x *étant la durée du fichier audio en ms et* npix *le nombre de pixels de l'image. On découpe le fichier en* npix *portions, on cherche l'échantillon ayant la valeur maximale* $y_{\max}$ *sur chaque portion, puis on dessine une barre allant de* $-y_{\max}$ *à* $+y_{\max}$.

De manière à rendre cet affichage performant, il est nécessaire d'adapter celui-ci en fonction du zoom soumis par l'utilisateur. Pour des fichiers de grande taille et un zoom de valeur 1.0x, on peut conserver l'image créée par le système (en effet, j'ai souhaité écarter l'hypothèse de suppression de cette image, car permettant une prévisualisation rapide du fichier en limitant le temps de calcul). Pour des fichiers plus courts ou des niveaux de zoom supérieurs, il faut afficher les échantillons et les relier entre eux. Cependant, il est impossible d'envisager un affichage générique à l'échantillon près, qui serait beaucoup trop gourmand en temps de calcul et d'affichage. Il faut donc, en fonction du niveau de zoom, afficher les échantillons avec une résolution plus ou moins grande.

Pour cela, j'ai écrit une fonction recevant entre autre comme argument une valeur nommée *smpstep*, représentant le nombre d'échantillons que l'on souhaite « sauter » lors de l'affichage, de manière à n'afficher qu'un échantillon sur *smpstep*, et ainsi gagner en temps de calcul. Attention, *smpstep* représente un nombre d'échantillon, mais il apparaît comme une durée (le nombre d'échantillons correspondant sera calculé en fonction de la fréquence d'échantillonnage du fichier). Selon le niveau de zoom (la durée affichée par l'utilisateur), on fera varier ce *smpstep* jusqu'à obtenir un affichage à l'échantillon près pour un niveau de zoom précis. C'est là que réside la différence avec la méthode précédente : *smpstep* joue le rôle de *npix* mais change selon le niveau de zoom. Aucune image n'est donc pré calculée, l'affichage est dynamique.

Après des tests d'affichage et des évaluations de performances, il est apparu que le meilleur compromis entre qualité et temps d'affichage se trouve aux alentours de 7000 échantillons dans une fenêtre. Par conséquent, j'ai pu calculer le *smpstep* nécessaire en fonction de la durée du fichier affiché. La figure 8 montre ces résultats, présentant des arrondis parfois grossiers étant donné qu'il ne serait pas performant de créer de trop nombreux niveaux de zoom :

Durée affichée (intervalle en ms)	Smpstep (en µs)	Nombre d'échantillons affichés
[250000 ; 300000]	50000	[5000; 6000]
[190000 ; 250000]	40000	[4750; 6250]
[130000 ; 190000]	30000	[4333; 6333]
[80000 ; 130000]	20000	[4000; 6500]
[40000 ; 80000]	10000	[4000; 8000]
[25000 ; 40000]	5000	[5000; 8000]
[20000 ; 25000]	4000	[5000; 6250]
[14000 ; 20000]	3000	[4666; 6666]
[7500 ; 14000]	2000	[3750; 7000]
[5000 ; 7500]	1000	[5000; 7500]
[3000 ; 5000]	800	[3750; 6250]
[1750 ; 3000]	400	[4375; 7500]
[900 ; 1750]	200	[4500; 8750]
[500 ; 900]	100	[5000; 9000]
[1 ; 500]	1 sample	[1; SampleRate/2]

Figure 8 : Tableau des niveaux de zoom.

Si la durée affichée est supérieure à 300000ms, on affiche l'image calculée par le système.
La figure 9 montre un comparatif de l'ancien affichage et du nouveau pour deux niveaux de zoom extrêmes :

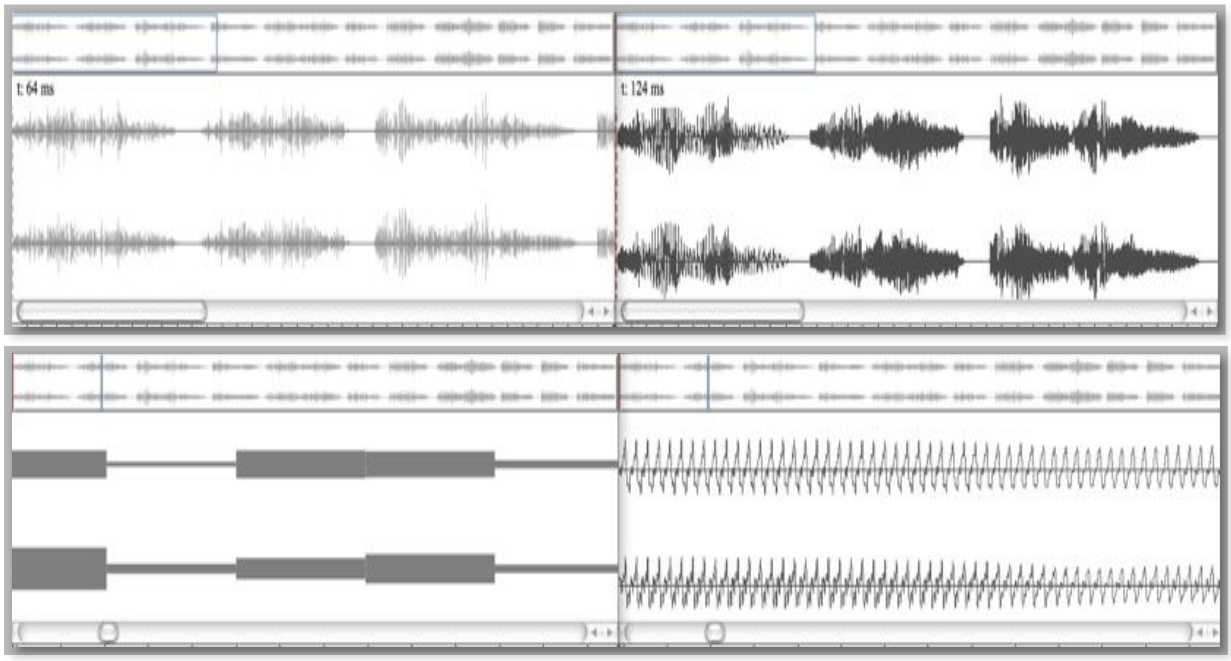


Figure 9 : Comparatif ancien affichage / nouvel affichage de la waveform d'un fichier stéréo.

(En haut : zoom lointain / En bas : zoom proche).

On peut voir que pour des durées affichées grandes, l'affichage n'est que peut modifié. En revanche, dans le cas d'un zoom proche, la différence est grande (plus aucun perte d'information).
La fonction décrivant ce processus d'affichage est disponible en annexe 2.

4.3.Architecture audio

La redéfinition de l'architecture audio a constitué le cœur de mon stage. Mon but fut, outre de respecter les consignes définies dans les objectifs, de proposer une structure innovante avec laquelle un compositeur pourrait satisfaire tous ses besoins.
Mon stage se déroulant dans le cadre du projet ANR INEDIT, j'ai souhaité construire cette architecture atour de LibAudioStream. Cette bibliothèque décrite précédemment convient aux besoins d'OpenMusic, et permettra une collaboration étroite avec le GRAME.

4.3.1. Player LibAudioStream

LibAudioStream permet la création de *players*. Un player se construit grâce à la fonction suivante, fournie par la bibliothèque :

OpenAudioPlayer(*inchan*,*outchan*,*channels*,*srate*,*buffsize*,*streambuffsize*,*instreamdur*,*renderer*,*thread*)

avec :

- Inchan : nombre d'entrées audio,
- Outchan : nombre de sorties audio,
- Channels : nombre de pistes,
- Srate : fréquence d'échantillonnage (sample rate),
- Buffsize : taille du buffer de la sortie audio (en nombre d'échantillons),
- Streambuffsize : taille du buffer de l'entrée audio (en nombre d'échantillons),
- Instreamdur : durée maximale en échantillons de l'entrée audio temps réel,
- Renderer : moteur de rendu (CoreAudio, Jack ou PortAudio),
- Thread : nombre de threads basse priorité additionnels utilisés pour pré calculer les données.

Un player LAS repose sur l'idée d'un mixeur multipistes et impose donc l'utilisation de pistes pour le rendu des ressources audio. Une fois instancié, on démarre le player, puis on peut charger ses pistes avec des streams obtenus par les fonctions de la bibliothèque (voir section 3.2.1). Travaillant à présent avec des « streams » LAS, j'ai accès aux fonctions de coupe/colle de l'API, que j'ai appliqué dans l'éditeur audio pour l'édition des fichiers. Ces fonctions étant directement tirées de l'API, ne seront pas détaillées ici.

Souhaitant apporter plus de flexibilité à l'architecture audio, j'ai décidé d'implémenter un système avec et sans pistes, tout en travaillant avec la définition d'un player LibAudioStream. Pour ce faire j'ai réfléchi à une implémentation de deux players simultanément.

4.3.2. Architecture à deux players

Pour proposer un moyen de « contourner » le système de pistes, j'ai imaginé le système suivant :

- Un player nommé *audio-player-visible*, dont la gestion des pistes est accordée à l'utilisateur,

- Un player nommé *audio-player-hidden*, dont la gestion des pistes n'est pas accessible par l'utilisateur, mais sera automatisée.

Concrètement, pour chaque fichier audio, l'utilisateur pourra soit :

- choisir une piste, et alors envoyer explicitement son fichier audio sur une piste du player *audio-player-visible*,
- choisir la piste 0, et ainsi envoyer son fichier audio sur le player *audio-player-hidden*, sans avoir aucune notion de piste mais plutôt de ressource audio individuelle.

En réalité, le *audio-player-hidden* dispose bien d'un système par piste étant donné que c'est un player instancié via LibAudioStream, mais la répartition de données sur celui-ci est automatisé par un gestionnaire créé par mes soins. La figure 10 représente un schéma récapitulatif de l'architecture envisagé :

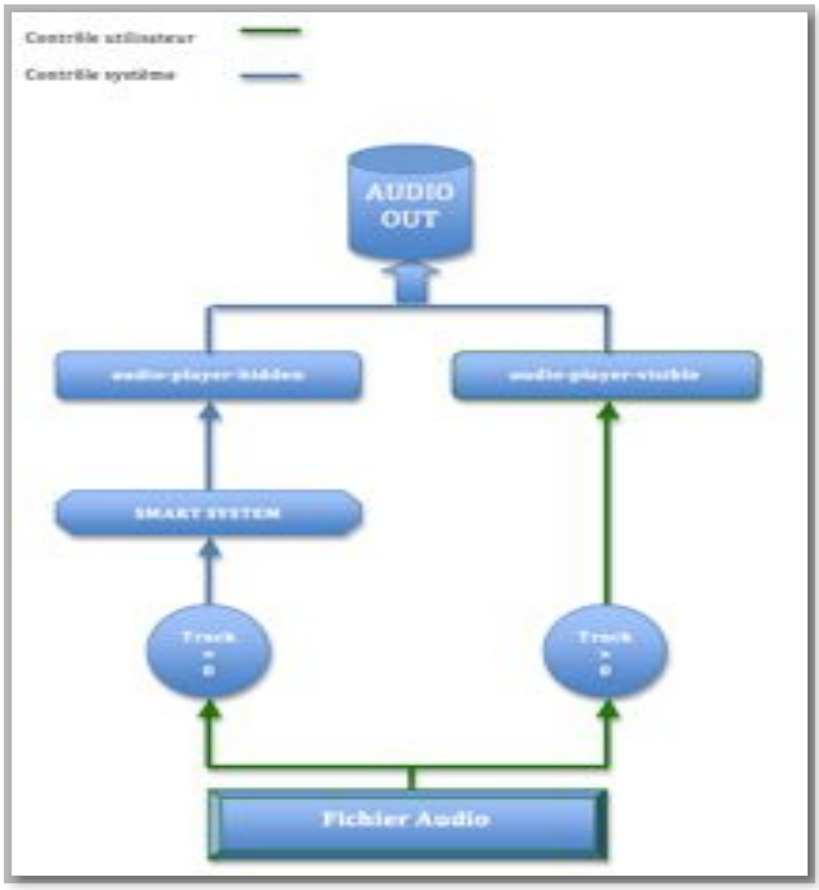


Figure 10 : Schéma explicatif de l'architecture à deux players.

Le « Smart System » présent dans le schéma ci-dessus consiste en un registre, actualisé en permanence, et rendant compte de l'état de chacune des pistes du *audio-player-hidden*. On peut ainsi connaître l'état de chaque piste et répartir les données nouvelles du compositeur sur des pistes libres, dans que celui-ci ne sache qu'il utilise en réalité un système de pistes. Une description de ce système est disponible en annexe 3.
On distinguera ainsi deux méthodes de travail possibles pour le compositeur : contrôle complet du système avec le player par piste et contrôle de l'objet uniquement avec le player « sans » pistes.

La figure 11 représente un schéma détaillant la procédure d'affectation d'une ressource audio grâce au « Smart System » :

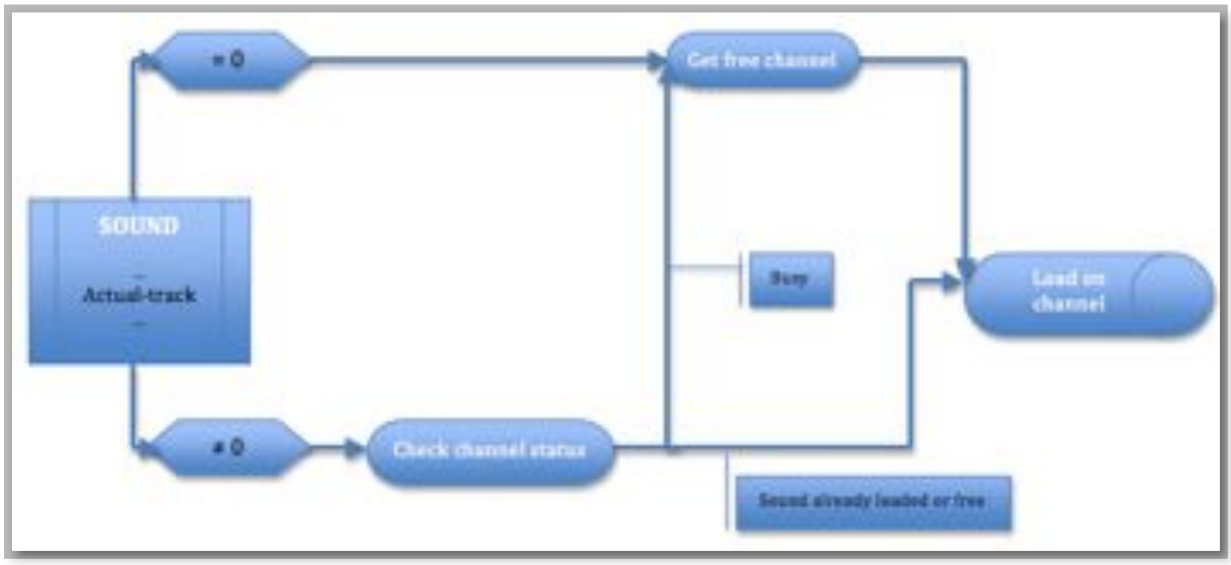


Figure 11 : Récapitulatif du "Smart System"

Cette solution a pu être implémentée, car très peu couteuse pour le système. Elle permet au compositeur plus d'expressivité et de rapidité d'exécution car, contrairement à un player de CAO classique, il n'est pas obligé d'utiliser un système de piste lorsqu'il n'en a pas besoin.

Une des limitations de ce système est le nombre de pistes, qui même s'il est élevé, reste en nombre fini (pour l'instant, chacun des players implémentés compte 32 pistes, donc 64 au total. Ce nombre sera sujet à modification dans LibAudioStream. Pour palier d'éventuels problèmes liés à cette limitation, une sécurité est implémentée prévenant le compositeur lorsque toutes les pistes sont occupées. Sinon, il est possible d'augmenter le nombre de players, et donc de pistes.

4.4.Intégration de Faust

Pour pouvoir proposer des moyens de production audio (synthétiseurs) programmables et de traitement du signal (effets) également programmables et contrôlables, je me suis tourné vers le langage Faust (3.2.2).
Un effet Faust est une unité de traitement audio : il reçoit en entrée un signal audio, lui applique des transformations et fournit le signal transformé en sortie.
Un synthétiseur est une unité de production audio : sans entrée, il crée un signal audio, qu'il fournit en sortie.
Le but est d'obtenir un moyen de créer ses propres effets et synthétiseurs, et de les rendre contrôlables comme on le désire. Sachant qu'un compositeur ne connaît pas forcément ce langage, ou ne souhaite pas le pratiquer, il existe de nombreux effets et synthétiseurs déjà « codés » disponibles. On pourra ainsi disposer des synthétiseurs et effets basiques sans avoir à programmer soi même. Il restera bien sûr possible d'approfondir et modifier ces objets.

Lors de cette intégration, même si les effets et synthétiseurs peuvent être codés et compilés de la même manière, j'ai souhaité les distinguer en deux objets remplissant des fonctions distinctes. J'ai créé deux classes : *FAUST-EFFECT* et *FAUST-SYNTH*.

4.4.1. Effets

Nous allons décrire ici le comportement de l'objet *FAUST-EFFECT*.
Pour rappel, les objets OpenMusic se comportent comme des fonctions. Ils possèdent donc une ou des entrées (arguments) et sorties. Les arguments d'une *FAUST-EFFECT* sont les suivants :

- Un objet *Textfile*, éditeur de texte déjà implémenté dans le logiciel. C'est dans cet objet que le compositeur éditera du code Faust,
- Un nom, sous forme de chaine caractère,
- Un numéro de piste (optionnel) si le compositeur souhaite brancher son effet immédiatement sur une piste du player.

Le rôle de cet objet va être de :

- Compiler le code fourni (renvoyer une erreur si la compilation échoue),
- Enregistrer l'effet dans un registre, avec le nom choisi ou un nom par défaut si aucun nom n'est soumis,
- Eventuellement le brancher à la piste choisie, ou ne rien faire si aucune piste n'est spécifiée,
- Proposer une interface graphique pour le contrôle de cet effet.

La compilation du code Faust se fait via la fonction suivante de l'API de la bibliothèque LibAudioStream :

MakeFaustAudioEffect(name,library_path,draw_path);

avec :

- Name : le code Faust,
- Library_path : le chemin vers les bibliothèques (permet l'accès à des effets basiques déjà créés),
- Draw_path : le chemin où l'on souhaite que le fichier de description SVG s'écrive.

En retour de cette fonction, on obtient un pointeur vers l'effet créé. Si le pointeur est un pointeur nul, cela signifie que la compilation a échoué. On peut alors récupérer l'erreur du compilateur via la fonction suivante, qui la retourne sous forme de chaine de caractères :

GetLastLibError()

Une fois l'effet compilé, on souhaite construire son interface graphique. LibAudioStream permet de récupérer un fichier Json détaillant la structure graphique d'un effet, grâce à la fonction suivante :

GetJsonEffect(effect)

avec :

- Effect : Pointeur vers un effet Faust.

En Lisp, il existe quelques parseurs de Json [21]. Certains permettent d'obtenir une représentation du fichier sous forme de liste (Yason [22]). Ensuite, je pourrais utiliser cette liste pour créer des objets père/fils, et ainsi obtenir un arbre détaillant la représentation graphique de l'effet. Cette partie de récupération de la structure graphique n'étant pas le cœur de la problématique de mon stage, je ne vais pas détailler sa conception. Cependant, n'ayant jamais créé de parseur auparavant, ce travail a été très formateur, notamment le fait d'écrire un parseur entièrement récursif, et donc avec un code très mince. Ce code, ainsi que la spécification Faust Json sont disponibles en annexe 4 et 5.

La bibliothèque LibAudioStream nous ayant renvoyé un fichier SVG, nous proposons à l'utilisateur d'y accéder via un navigateur internet. Connaissant le chemin où le fichier à été créé, il nous suffit de créer un bouton appelant une ligne de commande d'ouverture du fichier.

La figure 12 illustre la création d'un effet Faust, ainsi que son interface graphique :

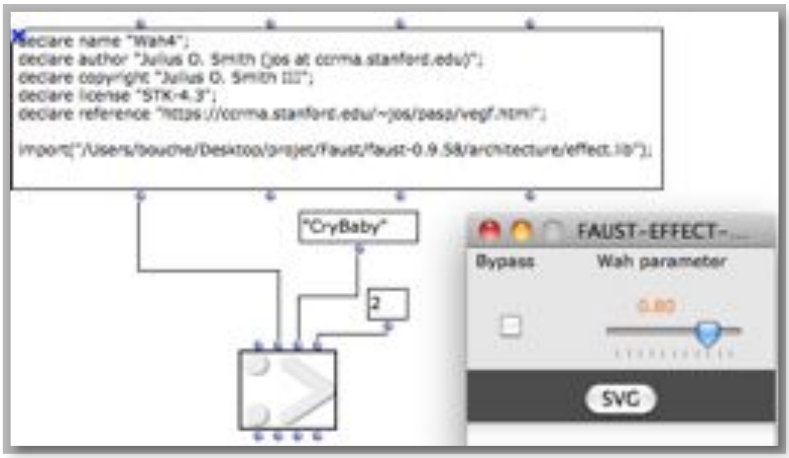


Figure 12 : FAUST-EFFECT pour l'effet CrybBay (Wah-Wah) et son résultat graphique.

4.4.2. Synthétiseurs

Le fonctionnement des synthétiseurs est similaire à celui des effets. En effet, la compilation et création de l'interface graphique s'effectuent de la même manière. Cependant, on peut observer un léger changement dans les entrées de la *FAUST-SYNTH*, dont les arguments sont les suivants :

- Un objet *Textfile*,
- Un nom, sous forme de chaine caractère,
- Un numéro de piste (optionnel) si le compositeur souhaite brancher son synthétiseur immédiatement sur une piste du player,
- Une durée, durant laquelle le synthétiseur produira du son.

En comparaison avec l'objet *FAUST-EFFECT,* j'ai donc simplement ajouté un argument d'entrée : la durée. La séparation entre effet et synthétiseurs est marquée également dans les registres : on ne peut pas autoriser le branchement de plusieurs synthétiseurs sur la même piste (comme dit précédemment, un synthétiseur est une unité de production, il n'est donc pas possible d'avoir plusieurs unités de production sur la même piste). Il est donc nécessaire de tenir deux registres différents pour les effets et synthétiseurs, les effets pouvant être branchés en cascade.

Pour ce qui est de l'intégration dans la nouvelle architecture audio, j'ai souhaité que l'on puisse considérer les synthétiseurs comme des objets à part entière. Dans une architecture classique, un synthétiseur se branche sur une piste audio. Ici, on retrouve ce schéma avec le *audio-player-visible*. Pour qu'ils puissent s'utiliser sans ce système, et donc sur le *audio-player-hidden*, j'ai du créer un système de recherche de piste libre (comme pour le *Smart System* cité précédemment), mais au lieu de charger un fichier audio dans une piste libre, je branche un synthétiseur. Etant donné que l'on alloue soit des fichiers soit des synthétiseurs sur ces pistes « cachées », il est nécessaire de débrancher un synthétiseur une fois son temps de jeu écoulé. Pour cela, on utilise les fonctions *om-synth-preview-play* et *om-synth-preview-stop* décrites en annexe 6.

4.4.3. Automations

On peut déjà, grâce à l'interface graphique obtenue grâce au fichier Json (voir section 4.4.1), contrôler les paramètres des effets et synthétiseurs en temps réel via des *sliders* et boutons.

Dans cette partie, nous allons étudier la partie « contrôle » de l'architecture audio. Par contrôle, on entend bien sûr contrôle temps réel des paramètres, mais surtout écriture de l'interaction. Par exemple, si l'on crée un synthétiseur ayant parmi ses paramètres un paramètre de fréquence, on souhaite pouvoir écrire l'évolution de celui-ci au cours du temps. Ainsi, synthétiseur et effets pourront être un terrain de composition, et non pas des objets « statiques ».

Pour cela, j'ai créé une classe nommée *faust-automation*. Cette classe, dérivée d'une classe déjà existante nommée *bpf* et permettant l'édition de courbe, se comporte comme un objet audio, c'est à dire qu'elle possède une durée (variable) et est lue via une fonction « play ». Les boîtes représentant cette classe possèdent comme entrées (arguments) :

- X-coordinates : les abscisses des points que l'on souhaite voir apparaître (si l'on ne souhaite pas éditer la courbe manuellement, mais plutôt calculer ses points),
- Y-coordinates : les ordonnées de ces même points,
- Precision : la précision décimale des valeurs des points,
- Faust-control : une liste comprenant une *FAUST-EFFECT* ou une *FAUST-SYNTH*, et le nom d'un paramètre.

Les figures 13 et 14 illustrent comment ces automations sont représentées et éditées graphiquement :

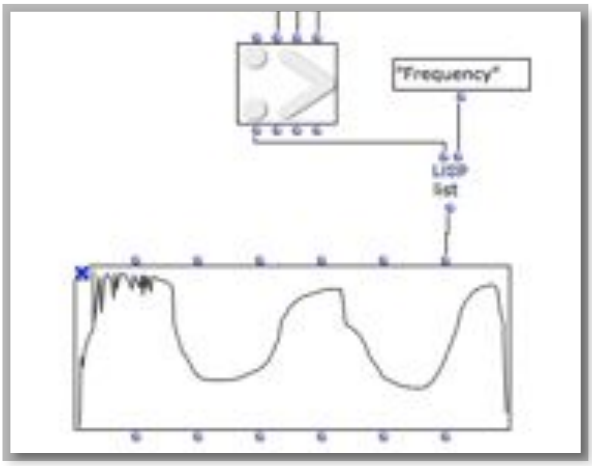


Figure 13 : Faust-automation, les automations des effets et synthétiseurs.

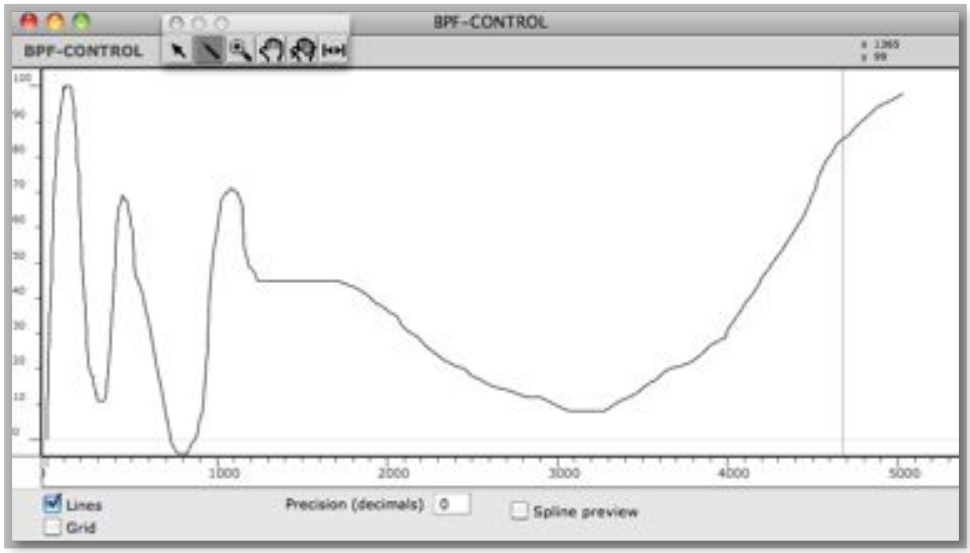


Figure 14 : Edition d'une courbe d'automation (Faust-automation).

Cette Faust-automation applique donc, à chaque passage du curseur sur un point, une fonction prenant comme argument la valeur du point rencontré.

La fonction permettant la construction de la fonction souhaitée (incluant les appels à LibAudioStream) et la fonction de scheduling des appels en fonction de la courbe sont disponibles en annexe 7.

4.5.General mixer

<u>Remarque :</u> Dans cette partie, seul le fonctionnement global du « General Mixer » sera décrit. Le développement étant surtout un travail graphique, aucun code ne figure ici.

Le « General mixer » est une table de mixage virtuelle (voir figure 15). Celle-ci doit permettre une gestion complète du player *audio-player-visible*, ainsi que de tous les effets et synthétiseurs créés. En effet, on souhaite que le compositeur n'ait pas à supprimer puis recréer un effet pour le changer de piste par exemple. Tout doit pouvoir se faire via une interface graphique. La difficulté majeure ici a été de réaliser de nombreux tests de manière à tester tous les cas d'utilisation possibles. Pour citer un exemple simple d'un problème : si l'on n'empêche pas le branchement multiple d'un

même effet, l'utilisateur pourra le faire (LibAudioStream est peu sécurisée de ce point de vue), et cela entraîne des craquements audio importants, rendant l'écoute impossible (et pouvant causer des dommages physiques lors de l'utilisation au casque). On portera donc une attention à la « sécurité » de cette table de mixage.

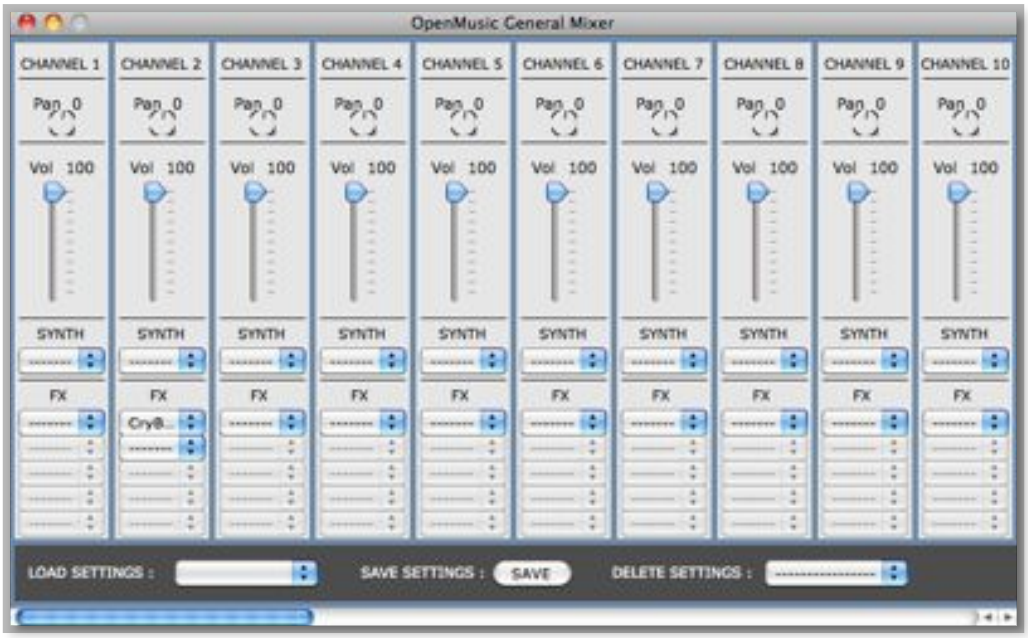


Figure 15 : General Mixer.

On peut voir sur la figure 15 que, via cette table de mixage virtuelle, on fournit les contrôles suivants :

- Volume et Panoramique de chacune des pistes,
- Possibilité de brancher/débrancher un unique synthétiseur par piste,
- Possibilité de brancher/débrancher cinq effets par piste (extensible),
- Possibilité de charger/sauvegarder/supprimer des réglages de cette table.

5. Perspectives

5.1.Travaux à terminer

Le présent rapport devant être rédigé avant ma soutenance, ayant lieu un mois avant la fin effective de mon stage, je vais avoir l'occasion de terminer les travaux que je n'ai pu terminer. Pendant un mois, mes objectifs sont de :

- Effectuer une batterie de tests approfondis sur les développements effectués, et corriger les bugs éventuels,
- Rétablir une fonction d'enregistrement audio avec les nouvelles fonctionnalités de LibAudioStream. Une nouvelle entrée temps réel a été ajoutée dans la bibliothèque permettant d'utiliser les données en cours d'enregistrement, ce qui ouvre de nouvelles perspectives sur l'interaction des flux audio d'entrée/sortie.

5.2.Ouvertures

L'architecture développée durant mon stage est fonctionnelle et stable. Cependant, on pourra très certainement continuer de l'améliorer. Il sera également intéressant d'observer plus précisément les possibilités ouvertes par les différents outils implémentés, et d'adapter le logiciel en fonction de celles ci.

Pour ce qui est du projet ANR INEDIT, une grande interopérabilité a été créée avec les outils du GRAME. Pour ce qui est du LaBRI où des technologies sont en cours de développement, il pourra être intéressant de créer un protocole de communication entre le logiciel i-score et OpenMusic, ainsi qu'éventuellement intégrer libTuiles [23], permettant de pousser encore plus loin l'interactivité dans l'écriture du temps et dans la manipulation des fichiers audio.

Aussi, les développements réalisés durant mon stage ont été en permanence mis à jour sur une branche séparée du dossier SVN [24] OpenMusic. Il est prévu de transférer mon travail sur la branche principale et de proposer une nouvelle version du logiciel pour l'automne prochain.

6. Conclusion

Pour ma part, le bilan de ce stage est très positif. Il m'a permis à la fois de mettre en œuvre les connaissances acquises au fil de mes études, d'approfondir certains points, mais également d'apprendre beaucoup de choses nouvelles. Cette expérience confirme mon envie de poursuivre une carrière dans l'informatique appliquée à la musique, et très certainement dans la recherche dans ce domaine. En effet, les différentes collaborations avec des équipes de recherche ont été pour moi une grande source de motivation. J'ai pu présenter à de nombreux chercheurs mes travaux, mais aussi réfléchir avec eux aux problématiques soulevées par mon travail. Aussi, avoir eu la chance de travailler sur l'architecture audio, et donc le cœur d'un logiciel phare de l'Ircam est une grande source de satisfaction.

Du point de vue de l'Ircam, le bilan est également positif. Il est envisagé par mon encadrant de stage que l'on poursuive en thèse les travaux entrepris durant ce stage, notamment la résolution des problématiques d'ordonnancement posées par l'intégration de calculs temps réel au sein d'un environnement de CAO. Nous comptons également rester en contact, notamment pour ce qui est de l'intégration de mes développements dans la version officielle du logiciel pour mener le projet à bien.

Aussi, j'envisage de poursuivre mes études l'an prochain, au sein du master ATIAM de l'Ircam, pour parfaire ma formation dans le domaine de l'informatique musicale (traitement du signal, acoustique, informatique etc...).

ANNEXES

Annexe 1 : Récupération des métadonnées

```
;/SOUND GET INFO FUNCTION
;This function get the metada from a sound only with libsndfile.
(defun sound-get-info (filename)
  (let ((format_save (audio-file-type filename)))
    (when (om-supported-audio-format format_save)
        (multiple-value-bind (format nch sr ss size skip)
            ;Here is the call to a libsndfil library function.
            (oa::sndfile-get-info (convert-filename-encoding filename))
            (values format nch sr ss size skip)))))

;/FILL SOUND INFO FUNCTION
;This function fill all the slots of a sound object including pointer to stream etc...
(defun om-fill-sound-info (sound)
  (when (and sound (filename sound) (probe-file (filename sound)))
    (print (format nil "Loading sound file : ~s" (namestring (filename sound))))
    (multiple-value-bind (format nch sr ss size skip)
        ;HERE IS THE ONLY CALL TO GET METADATA, CALLING SOUND-GET-INFO FUNCTION.
        (sound-get-info (filename sound))
      ;THEN ITS TO FILL SLOTS, NOT METADATA.
      (if (and format size nch (> size 0) (> nch 0))
          (progn
            (setf los-infos (los-get-sound-infos (om-path2cmdpath (filename sound))))
            (setf (audio-format sound) format
                  (number-of-samples sound) size
                  (number-of-channels sound) nch
                  (sample-size sound) ss
                  (sample-rate sound) sr
                  (data-position sound) skip
                  (sndbuffer sound) (multiple-value-bind (data size nch)
                                        (oa::load-audio-data (oa::convert-filename-encoding (om-sound-file-name sound)) :float)
                                      (let* ((sndbuffer data)) sndbuffer))
                  (sndloaptr sound) (car los-infos)
                  (sndloaptr-current sound) (sndloaptr sound)
                  (sndloaptr-current-save sound) (sndloaptr sound)
                  (number-of-samples-current sound) (cadr los-infos)
                  (sndloaptr-to-play sound) (sndloaptr sound)
                  (number-of-samples-to-play sound) (cadr los-infos)
                  (snd-slice-to-paste sound) nil)
            (setf (loaded sound) t)
            (unless (om-supported-audio-format format)
              (print (format nil "Warning : unsupported audio format ~A" format))
              (setf (loaded sound) :error)))
          (progn
            (print (format nil "Error while loading file ~s" (filename sound)))
            (setf (loaded sound) :error))))
    (loaded sound)))
```

Annexe 2 : Affichage de la waveform

```
;/DRAW WAVEFORM FUNCTION
;This function receive a panel from a sound editor window and a smpstep as arguments.
;Then it draws in the panel the sound waveform according to the smpstep.
(defmethod om-draw-waveform ((self soundPanel) smpstep)
  ;Here we set the local variables we will need.
  (let* ((thesound (object (om-view-container self)))
         (sr (if (om-sound-las-using-srate-? thesound)
                 las-srate
               (om-sound-sample-rate thesound)))
         (timestep (/ (/ sr 1000.0) smpstep))
         (nch (om-sound-n-channels thesound))
         (window-v-size (om-point-v (om-view-size self)))
         (stream-buffer (om-sound-sndbuffer thesound))
         (system-etat (get-system-etat self))
         (xmin (car (rangex self)))
         (pixmin (om-point-h (point2pixel self (om-make-point xmin 0) system-etat)))
         (xmax (cadr (rangex self)))
         (pixmax (om-point-h (point2pixel self (om-make-point xmax 0) system-etat)))
         (xtime (- xmax xmin))
         (basicstep smpstep)
         (channels-h (round window-v-size nch))
         (offset-y (round channels-h 2))
         ;Here is the values we will need to display.
         (datalist (loop for pt from 0 to (* timestep xtime) collect
                         (loop for chan from 0 to (- nch 1) collect
                               (fli::dereference stream-buffer
                                                 :index (+ (* xmin (round sr 1000) nch) (round (* pt basicstep nch)) chan)
                                                 :type :float)))))
    ;Here we draw the level 0 line (1 if mono, 2 if stereo).
    (loop for i from 0 to (- nch 1) do
          (om-draw-line pixmin (+ (* i channels-h) offset-y) pixmax (+ (* i channels-h) offset-y)))
    ;Here we set points and draw lines between them.
    (setf sampleprev (car datalist))
    (loop for sample in (cdr datalist)
          for i = 0 then (+ i 1) do
          (loop for val in sample
                for c = 0 then (+ c 1) do
                (setf pixpoint (round (* offset-y val))) ; scaled 0-1 --> 0 -->256/2
                (setf pixtime (om-point-h (point2pixel self
                                                       (om-make-point
                                                        (+ xmin (* i (/ 1 timestep))) (/ 1 timestep)) 0) system-etat)))
                (setf pixtimeprev (om-point-h (point2pixel self (om-make-point (+ xmin (* i (/ 1 timestep))) 0) system-etat)))
                (om-draw-line  pixtimeprev (+ offset-y (* c channels-h) (round (* offset-y (nth c sampleprev))))
                               pixtime (+ offset-y (* c channels-h) pixpoint)))
          (setf sampleprev sample))))
```

```
;/DRAW CONTENTS FUNCTION
;This function chooses a zoom level and call the om-draw-waveform function with the right smpstep.
(defmethod om-draw-contents ((self soundPanel))
  (call-next-method)
  ;We first run some tests to avoid issues, if there's no sound loaded for example.
  (if (equal (state (player (editor self))) :recording)
      (om-with-focused-view self
        (om-with-fg-color self *om-red2-color*
          (om-with-font *om-default-font4b*
            (let ((halftext (round (om-string-size "Recording" *om-default-font4b*) 2)))
              (om-draw-string (- (round (w self) 2) halftext) (round (h self) 2) "Recording")))))
    (if (om-sound-file-name (object (om-view-container self)))
        ;Here we set the local variables we will need.
        (let* ((thesound (object (om-view-container self)))
               (sr (if (om-sound-las-using-srate-? thesound)
                     las-srate
                   (om-sound-sample-rate thesound)))
               (size (om-sound-n-samples-current thesound))
               (dur (or (and (and sr size)
                             (/ size sr)) 0))
               (dur-ms (round size (/ sr 1000.0)))
               (total-width (om-point-h (om-field-size self)))
               (thepicture (and dur (pic-to-draw thesound)))
               (window-h-size (om-point-h (om-view-size self)))
               (window-v-size (om-point-v (om-view-size self)))
               (stream-buffer (om-sound-sndbuffer thesound))
               (system-etat (get-system-etat self))
               (xmin (car (rangex self)))
               (pixmin (om-point-h (point2pixel self (om-make-point xmin 0) system-etat)))
               (xmax (cadr (rangex self)))
               (pixmax (om-point-h (point2pixel self (om-make-point xmax 0) system-etat)))
               (xview (- xmax xmin))
               (pict-threshold (if (> size (* 5 sr)) (/ dur-ms 3.0) 15000))
               ;Here are all the smpstep we need.
               (step-1smp 1) (step-100us (/ sr 10000.0)) (step-200us (/ sr 5000.0))
               (step-400us (/ sr 2500.0)) (step-800us (/ sr 1250.0)) (step-1ms (/ sr 1000.0))
               (step-2ms (/ sr 500.0)) (step-3ms (/ sr 333.3333)) (step-4ms (/ sr 250.0))
               (step-5ms (/ sr 200.0)) (step-10ms (/ sr 100.0)) (step-20ms (/ sr 50.0))
               (step-30ms (/ sr 33.3333)) (step-40ms (/ sr 25.0)) (step-50ms (/ sr 20.0)))
```

```
(om-with-focused-view self
  (when (and thesound thepicture)
    (om-with-fg-color self *om-dark-gray-color*
      (if (< xview pict-threshold)
          ;Then we can choose which smpstep we choose according to the length of display.
          (cond ((>= xview 3000000) (om-draw-picture self thepicture (om-make-point 0 0)
                                                     (om-subtract-points (om-field-size self) (om-make-point 0 15))))
                ((and (< xview 3000000) (>= xview 2500000)) (om-draw-waveform self step-50ms))
                ((and (< xview 2500000) (>= xview 1300000)) (om-draw-waveform self step-40ms))
                ((and (< xview 1900000) (>= xview 1300000)) (om-draw-waveform self step-30ms))
                ((and (< xview 1300000) (>= xview 800000)) (om-draw-waveform self step-20ms))
                ((and (< xview 800000) (>= xview 400000)) (om-draw-waveform self step-10ms))
                ((and (< xview 400000) (>= xview 250000)) (om-draw-waveform self step-5ms))
                ((and (< xview 250000) (>= xview 200000)) (om-draw-waveform self step-4ms))
                ((and (< xview 200000) (>= xview 140000)) (om-draw-waveform self step-3ms))
                ((and (< xview 140000) (>= xview 75000)) (om-draw-waveform self step-2ms))
                ((and (< xview 75000) (>= xview 50000)) (om-draw-waveform self step-1ms))
                ((and (< xview 50000) (>= xview 30000)) (om-draw-waveform self step-800us))
                ((and (< xview 30000) (>= xview 17500)) (om-draw-waveform self step-400us))
                ((and (< xview 17500) (>= xview 9000)) (om-draw-waveform self step-200us))
                ((and (< xview 9000) (>= xview 5000)) (om-draw-waveform self step-100us))
                ((< xview 5000) (om-draw-waveform self step-1smp)))
        (om-draw-picture self thepicture (om-make-point 0 0) (om-subtract-points (om-field-size self)
                                                                                 (om-make-point 0 15))))
      (om-with-fg-color self *om-blue-color*
        (loop for item in (markers thesound)
              for k = 0 then (+ k 1) do
              (om-with-line-size (if (member k (selection? self)) 2 1)
                (om-draw-line (round (* total-width item) dur) 0 (round (* total-width item) dur) (h self))
                (om-fill-rect (- (round (* total-width item) dur) 2) 0 5 5)))))
    (when (grille-p self)
      (draw-grille self))
    (draw-interval-cursor self)
    (unless thepicture
      (if (and (om-sound-file-name thesound) (zerop dur))
          (om-with-focused-view self
            (om-draw-string 30 30 (format nil "Error: File ~s is empty" (om-sound-file-name (object (editor self))))))
        (om-with-focused-view self
          (om-draw-string (round (w self) 2) (round (h self) 2) "..."))
        )))) (om-with-focused-view self (om-draw-string 10 40 (format nil "You have to load a file."))))
```

Annexe 3 : Smart system

```
;/GET FREE CHANNEL FUNCTION
;Tool that find a free channel (which state is IDLE, no matter if a sound is loaded) starting from 0.
;Returns the first encountered free channel (as an int).
(defun get-free-channel (player)
  (let ((i 0)
        (status "init")
        (status-list (if (eq player *audio-player-hidden*)
                         *audio-player-hidden-tracks-info*
                       *audio-player-visible-tracks-info*)))
    (while (not (string-equal status "Idle"))
      (setf status (cadr (gethash i status-list)))
      (setf freetrack i)
      (incf i)
      )
    freetrack))

;/PLAY FUNCTION FOR HIDDEN PLAYER
;This function works based on a little system that checks if the sound is already loaded :
;           -if yes it uses a basic transport system
;           -if not it assigns it to the first available track
;           -if yes but since it was idle its track was allocated to an other sound, it assigns it to the first available track
(defun om-smart-play-hidden (snd)
  (let* ((actual-track (tracknum-sys snd))
         (player *audio-player-hidden*))
    (if (/= actual-track -1)
        (if (eq snd (car (gethash actual-track *audio-player-hidden-tracks-info*)))
          (cond ((string-equal "Idle" (cadr (gethash actual-track *audio-player-hidden-tracks-info*)))
                 (let ()
                   (load-sound-on-one-channel player snd actual-track)
                   (play-one-channel player actual-track)))
                ((string-equal "Paused" (cadr (gethash actual-track *audio-player-hidden-tracks-info*)))
                 (cont-one-channel player actual-track))
                ((string-equal "Playing" (cadr (gethash actual-track *audio-player-hidden-tracks-info*))) nil))
          (if (string-equal "Idle" (cadr (gethash actual-track *audio-player-hidden-tracks-info*)))
              (let ()
                (load-sound-on-one-channel player snd actual-track)
                (play-one-channel player actual-track))
            (let ((chan (get-free-channel player)))
              (setf (tracknum-sys snd) chan)
              (load-sound-on-one-channel player snd chan)
              (play-one-channel player chan))
            ))
      (let* ((chan (get-free-channel player)))
        (setf (tracknum-sys snd) chan)
        (load-sound-on-one-channel player snd chan)
        (play-one-channel player chan))))))

;/PAUSE FUNCTION FOR HIDDEN PLAYER
;This function is a basic pause function which works only if the sound is playing. It also check if the channel of the sound
;is well loaded with it to avoid issues.
(defun om-smart-pause-hidden (snd)
  (let ((actual-track (tracknum-sys snd))
        (player *audio-player-hidden*))
    (if (eq snd (car (gethash actual-track *audio-player-hidden-tracks-info*)))
        (if (string-equal "Playing" (cadr (gethash actual-track *audio-player-hidden-tracks-info*)))
            (pause-one-channel player actual-track)))))

;/STOP FUNCTION FOR HIDDEN PLAYER
;This function is a basic stop function. It also check if the channel of the sound is well loaded with it to avoid issues.
(defun om-smart-stop-hidden (snd &optional synth)
  (let ((actual-track (tracknum-sys snd))
        (player *audio-player-hidden*))
    (if (eq snd (car (gethash actual-track *audio-player-hidden-tracks-info*)))
        (stop-one-channel player actual-track))))
```

Annexe 4 : Spécification du Json de Faust

```
<json> = {
                    "name"        : "karplus"
                    "address"     : "localhost"
                    "port"              : 5510,
//                  "inputs"      : 2,                still pending
//                  "outputs"     : 2,                still pending
//                  "meta"              : [{"clef":"valeur"}, ...], still pending
                    "ui"          : <ui>
}

<ui> = <group> | <slider> | <nentry> | <button> | <bargraph>

<group> = { "type" : "(vht)group",
                    "label": "excitation",
                    "items": [<ui>...]
                }

<slider>= {
                    "type": "(vh)slider",
                    "label": "excitation",
                    "address": "/karplus/excitator/excitation",
                    "meta": [{"clef" : "valeur"}, ...],
                    "init": "128",
                    "min": "2",
                    "max": "512",
                    "step": "1"
                }

<nentry>= {
                    "type": "nentry",
                    "label": "excitation",
                    "address": "/karplus/excitator/excitation",
                    "meta": [{"clef" : "valeur"}, ...],
                    "init": "128",
                    "min": "2",
                    "max": "512",
                    "step": "1"
                }

<button>= {
                    "type": "button",
                    "label": "excitation",
                    "address": "/karplus/excitator/excitation"
                    "meta": [{"clef" : "valeur"}, ...],
                }

<checkbox>= {
                    "type": "checkbox",
                    "label": "excitation",
                    "address": "/karplus/excitator/excitation"
                    "meta": [{"clef" : "valeur"}, ...],
                }

<bargraph>= {
                    "type" : "(vh)bargraph",
                    "label": "excitation",
                    "address": "/karplus/excitator/excitation",
                    "meta": [{"clef" : "valeur"}, ...],
                    "min" : "2",
                    "max" : "512",
                }
```

Annexe 5 : Parseur Faust

```
;;;=============================================================================
;;;============================ OBJECTS AND ACCESSORS ==========================
;;;=============================================================================
(defclass faust-group ()
  ((group-type :initform nil :initarg :group-type :accessor group-type)
   (label :initform nil :initarg :label :accessor label)
   (items :initform nil :initarg :items :accessor items)
   (size :initform nil :initarg :size :accessor size)))

(defmethod las-faust-get-group-type ((self faust-group))
  (group-type self))
(defmethod las-faust-get-group-label ((self faust-group))
  (label self))
(defmethod las-faust-get-group-items ((self faust-group))
  (items self))
(defmethod las-faust-get-group-size ((self faust-group))
  (size self))


(defclass faust-param ()
   ((param-type :initform nil :initarg :param-type :accessor param-type)
    (label :initform nil :initarg :label :accessor label)
    (osc-address :initform nil :initarg :osc-address :accessor osc-address)
    (metadata :initform nil :initarg :metadata :accessor metadata)
    ;(style :initform nil :initarg :style :accessor style)
    ;(tooltip :initform nil :initarg :tooltip :accessor tooltip)
    ;(unit :initform nil :initarg :unit :accessor unit)
    (init-val :initform nil :initarg :init-val :accessor init-val)
    (min-val :initform nil :initarg :min-val :accessor min-val)
    (max-val :initform nil :initarg :max-val :accessor max-val)
    (step-val :initform nil :initarg :step-val :accessor step-val)
    (size :initform nil :initarg :size :accessor size)))


(defmethod las-faust-get-param-type ((self faust-param))
  (param-type self))
(defmethod las-faust-get-param-label ((self faust-param))
  (label self))
(defmethod las-faust-get-param-address ((self faust-param))
  (osc-address self))
(defmethod las-faust-get-param-metadata ((self faust-param))
  (metadata self))
(defmethod las-faust-get-param-init-val ((self faust-param))
  (init-val self))
(defmethod las-faust-get-param-min-val ((self faust-param))
  (min-val self))
(defmethod las-faust-get-param-max-val ((self faust-param))
  (max-val self))
(defmethod las-faust-get-param-step-val ((self faust-param))
  (step-val self))
(defmethod las-faust-get-param-size ((self faust-param))
  (size self))

;;;=============================================================================
;;;================================ PARSER =====================================
;;;=============================================================================
(defun las-faust-parse (string)
  (let ((children-list (list))
        father
        obj
        type)
    (setf father (construct-faust-ui (get-faust-json-ui string)))
    (group-items-to-objects father)
    (set-group-size father)
    father
    ))

(defun las-faust-translate-tree (tree)
  (let (children final-list)
    (setf children (las-faust-get-group-items tree))
    (if (> (length children) 0)
        (loop for i from 0 to (- (length children) 1) do
              (cond ((typep (nth i children) 'faust-group)
                     (setf final-list (append final-list (las-faust-translate-tree (nth i children)))))
                    ((typep (nth i children) 'faust-param)
                     (setf final-list (append final-list (list (nth i children))))))))
        (setf final-list (list)))
    final-list))

(defun las-faust-get-groups-only (tree)
  (let (children final-list)
    (setf children (las-faust-get-group-items tree))
    (if (> (length children) 0)
        (loop for i from 0 to (- (length children) 1) do
              (cond ((typep (nth i children) 'faust-group)
                     (setf final-list (append final-list (list (nth i children)))))
                    ((typep (nth i children) 'faust-param)
                     nil))))
```

```
  (if (items group)
      (loop for i from 0 to (- (length (items group)) 1) do
            (setf type (check-type-key (nth i (items group))))
            (setf obj
                  (cond ((or (string= type "hgroup") (string= type "vgroup"))
                         (construct-faust-group (nth i (items group))))
                        (t
                         (construct-faust-param (nth i (items group))))))
            (if (<= ground max) (setf top-list (append top-list (list obj))))
            (if (typep obj 'faust-group)
                (group-items-to-objects obj (+ ground 1))
              (set-param-size obj))
            (setf children-list (append children-list (list obj)))))
  (setf (items group) children-list)
  (if (<= ground max) (setf (items group) top-list)))


(defun construct-faust-ui (ui)
  (cond ((or (string= (check-type-key ui) "hgroup") (string= (check-type-key ui) "vgroup"))
         (construct-faust-group ui))
        (t
         (construct-faust-param ui))))

(defun construct-faust-param (list)
  (let (type label address metadata init-val min-val max-val step-val)
    (progn
      (pop list)
      (setf type (pop list))
      (pop list)
      (setf label (list))
      (while (not (string= (car list) "address"))
        (setf label (append label (pop list))))
      (pop list)
      (setf address (pop list))
      (if (string= (car list) "meta")
          (progn
            (pop list)
            (setf metadata (pop list))))
      (pop list)
      (setf init-val (pop list))
      (pop list)
      (setf min-val (pop list))
      (pop list)
      (setf max-val (pop list))
      (pop list)
      (setf step-val (pop list))
      (make-instance 'faust-param
                     :param-type type
                     :label label
                     :osc-address address
                     :metadata metadata
                     :init-val (if init-val init-val "0")
                     :min-val (if min-val min-val "0")
                     :max-val (if max-val max-val "10000000")
                     :step-val step-val))))

(defun construct-faust-group (list)
  (let (type label items)
    (progn
      (pop list)
      (setf type (pop list))
      (pop list)
      (setf label (list))
      (while (not (string= (car list) "items"))
        (setf label (append label (list (pop list)))))
      (pop list)
      (setf items (pop list))
      (make-instance 'faust-group
                     :group-type type
                     :label label
                     :items items))))

(defun check-type-key (list)
  (let ((curkey (pop list)) res)
    (if (string= curkey "type")
        (setf res (pop list)))
    res))

;;;=============================== GRAPH TOOLS ================================
;;;============================================================================
(defconstant buttonSize (list 124 40))
(defconstant checkboxSize (list 80 81))
(defconstant hsliderSize (list 124 81))
(defconstant vsliderSize (list 80 154))
(defconstant numentrySize (list 80 65))
```

```
(defmethod set-param-size ((self faust-param))
  (cond ((string= (param-type self) "checkbox") (setf (size self) checkboxSize))
        ((string= (param-type self) "button") (setf (size self) buttonSize))
        ((string= (param-type self) "hslider") (setf (size self) hsliderSize))
        ((string= (param-type self) "vslider") (setf (size self) vsliderSize))
        ((string= (param-type self) "numentry") (setf (size self) checkboxSize))
        (t (setf (size self) (list 25 25)))))

(defmethod set-group-size ((self faust-group))
  (let ((size (list 0 0))
        (type (group-type self))
        (x 0)
        (y 0)
        (max 0)
        curitem
        cursize)
    (if (string= type "hgroup")
        (progn
          (loop for i from 0 to (- (length (items self)) 1) do
                (setf curitem (nth i (items self)))
                (if (typep curitem 'faust-group) (set-group-size curitem))
                (setf cursize (size curitem))
                (setf x (+ x (car cursize)))
                (if (> (cadr cursize) max) (setf max (cadr cursize))))
          (setf size (list x max)))
      (progn
        (loop for i from 0 to (- (length (items self)) 1) do
              (setf curitem (nth i (items self)))
              (if (typep curitem 'faust-group) (set-group-size curitem))
              (setf cursize (size curitem))
              (setf y (+ y (cadr cursize)))
              (if (> (car cursize) max) (setf max (car cursize))))
        (setf size (list max y))))
    (setf (size self) size)))
```

Annexe 6 : Synthétiseur Play et Stop

```
;/SYNTH PREVIEW PLAY FUNCTION
;This function plays a preview of a selected synth, on the track which it's plugged or on the hidden player if it's not plugged.
(defun om-synth-preview-play (obj)
  (let ((search-res (las-faust-search-synth-console-in-register obj)))
    (if (car search-res)
        (let* ((info (cadr search-res))
               (synth-ptr (nth 1 info))
               (nullsnd (nth 2 info))
               (actual-track (trocknum-sys nullsnd))
               (res (las-faust-synth-already-plugged-? synth-ptr))
               chan liste)
          (if res
              (progn
                (om-smart-play nullsnd nil nil (+ (car res) 1)))
            (if (not (las-faust-synth-hidden-already-plugged-? synth-ptr))
                (progn
                  (if (/= -1 actual-track)
                      (setf chan actual-track)
                    (setf chan (get-free-channel *audio-player-hidden*)))
                  (setf (gethash chan *faust-synths-by-track-hidden*) synth-ptr)
                  (las::AddAudioEffect (gethash chan *effects-lists-hidden*) synth-ptr)
                  (om-smart-play nullsnd))))))))

;/SYNTH PREVIEW STOP FUNCTION
;This function stops a preview of a selected synth.
(defun om-synth-preview-stop (obj)
  (let ((search-res (las-faust-search-synth-console-in-register obj)))
    (if (car search-res)
        (let* ((info (cadr search-res))
               (synth-ptr (nth 1 info))
               (nullsnd (nth 2 info))
               (res (las-faust-synth-already-plugged-? synth-ptr))
               chan liste)
          (if res
              (om-smart-stop-visible nullsnd (car res))
            (let ((chan1 (find-synth-hidden synth-ptr)))
              (if chan1
                  (progn
                    (remove-faust-effect-from-list synth-ptr (gethash chan1 *effects-lists-hidden*))
                    (setf (gethash chan1 *faust-synths-by-track-hidden*) nil)))
              (om-smart-stop-hidden nullsnd))))))
```

Annexe 7 : Bpf-control : Faust-function builder & Scheduler

```
;/GET FUNCTION FROM FAUST CONTROL FUNCTION
;This function gets a faust-control as an argument (which is a list of a Faust FX or Synth and parameter name)
;and return the function which will change the value of the selected parameter
(defun get-function-from-faust-control (faust-control)
  (let* ((name (cadr faust-control))
         (console (car faust-control))
         (ptr (if (typep console 'faust-effect-console) (effect-ptr console) (synth-ptr console)))
         (maxnum (las-faust-get-control-count ptr))
         infos minval maxval range found text-to-up display graph-to-up paramtype)
    (loop for i from 0 to (- maxnum 1) do
          (if (string= name (car (las-faust-get-control-params ptr i)))
              (setf found i)))
    (setf infos (las-faust-get-control-params ptr found))
    (setf minval (nth 1 infos)) (setf maxval (nth 2 infos))
    (if (= minval maxval) (setf minval 0
                                maxval 1))
    (setf range (- maxval minval))
    (setf display (display (nth found (params-ctrl console))))
    (setf text-to-up (paramval display)) (setf graph-to-up (paramgraph display))
    (setf paramtype (param-type (nth found (params-ctrl console))))
    (if graph-to-up
        #'(lambda (val)
            (if (< val minval) (setf val minval))
            (if (> val maxval) (setf val maxval))
            (las-faust-set-control-value ptr found (float val))
            (cond ((string= paramtype "checkbox")
                   (om-set-check-box graph-to-up (if (>= val 1) t)))
                  ((string= paramtype "numentry")
                   (progn
                     (om-set-dialog-item-text text-to-up (number-to-string (float val)))
                     (set-value graph-to-up (* 100 (/ (- val minval) range)))))
                  (t
                   (progn
                     (om-set-dialog-item-text text-to-up (number-to-string (float val)))
                     (om-set-slider-value graph-to-up (* 100 (/ (- val minval) range)))))))
      #'(lambda (val)
          (las-faust-set-control-value ptr found (float val))))))

;/PREPARE TO PLAY BPF PLAYER FUNCTION
;This function schedules the calls to a bpf-control function according to the bpf points.
(defmethod prepare-to-play ((self (eql :bpfplayer)) (player omplayer) (object bpf-control) at interval)
  ;Here we get the faust function with the get-function-from-faust-control function
  (let ((faustfun (if (faust-control object)
                      (get-function-from-faust-control (faust-control object)))))
    (when (or (c-action object) (faust-control object))
      ;Scheduling, if an interval is selected
      (if interval
          (mapcar #'(lambda (point)
                      (if (and (>= (car point) (car interval)) (<= (car point) (cadr interval)))
                          (if (faust-control object)
                              (schedule-task player
                                             #'(lambda () (funcall faustfun (cadr point)))
                                             (+ at (car point))))))
                  (point-pairs object))
        ;Scheduling with no interval (all points)
        (mapcar #'(lambda (point)

                    (if (faust-control object)
                        (schedule-task player
                                       #'(lambda () (funcall faustfun (cadr point)))
                                       (+ at (car point)))))
                (point-pairs object))))))
```

Bibliographie

[1] : *J. Bresson, C. Agon, G. Assayag, « OpenMusic Visual Programming Environment for Music Composition, Analysis and Research », ACM Multimedia, Scottsdale 2011.*
[2] : *A. Allombert, G. Assayag, M. Desainte-Catherine, « A system of interactive scores based on Petri nets », Proceedings of Sound an Music Computing Conference, SMC 2007.*
[3] : *D. Fober, Y. Orlarey, S. Letz, « INScore - An Environment for the Design of Live Music Scores », Proceedings of the Linux Audio Conference, LAC 2012.*
[4] : *A. Cont, « ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music », International Computer Music Conference. ICMC 2008.*
[5] : *Y. Orlarey, D. Fober, S. Letz. « Faust : an Efficient Functional Approach to DSP Programming », New Computationals Paradigms for Computer Music, Edition Delatour (France), 2009.*
[6] : *LibAudioStream [Online] http: //libAudioStream.sourceforge.net/*
[7] : *G. Steele, « Common Lisp, The Language – 2nd Edition », Digital Press, 1990.*
[8] : *OpenMusic [Online] http://repmus.ircam.fr/openmusic/download*
[9] : J*. Bresson, C. Agon, « The Sheet Object in OpenMusic », Computer Music Journal, 32(4), 2008.*
[10] : *G. Assayag, C. Rueda, M. Laurson, C. Agon, O. Delerue, « Computer Assisted Composition at Ircam: PatchWork & OpenMusic », Computer Music Journal, 23(3), 1999.*
[11] : *Y. Orlarey, H. Lequay, « MidiShare : a Real Time multi-tasks software module for Midi applications », Proceedings of the International Computer Music Conference, ICMC 1989.*
[12] : *M. Schumacher, J. Bresson, « Compositional Control of Periphonic Sound Spatialization », 2nd International Symposium on Ambisonics and Spherical Acoustics, Paris 2010.*
[13] : *A. Bancquart, C. Agon, M. Andreatta, « Microtonal Composition », in The OM Composer's Book vol 2, Editions Delatour – Ircam 2008.*
[14] : *A. Schmeder, A. Freed, D. Wessel, « Best Practices for Open Sound Control », Proceedings of the Linux Audio Conference, LAC 2010.*
[15] : *J. McCartney. "Rethinking the Computer Music Language : SuperCollider." Computer Music Journal, 26(4), 2002.*
[16] : *M. Puckette. "Combining Event and Signal Processing in the MAX Graphical Programming Environment." Computer Music Journal, 15(3), 1991.*
[17] : *C. Lattner, « LLVM : An Infrastructure for Multi-Stage Optimization », Masters Thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Dec. 2002.*
[18] : *A. Gräf, « Functional Signal Processing with Pure and Faust using the LLVM Toolkit », Proceedings of the Sound and Music Computing Conference, SMC 2011.*
[19] : *libsndfile [Online] http://www.mega-nerd.com/libsndfile/*
[20] : *CFFI [Online] http://common-lisp.net/project/cffi/*
[21] : *JSON [Online] http://www.json.org/*
[22] : *Yason [Online] http://common-lisp.net/project/yason/*
[23] : *D. Janin, F. Berthaut, M. Desainte-Catherine, « Multi-scale design of interactive music systems : the libTuiles experiment », Sound And Music Computing Conference, SMC 2013.*
[24] : *SVN, Subversion [Online] http://subversion.apache.org/*