HAL open science

OpenMusic - LibAudioStream - Faust

Jean Bresson, Dimitri Bouche

▶ To cite this version:

Jean Bresson, Dimitri Bouche. OpenMusic - LibAudioStream - Faust. [Rapport de recherche] IRCAM / ANR-12-CORD-0009 INEDIT. 2013. hal-00959371

HAL Id: hal-00959371
https://hal.science/hal-00959371

Submitted on 14 Mar 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ANR-12-CORD-0009
INEDIT Project
Programme ContInt
http://inedit.ircam.fr/

OpenMusic - LibAudioStream - Faust

Jean Bresson, Dimitri Bouche
IRCAM / UMR 9912 STMS
bresson@ircam.fr, bouche@ircam.fr

Résumé

Ce document présente un rapport du WP 3.3.a portant sur l'interopérabilité de l'environnement de composition assistée par ordinateur OpenMusic avec les outils développés par nos partenaires du projet INEDIT. Nous nous sommes concentrés sure les outils de traitement et de rendu audio développés par Grame, c'est-à-dire la bibliothèque LibAudioStream et les langage de traitement et synthèse sonore Faust.

Participants :
- Jean Bresson (IRCAM)
- Dimitri Bouche (Stage ENSEA / IRCAM)
- Stéphane Letz (Grame)

1 Introduction

OpenMusic est un langage de programmation visuel basé sur Common Lisp et dédié à la composition musicale assistée par ordinateur, utilisé par les compositeurs pour modéliser, transformer ou produire des données et structures musicales [1, 6]. Cet environnement a été utilisé par les compositeurs de musique contemporaine depuis une quinzaine d'années, et est aujourd'hui considéré comme l'une des principales références dans le domaine de la composition assistée par ordinateur. La composition

assistée par ordinateur (ou CAO) se concentre sur le développement de modèles correspondant à des processus compositionnels souvent complexes, intégrant programmes, structures de données, notation et autres représentations graphiques dans la conception de processus génératifs ou transformationnels liées à la production musicale. Cette approche met en avant une orientation fonctionnelle symbolique de "haut niveau" dans la spécification et le traitement des données harmoniques, temporelles, rythmiques ou des autres aspects entrant en jeu dans les processus compositionnels.
Plus récemment, le domaine de la composition assistée par ordinateur a été étendu à la conception et au contrôle des processus plus "bas niveau" liés à la synthèse, le traitement ou la spatialisation des signaux sonores [5, 4]. Ainsi la production et le traitement sonore se trouvent directement connectés aux processus formels relevant du champ compositionnel. Pour cela, OpenMusic interagit avec des outils de traitement et de synthèse sonore, généralement par l'intermédiaire d'appels systèmes via ligne de commande ou par communication de type "client-serveur", pour lesquels des fichiers ou commandes de contrôle sont générés et transmis au sein des processus musicaux de plus haut niveau [3].
L'architecture audio de OpenMusic est basée sur la bibliothèque LibAudioStream (LAS) de Grame. Celle-ci permet la manipulation de fichiers et ressources audio via le concept de *stream*. Un *stream* dans LAS est un pointeur représentant représentant un flux d'échantillons audio et pouvant être transformé ou combiné à d'autres flux dans un graphe audio fonctionnel. Différents opérateurs disponibles dans la bibliothèque (*MakeReadSound*, *MakeCutSound*, *MakeMixSound*, *MakeSeqSound*...) permettent de produire des *streams* à partir des ressources du système (fichiers, entrées temps réel) ou par combinaison et transformation d'autres *streams*. Ces opérateurs permettent le montage de pistes sonores par la création des graphes de traitement et mixage, évalués et restitués par un système de *player* multi-pistes connecté à différents pilotes et architectures audio bas niveau (CoreAudio, PortAudio, Jack). Le moteur de rendu audio dans OpenMusic se base sur ce système pour restituer séquences et autres ressources sonores, produisant de manière "statique" des *streams* initialisant les différentes pistes du *player* LibAudioStream.
Le défi principal posé dans le projet INEDIT concerne l'interopérabilité et la formalisation des relations (notamment dans le domaine temporel) entre outils et environnements musicaux spécialisés. En particulier, la notion de GALS (Globally Asynchronous, Locally Synchronous) met en avant la conciliation de contraintes temporelles synchrones et asynchrones dans un contexte où coexistent plusieurs échelles de temps hétérogènes. Nous nous intéressons ici aux relations entre la formalisation compositionnelle et les interactions de haut niveau (dans OpenMusic), et le traitement synchrone des signaux et ressources audio (dans LibAudioStream et le langage Faust). Dans ce document, nous décrivons la conception d'un nouveau moteur audio implémenté dans OpenMusic, utilisant les récents développements de LibAudioStream pour offrir un contrôle optimisé et flexible des ressources audio sur un système multi-pistes autonome [2]. La nouvelle architecture intègre une connexion directe avec le langage Faust (édition, compilation des programmes) et un contrôle dynamique et interactif des modules de traitement et synthèse sonore produits avec celui-ci.

2 Architecture audio

OpenMusic utilise des moteurs de rendu et d'ordonnancement externes pour restituer les structures musicales (en particulier les partions, généralement converties séquences MIDI) ou les fichiers audio produits ou manipulés dans l'environnement. Dans le cas des séquences MIDI, la bibliothèque MidiShare [10] prend en charge cette fonctionnalité de *player*. Pour l'audio, le player LibAudioStream est sollicité et effectue le rendu des *streams* assignés à ses différentes pistes.

2.1 Architecture "statique"

Qu'il s'agisse de MIDI ou d'audio, le rendu d'un ensemble de données musicales dans l'architecture actuelle de OpenMusic (v. $\leq$ 6.6) donne lieu à la création préalable d'une séquence distribuée sur plusieurs canaux (dans le cas de l'audio, d'un ensemble de *streams* résultant du mélange et de l'arrangement dans le temps des différentes sources) transmise au moteur de rendu qui prend alors la main sur la suite du processus.
Dans ce mode de fonctionnement l'environnement (et l'utilisateur) "perd la main" dès lors qu'il sollicite le player externe : si une ressource est en cours lecture, le statut du player est "occupé" et il est nécessaire de repasser dans un état libre (en arrêtant la lecture) afin de relancer une lecture après re-initialisation des différentes pistes (OM dispose donc d'une barre de transport – ou "palette" – unique, empêchant des appuis multiples sur les boutons de lecture).
Le lancement de la lecture constitue donc ici un goulot d'étranglement dans la connexion entre haut-niveau (spécification musicale) et bas niveau (rendu sonore). La révision de cette architecture permettra une plus grande perméabilité entre ces deux niveaux, c'est-à-dire entre les domaines temporels asynchrone et synchrone qui coexistent dans la chaine de traitement de l'information musicale.

2.2 Nouvelle architecture "double-player"

Le nouveau *player* audio que nous avons réalisé (qui sera intégré dans la version OM 6.7 distribuée aux utilisateurs en novembre 2013) propose un modèle plus flexible, permettant à la fois (i) de démarrer la lecture d'une source audio indépendamment des lectures en cours, et (ii) un affranchissement optionnel de la notion de "piste", nécessaire au bon fonctionnement de (i).
La condition (i) visée ci-dessus voudrait permettre de lancer la lecture d'un *stream* audio à n'importe quel moment, y compris en cours de lecture d'un ou plusieurs autres *streams*. Le player sera donc lancé en début de session et tournera en permanence en tâche de fond, intégrant les *streams* à la volée sur ses différentes pistes.
L'architecture LibAudioStream, dans sa fonctionnalité de player, abstrait l'idée d'un mixeur et impose l'assignation des ressources audio par piste. La piste (*track*) est en effet un attribut des objets *sound* dans OpenMusic, que l'utilisateur peut spécifier afin de répartir les sources audio sur les différents canaux du player. Une seule source (ou *stream*) peut en réalité être assignée par piste : dans le mode actuel "statique" de l'architecture, la coexistence de sons sur une même piste est gérée en amont par la combinaison/mixage des différentes sources en un *stream* unique transféré au player. La condition (i) impose donc un certain nombre de contraintes sur la gestion des pistes : pour "lancer" un *stream* dans le flux de rendu du player, il faut s'assurer que celui-ci ne soit pas déjà occupé par un autre *stream*.
Deux modes sont donc disponibles à présent, correspondant à deux players coexistant dans la nouvelle architecture audio : un player "visible" accordant la gestion des pistes à l'utilisateur (si une piste est occupée, le son ne pourra pas être joué), et un player "caché" dont la gestion de l'assignation des pistes est automatisée. L'utilisateur peut ainsi choisir d'assigner une piste explicitement, ou d'utiliser une piste spéciale (0, par convention) et ainsi envoyer son fichier audio sur le player "caché" (voir figure 1). Le *smart system* visible sur la figure 1 effectue la sélection automatique des pistes sur le player "caché". Il consiste en un registre actualisé en permanence et rendant compte de l'état de chacune de ces pistes (voir figure 2). Ce système permet de répartir les nouvelles sources entrantes sur des pistes libres, à faible coût et de façon transparente pour l'utilisateur.
Les pistes du player "visible" sont en revanche contrôlable individuellement sur le modèle d'un système de mixage numérique classique. Une table de mixage virtuelle (*general mixer*, voir figure 3) permet une gestion par piste du volume, panoramique, ainsi que des effets et synthétiseurs créés avec Faust (voir partie suivante). Celle-ci offre également la possibilité de sauvegarder et re-charger différentes configurations (*settings*) du player.

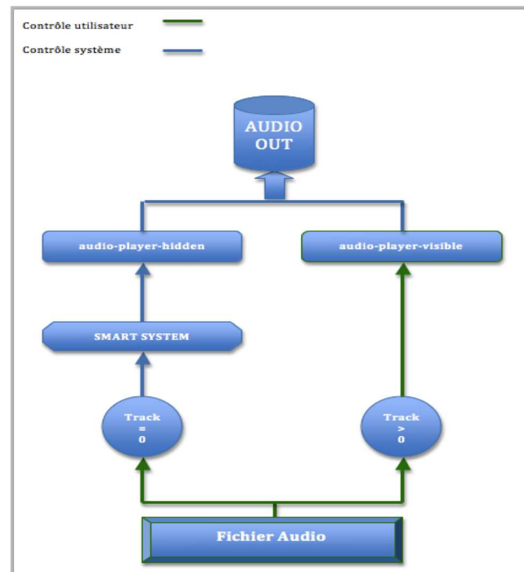

Figure 1 – Assignation des sources sur les players audio (extrait de [2]).

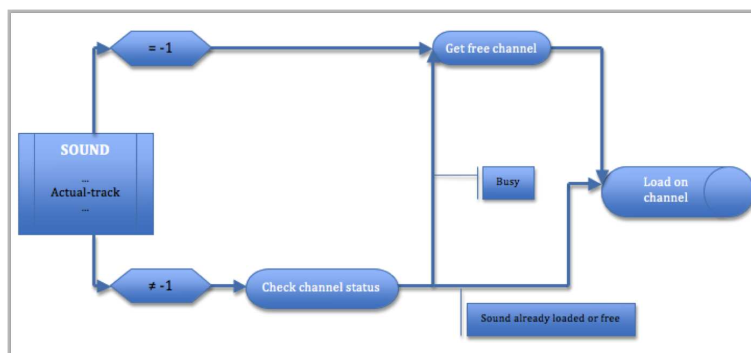

Figure 2 – Sélection automatique des pistes (“smart-system”) (extrait de [2]).

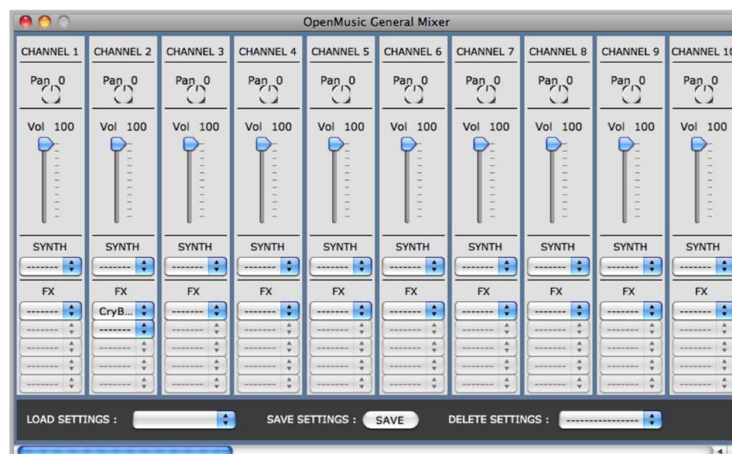

Figure 3 – *General mixer* : contrôle des paramètres du player audio.

3 Intégration de Faust

Faust [9] 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. Le compilateur Faust analyse et traduit le code en C++ optimisé et l'intègre dans une "architecture" cible (Max, PureData, VST etc...) où il sera "enveloppé" d'une interface de contrôle. Faust permet donc à l'utilisateur de créer ses propres effets et synthétiseurs, puis de les appliquer et contrôler dans l'environnement de son choix. De plus, un grand nombre de bibliothèques sont fournies et peuvent être chargées/utilisées dans un programme Faust. Celles-ci implémentent la plupart des fonctionnalités basique de synthèse et traitement audio (filtres, réverbération, synthétiseurs élémentaires...) Faust permet également une visualisation sous fore de "bloc diagram" du processus créé, en produisant un fichier au format SVG. Ce format permet un affichage interactif à plusieurs niveaux, et l'exploration de la structure hiérarchique des blocs fonctionnels à l'aide d'un navigateur web.
Les programmes Faust peuvent être déployés sur un grand nombre de plateformes et formats, et s'articulent notamment avec LibAudioStream. Tout comme le *stream*, un *effect* est un pointeur abstrait représentant un effet Faust compilé manipulé par les fonction de la bibliothèque, qui peuvent modifier son état (valeurs de ses différents paramètres) ou l'appliquer comme traitement sur un *stream* audio. LibAudioStream embarque par ailleurs la technologie LLVM [8, 7], qui lui permet la compilation de code Faust à la volée, et la donc création dynamique d'effets immédiatement utilisables dans le système audio.
Une nouvelle bibliothèque OpenMusic, OM-Faust, porte au niveau utilisateur des fonctionnalités avancées pour l'intégration des effets et synthétiseurs Faust dans l'architecture audio OpenMusic.

3.1 Effets et synthétiseurs

Deux objets ont été créés dans OM-Faust, permettant de produire des effets (*Faust-Effect*) ou des synthétiseurs (*Faust-Synth*) avec le langage Faust. Pour rappel, les objets OpenMusic se comportent comme des fonctions. Les boites dans un programme visuel comportent toutes des entrées (arguments) et sorties (valeurs produites).

Les arguments du constructeur *Faust-Effect* sont :

- Un objet *Textfile*, l'éditeur de texte de OpenMusic. C'est dans cet éditeur que l'utilisateur écrira et éditera le 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 audio.

L'évaluation de cette boîte dans OpenMusic produit un certain nombre d'actions :

1. Compilation du code par appel à LibAudioStream (*MakeFaustAudioEffect*) – ou renvoi d'une erreur si celle-ci échoue (*GetLastLibError*) ;
2. Enregistrement de l'effet compilé (pointeur LAS) dans un registre, associé avec le nom choisi (ou un nom assigné par défaut) ;
3. Connexion sur une piste du player audio si celle-ci est spécifiée ;
4. Génération d'une interface graphique pour le contrôle de l'effet.

La dernière étape (génération d'une interface graphique) est fondamentale dans l'intégration du contrôle interactif des effets créés avec Faust. LibAudioStream permet de produire un fichier au format

JSON spécifiant la structure d'une interface de contrôle pour un effet Faust donné (*GetJsonEffect*). La bibliothèque Common Lisp Yason[1] est ensuite utilisée comme "parser" afin d'obtenir une représentation du contenu d'un fichier JSON sous forme de liste. Cette liste est utilisée pour recréer la structure arborescente des contrôles et l'interface utilisateur correspondante, qui sera attachée à la boite *Faust-Effect* dans un programme visuel OpenMusic (voir figure 4).

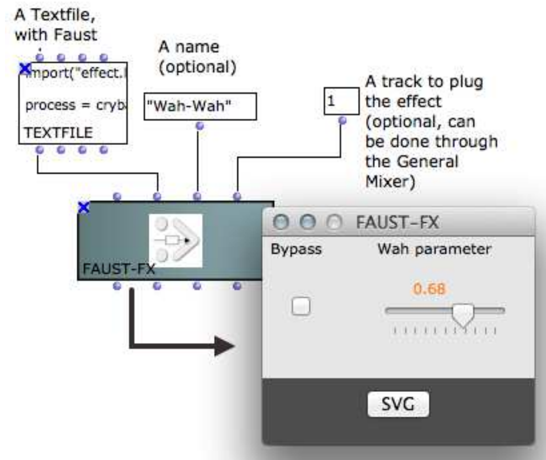

FIGURE 4 – Création d'un *Faust-Effect* dans OpenMusic et visualisation de l'interface utilisateur. Sur la fenêtre de contrôle, le bouton *SVG* permet d'ouvrir le fichier SVG produit lors de la compilation dans un navigateur web.

Le fonctionnement des synthétiseurs Faust est similaire à celui des effets. L'édition, la compilation et la création de l'interface graphique s'effectuent de la même manière avec un objet *Faust-Synth*, à première vue très semblable à *Faust-Effect*.
Cependant, la fonction d'un synthétiseur (production de son) diffère de celle d'un effet : un effet transforme un son (ou un *stream* dans LAS), et peut donc s'appliquer en cascade. En d'autre termes, on peut par exemple assigner plusieurs effets à une même piste audio, alors qu'un synthétiseur est considéré comme une source audio à part entière : il occupera une piste audio (qui ne sera donc plus disponible pour les autres ressources) et pourra lui aussi subir des transformations par application d'effets.
La figure 3 montre la possibilité d'assigner un synthétiseur, et jusqu'à 5 effets par piste sur la console de mixage. Les effets et synthétiseurs disponibles dans les menus déroulants de la fenêtre *general mixer* correspondent aux effets Faust compilés par des objets *Faust-Effect* ou *Faust-Synth* dans l'environnement OpenMusic, désignés par leur nom. Ces listes sont donc tenues à jour au fur et à mesure de l'utilisation du logiciel.
Un synthétiseur (*Faust-Synth*) peut également être utilisé dans le player "caché" de OpenMusic, sans spécification d'une piste particulière. Dans ce cas il sera considéré comme une source audio standard dans la recherche et l'allocation d'une piste disponible. Afin de déterminer des bornes temporelles pour la l'exécution (ou "lecture") d'un synthétiseur (en particulier dans un mode de "pré-écoute", mais également pour permettre de libérer les pistes du player "caché" au terme de cette exécution) on assignera également une durée à l'objet *Faust-Synth*, qui constitue un argument supplémentaire du constructeur.

1. http ://common-lisp.net/project/yason/

3.2 Automations

Si les interfaces utilisateur créées grâce aux spécifications du fichier Json produit par Faust/LAS nous permettent une interaction directe sur les paramètres des traitements et synthétiseurs Faust créés dans notre environnement, l'*écriture* du contrôle de ces traitements et synthétiseurs est un aspect complémentaire fondamental dans le cadre d'un système de composition assistée par ordinateur.
L'objet *Faust-Automation* représente une courbe de contrôle pour les synthétiseurs et effets créés dans OM-Faust. Cet objet est une extension de la classe BPF (*break-point function*, ou fonction par points) à laquelle est ajouté un pointeur double identifiant un effet (ou synthétiseur) et le nom d'un de ses paramètres. Les objets *Faust-Automation* peuvent ainsi être créés par un processus algorithmique quelconque développé dans OpenMusic, et/ou produits/modifiés à la main grâce à l'interface d'édition de BPF (voir figure 5). Ils ont la propriété d'être "executables" sur un player interne, où chaque point de la courbe sera considéré comme une instruction, à une date donnée, de changement de valeur pour le paramètre ciblé de l'effet ou du synthétiseur.

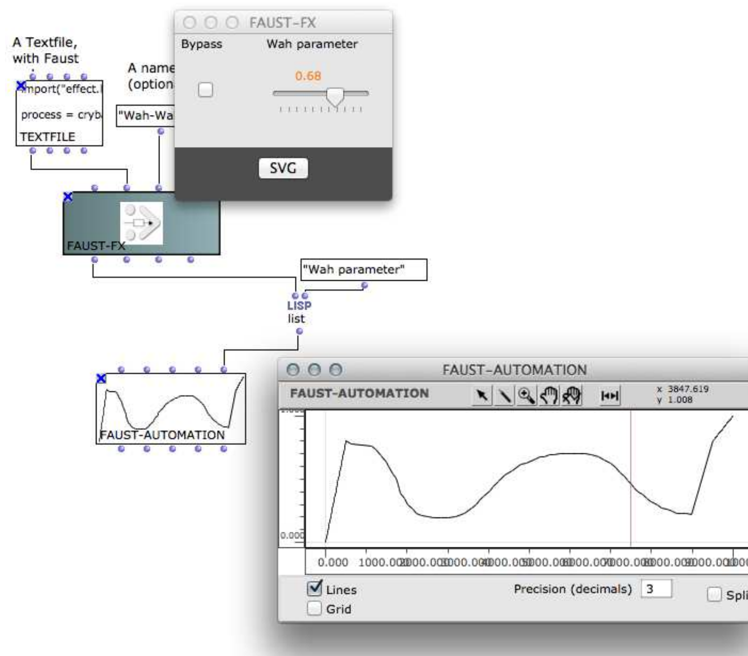

FIGURE 5 – Utilisation de *Faust-Automation* pour le contrôle d'un traitement *Faust-Effect*.

4 Conslusion

La compilation Faust à la volée, permise par LAS et LLVM, et les modalités duales de contrôle temps réel (via l'interface d'effet) vs. contrôle "écrit" (avec les courbes d'automation), centralisée par le système de player de cette nouvelle architecture, offre des possibilités de contrôle inédits pour la synthèse et le traitement audio dans OpenMusic, intégrant programmation "musicale" (OM) et signal (Faust) dans un workflow flexible et efficace.
Des perspectives intéressantes seront à explorer concernant l'utilisation de ce système avec des entrées audio (et non plus seulement des sources statiques), et la mesure dans laquelle un environnement compositionnel a priori "hors-temps" comme OpenMusic peut interférer dans une chaîne d'interaction audio temps réel.

References

[1] G. Assayag, C. Rueda, M. Laurson, C. Agon and O. Delerue : Computer Assisted Composition at IRCAM : From PatchWork to OpenMusic. *Computer Music Journal*, 23(3), 1999.

[2] D. Bouche : *Dynamisation de l'architecture audio du logiciel de composition assistée par ordinateur OpenMusic*. Rapport de projet de fin d'études, ENSEA / IRCAM, 2013.

[3] J. Bresson : Sound Processing in OpenMusic. *Proceedings of the International Conference on Digital Audio Effects - DAFx-06*, Montréal, Canada, 2006.

[4] J. Bresson : Spatial Structures Programming for Music. *Spatial Computing Workshop - SCW'12*, (co-located w. the 11th Int. Conf. on Autonomous Agents and MultiAgent Systems - AAMAS'2012), Valencia, Spain, 2012.

[5] J. Bresson and C. Agon : Musical Representation of Sound in Computer-Aided Composition : A Visual Programming Framework. *Journal of New Music Research*, 36(4), 2007.

[6] J. Bresson, C. Agon and G. Assayag : OpenMusic Visual Programming Environment for Music Composition, Analysis and Research. *ACM Multimedia*, Scottsdale,USA, 2011.

[7] A. Gräf : Functional Signal Processing with Pure and Faust using the LLVM Toolkit. *Proceedings of the Sound and Music Computing Conference*, Padova, Italie, 2011.

[8] C. Lattner : *LLVM : An Infrastructure for Multi-Stage Optimization*. Masters Thesis, Computer Science Dept., University of Illinois at Urbana-Champaign,USA, 2002.

[9] Y. Orlarey, D. Fober and S. Letz : Faust : an Efficient Functional Approach to DSP Programming. *In* G. Assayag and A. Gerzso (Eds.) *New Computational Paradigms for Computer Music*, Delatour France / Ircam, 2009.

[10] Y. Orlarey and H. Lequay : MidiShare : A Real Time multi-tasks software module for Midi applications. *Proceedings of the International Computer Music Conference*, Ohio State University, USA, 1989.