# Actes des Journées d'Informatique Musicale (JIM 2018), Amiens

Louis Bigo, Mathieu Giraud, Richard Groult, Florence Levé

JIM 2018

# Journées d'Informatique Musicale

Amiens, 16 – 18 mai 2018

www.algomus.fr/jim2018

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

mis

# Soutiens et sponsors










Les Journées d'Informatique Musicale 2018, pilotées par l'Association Francophone d'Informatique Musicale (AFIM) ont lieu du 16 au 18 mai 2018 à Amiens, au Logis du Roy, monument du XVI$^{e}$ siècle au cœur du centre historique d'Amiens. Après Bourges (2014), Montréal (2015), Albi (2016) et Paris (2017), les JIM 2018 sont organisées par l'équipe Algomus (CRIStAL, UMR 9189 CNRS, Université de Lille et MIS, Université de Picardie Jules Verne, Amiens, `www.algomus.fr`).

Le comité de programme a reçu 22 soumissions impliquant au total 31 auteurs. Chaque soumission a été relue et évaluée par au moins 3 membres de la communauté. Au final 13 articles ont été retenus pour une présentation longue et 4 pour une présentation sous forme de poster. Les actes contiennent ainsi des articles longs et des résumés, ainsi qu'une présentation du nouveau groupe de travail de l'AFIM et le résumé des trois présentations invitées.

Le thème choisi pour les JIM 2018, *informatique musicale et pédagogie*, concerne à la fois l'utilisation de l'outil informatique dans l'apprentissage et la pratique de la musique, mais aussi la pédagogie de l'informatique musicale en tant que discipline. Cinq présentations sont relatives à ce thème, et une table ronde regroupe des enseignants en collège, en lycée et à l'université.

Les journées comportent également une *table ronde entreprises* réunit des professionnels de la région Hauts-de-France développant des services et des produits liés à la musique et aux nouvelles technologies : Aodyo (contrôleur MIDI *Sylphyo*), Arobas Music (logiciel d'édition de partitions *Guitar Pro*) et Recisio (contenus de karaoké, *Jamzone*).

Au programme des JIM figurent enfin des concerts et spectacles (*Solo* de Peter Orins, *Fourier Revient* de Phusis, et un récital de chant lyrique de Bénédicte Hilbert et Anne-Lise Gilet organisé par le CERCLL), la présentation de l'installation *Algo-rhythme* de Val Kiri, et une visite musicale de la cathédrale d'Amiens.

Merci à l'ensemble des soutiens, à l'AFIM, aux comités de programme et d'organisation, aux auteurs et aux participants !

Louis Bigo Mathieu Giraud Richard Groult Florence Levé

## Comité de programme

Florent Berthaut
Nancy Bertin
Louis Bigo
Bruno Bossis
Muriel Boulan
Jean Bresson
Marc Chemillier
Jean-Marc Chouvel
Julien Debove
Myriam Desainte-Catherine
Dominique Fober
Jean-Louis Giavitto
Mathieu Giraud *(co-président)*
Richard Groult
Nathalie Hérold
Florent Jacquemard
David Janin
Florence Levé *(co-présidente)*
Mikhail Malt
Julien Rabin
Marc Rigaudière
Philippe Rigaux
Stephan Schaub
Anna Shvets
Alice Tacaille
Vincent Tiffon
Charlotte Truchet
Stéphanie Weisser

## Comité d'organisation

Florent Berthaut
Louis Bigo *(co-président)*
Romain Bricout
David Durand
Laurent Feisthauer
Mathieu Giraud
Richard Groult *(co-président)*
Nicolas Guiomard-Kagan
Emmanuel Leguy
Florence Levé

## Comité de pilotage AFIM

Daniel Arfib
Gérard Assayag
Marc Chemillier
Myriam Desainte-Catherine
Dominique Fober
Mikhail Malt
Yann Orlarey
François Pachet
Laurent Pottier
Julien Rabin
Jean Michel Raczinski
Anne Sedes

# Table des matières

**Soutiens** **iii**

**Préface** **v**

**Comités** **vii**

**Conférenciers invités** **1**

Fourier en musique
*Jean-Paul Chehab* . . . . . 3

Approches sémiotiques et algorithmiques pour la modélisation de schémas structurels musicaux
*Frédéric Bimbot* . . . . . 5

Hello World, le making of
*François Pachet* . . . . . 7

**Association Française d'Informatique Musicale** **10**

Présentation du groupe de travail AFIM « Archivage collaboratif et préservation créative »
*Alain Bonardi, Serge Lemouton, Laurent Pottier, Jacques Warnier* . . . . . 11

**Articles** **13**

Camomile, enjeux et développements d'un plugiciel audio embarquant Pure Data
*Pierre Guillot* . . . . . 15

Ordonnancement adaptatif d'un graphe audio avec dégradation de qualité
*Pierre Donat-Bouillud* . . . . . 25

Synchronisation de données inter-processus dans les applications audio temps réel : qu'est-ce qui débloque ?
*Thibaut Carpentier* . . . . . 35

Une nouvelle implémentation du Spatialisateur dans Max
*Thibaut Carpentier* . . . . . 45

Étude et appropriation de l'augmentation instrumentale proposée par Jean-Claude Risset dans l'élaboration de ses duos pour un pianiste
*Sébastien Clara* . . . . . 53

Outils informatiques et analyse musicale : représenter le routing de Jupiter
*Maxence Larrieu* . . . . . 63

Inférence de segmentation structurelle par compression via des relations multi-échelles dans les séquences d'accords
*Corentin Guichaoua, Frédéric Bimbot* . . . . . 71

Exploration de dépendances structurelles mélodiques par réseaux de neurones récurrents
*Nathan Libermann, Frédéric Bimbot, Emmanuel Vincent* . . . . . 81

Évaluation de la correction rythmique des partitions numérisées
*Francesco Foscarin, Florent Jacquemard, Raphaël Fournier-S'Niehotta, Philippe Rigaux* . . . . . 87

The structural function of musical texture : Towards a computer-assisted analysis of orchestration
*Didier Guigue, Charles de Paiva Santana* . . . . . 97

Enseigner le patching de manière collective avec le logiciel collaboratif Kiwi (**Prix AFIM 2018**)
*Philippe Galleron, Eric Maestri, Jean Millot, Alain Bonardi, Eliott Paris* . . . . 105

Appropriating Music Computing Practices Through Human-AI Collaboration
*Hugo Scurto, Frédéric Bevilacqua* . . . . . 115

RéDi-Musix : réseau pour la distribution de la musique mixte
*Alexander Mihalic, Mikhail Malt* . . . . . 121

**Posters** **129**

Music from Aura
*Pierre-Henri Vulliard* . . . . . 131

La théorie de la gravitation tonale
*Emanuele Di Mauro* . . . . . 135

Polyphonic Singing presented as a Multiplayer Classroom Online Game
*Jonathan Bell* . . . . . 139

Learning Geometry and Music through Computer-aided Music Analysis and Composition : A Pedagogical Approach
*Sonia Cannas* . . . . . 143

**Index des auteurs** **147**

À l'issue des JIM, Philippe Galleron et Éric Maestri ont été désignés lauréats du prix *jeune chercheur* AFIM 2018. Le public tout comme le jury AFIM a particulièrement apprécié leur étude en situation pédagogique de Kiwi sur le terrain... qui était de plus parfaitement adaptée au thème de cette année.

# Conférences invitées

# FOURIER EN MUSIQUE

*Jean-Paul Chehab*
LAMFA, Amiens
Université de Picardie Jules Verne

Le travail le plus célèbre de Joseph Fourier (1768-1830) porte sur la théorie analytique de la chaleur (1807) pour laquelle il a introduit une technique de résolution extrêmement novatrice, dont les développements et les applications ont bouleversé tous les domaines scientifiques. Si la musique n'était pas initialement considérée, l'analyse de Fourier se prête particulièrement bien à la nature vibratoire du son. Après avoir retracé la biographie du savant et replacé son travail fondateur dans son contexte, nous décrirons les phénomènes sonores, d'abord du point de vue physique à travers la notion de modes propres des instruments (corde, timbale, guitare) puis de celui du traitement du signal, s'agissant de leurs représentations fréquentielles (spectrogramme). Cet exposé grand public s'inscrit dans le cadre de la célébration du 250$^{e}$ anniversaire de la naissance de Joseph Fourier.

# APPROCHES SÉMIOTIQUES ET ALGORITHMIQUES POUR LA MODÉLISATION DE SCHÉMAS STRUCTURELS MUSICAUX

*Frédéric Bimbot*
IRISA, CNRS

Modéliser la structure musicale à travers différents styles est un enjeu important de la recherche en informatique musicale. Cet exposé présentera l'approche "Système & Contraste" qui permet de décrire avec une grande généralité l'organisation multi-échelle des éléments au sein des segments structurels musicaux.

## HELLO WORLD, LE MAKING OF

*François Pachet*
Spotify Paris

L'album *Hello world*, sorti en janvier 2018, est le premier à avoir utilisé l'intelligence artificielle de manière systématique pour la composition et l'orchestration de titres de musique *mainstream*, dans des styles divers. Cet exposé présentera la conception de cet album, en détaillant les enjeux techniques portant sur la génération (partitions et audio) tout comme les enjeux musicaux.

# PRESENTATION DU GROUPE DE TRAVAIL AFIM « ARCHIVAGE COLLABORATIF ET PRESERVATION CREATIVE »

*Alain Bonardi*
CICM-MUSIDANSE
Université Paris 8
alain.bonardi@gmail.com

*Serge Lemouton*
IRCAM
lemouton@ircam.fr

*Laurent Pottier*
CIEREC Université
Jean-Monnet
laurent.pottier@univ-st-etienne.fr

*Jacques Warnier*
CNSMDP
JWarnier@cnsmdp.fr

**RÉSUMÉ**

Le groupe de travail AFIM « Archivage collaboratif et préservation créative » a été lancé mi-décembre 2017. Ce texte en présente les objectifs initiaux, les tout premiers travaux et les perspectives de démarrage. A l'occasion des Journées d'Informatique Musicale 2018, il s'agit essentiellement d'engager le dialogue avec les centres de créations, les RIMs et toutes les personnes intéressées par ces thématiques.

## 1. PRÉSENTATION DES OBJECTIFS INTIAUX

Partant du constat de la volatilité des musiques informatiques et mixtes et des difficultés éprouvées à les rejouer, les réinterpréter ou les restaurer, l'objectif de ce groupe lancé mi-décembre 2017 est de travailler sur les modalités d'écriture des œuvres de ce répertoire permettant de les diffuser, de les enseigner et de les préserver.

A l'IRCAM, depuis une vingtaine d'années, ont été développés des outils (sous la forme d'une base de données accessible en ligne[1]) et des pratiques (principalement celle des réalisateurs en informatique musicale) destinées à préserver le répertoire propre de l'institut. D'autres studios ont également développé de telles pratiques. Nous pensons qu'il est temps de réfléchir à un modèle commun d'archives partagées.

Nous nous situons dans la continuité du groupe sur les nouveaux espaces de la notation musicale [3] puisque l'archive, au même titre que la partition musicale, est le lieu qui permet à la fois de collecter, de conserver et de consulter les intentions musicales du compositeur dans le but de pouvoir les ré-interpréter et les restituer aux auditeurs.

Il faut insister sur la nature en perpétuelle évolution du contenu de ces archives. A chaque remédiation de l'œuvre, pour des raisons techniques (mises à jour de logiciels, obsolescence des matériels) ou esthétiques, des modifications, des mises à jour sont effectuées sur les programmes, sur les patchs, sur les sons. Cela implique que les systèmes destinés à préserver les œuvres doivent à la fois avoir un contenu dynamique pour que les contenus puissent être facilement mis à jour. Ils doivent également permettre une traçabilité de l'évolution des différentes versions.

Un autre trait caractéristique est la multiplicité des acteurs qui interviennent sur ces archives : compositeurs, interprètes (instrumentistes et réalisateurs en informatique musicale), ingénieurs du son, documentalistes, musicologues, étudiants, éditeurs, producteurs. Sur une même œuvre, voire sur une même version, l'archive doit permettre à ces différents profils de collaborer.

La question des droits légaux et des licences d'utilisation dans ce contexte fortement collaboratif devra également être abordée.

L'objectif sera de faire le point sur les différents acteurs nationaux et internationaux proposant des démarches et des outils visant à la conservation des œuvres numériques, afin d'évaluer les dispositifs les plus appropriés pour traiter du cas des musiques informatiques.

Un des objectifs principaux de notre groupe sera ensuite d'établir un modèle opérationnel et fonctionnel d'archive ouverte participative permettant la diffusion et l'inscription dans la durée des œuvres de musique informatique. Dans le cadre de ce groupe de travail, on envisage de développer (au cours de la seconde année du projet) un site prototype répondant à ce modèle et mis en ligne, permettant aux partenaires du projet de mettre en commun leurs archives.

## 2. TRAVAUX PRÉVUS DU GROUPE EN 2018

Dans un premier temps, le groupe de travail se consacre à un état de l'art des pratiques de documentation des œuvres musicales avec technologie. Pour commencer, nous avons mené une enquête auprès des Centres Nationaux de Création Musicale (CNCM) : GRAME à Lyon, GMEM à Marseille, CIRM à Nice, Césaré à Reims, La Muse en circuit à Paris-Alfortville ainsi qu'auprès du SCRIME à Bordeaux. Les résultats montrent qu'il n'y a pas d'action concertée ni de pratique rigoureuse pour la préservation à long terme ou la documentation de la musique utilisant l'informatique temps réel. Les résidences de compositeurs dans ces centres sont souvent de courte durée (quelques semaines), ce qui ne laisse en général pas de temps pour la documentation. D'une manière générale, les fichiers sont correctement sauvés, les

[1] http://brahms.ircam.fr/sidney/

Alain Bonardi, Serge Lemouton, Laurent Pottier, Jacques Warnier, *Présentation du groupe de travail AFIM « Archivage collaboratif et préservation créative »*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 11–12, 2018. 

œuvres sont enregistrées et parfois filmées, les parties électroniques sont parfois enregistrées comme fichiers audio séparés. La tendance pour les œuvres temps réel est d'éviter des outils logiciels commerciaux fermés et de se limiter aux objets standard dans le cas de patchs Max. Les sauvegardes concernent trois aspects de l'œuvre : l'architecture des traitements DSP [4], la gestion des événements (déclenchements, presets, suivi de partition) et les interfaces de contrôle gestuel.

Cet état de l'art passe également par la consultation des résultats obtenus lors de projets antérieurs de recherche en documentation des œuvres musicales avec technologie, par exemple MUSTICA (2003-2006) [1] ou ASTRÉE (2009-2011) [2].

Une des étapes importantes de l'année est l'analyse des pratiques actuelles à l'IRCAM autour de la base de documentation des œuvres Sidney [5], qui servira de base à notre réflexion et nos propositions à venir, à partir d'un ensemble de « bonnes pratiques » que nous aurons identifiées.

## 3. REFERENCES


[1] Bachimont, B., et al. (2003). Preserving Interactive Digital Music: A Report on the MUSTICA Research Initiative, Proceedings of the Third International Conference on WEB Delivering of Music (WEB'03), Leeds, England.

[2] Bonardi, A. (2013). Pérenniser pour transmettre, transmettre pour pérenniser - Destins de l'œuvre mixte interactive - Autour de En Echo, pièce de Philippe Manoury, Musique et Technologie - Préserver, archiver, re-produire, Portraits polychromes, hors-série thématique, Paris, Institut National de l'audiovisuel, p.105-126.

[3] Fober, D., Bresson, J., Couprie, P., Geslin, Y. (2015). Les nouveaux espaces de la notation musicale : Groupe de travail AFIM, Actes des Journées d'Informatique Musicale, 2015, Montréal, Canada.

[4] Pottier, L. (2013). La régénération des sons de Turenas de John Chowning, Préserver - Archiver - Re-produire : musique et technologie, jeux vidéo, dir. Evelyne Gayou, Portraits polychromes, hors-série thématique n°21, Paris, INA-GRM, p.145-196.

[5] Lemouton, S. & Goldszmidt, S. (2016). La préservation des œuvres du répertoire IRCAM : Présentation du modèle Sidney et analyse des dispositifs temps réel, Actes des Journées d'Informatique musicale, Albi, GMEA, 2016.

# Articles

# CAMOMILE, ENJEUX ET DEVELOPPEMENTS D'UN PLUGICIEL AUDIO EMBARQUANT PURE DATA

*Pierre Guillot*
CICM – EA1572
Université Paris 8
guillotpierre6@gmail.com

**RÉSUMÉ**

Cet article présente Camomile un outil plugiciel audio multiplateforme, compatible avec de multiples formats et permettant de charger des patchs Pure Data dans les stations de travail audionumérique. Cette présentation reviendra tout d'abord sur les contextes d'utilisation dans lesquels la nécessité d'un tel outil est apparue. L'objectif sera de mettre en avant les principaux enjeux du projet sur un plan fonctionnel, usuel et technique. Par la suite, un état de l'art, général mais non exhaustif, sera présenté en revenant sur les différentes propositions et les évolutions dans ce domaine. Parcourir ces travaux, des premières approches des logiciels de type *patcher* embarqués jusqu'aux projets plus récents de plugiciels audio, permettra notamment de clarifier les difficultés, les contraintes et les questions soulevées par cette démarche afin de comprendre les choix opérés lors de la mise en œuvre du projet Camomile. Des solutions seront alors apportées en revenant sur les principaux axes du développement de l'outil. Il s'agira de présenter les choix importants opérés au cours des différentes versions et les répercussions sur l'outil final et son usage. Enfin, un bilan et les perspectives de ce projet seront exposés.

## 1. INTRODUCTION

Camomile[1] est un outil permettant de créer des plugiciels audionumériques – des modules externes destinés à être utilisés dans les stations de travail audionumériques – à partir de patchs Pure Data [1][2]. Cet outil est lui-même un plugiciel aux formats VST2, VST3[3] et Audio Unit[4] pour les systèmes d'exploitation Linux, Windows et MacOs embarquant le moteur de Pure Data. Après avoir créé un traitement ou un synthétiseur sonore dans Pure Data, l'utilisateur peut utiliser les plugiciels de Camomile comme base, et leurs associer les patchs élaborés afin de concevoir des plugiciels autonomes (Figure 1).

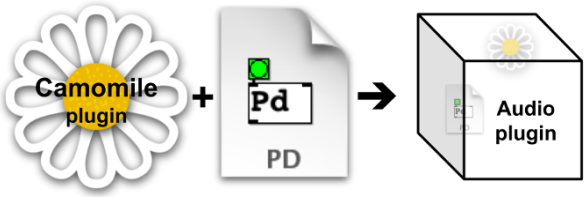



**Figure 1.** Fonctionnement de la création d'un nouveau plugiciel à partir d'un patch Pure Data et d'un plugiciel Camomile.

Cette approche offre donc à la fois les avantages du plugiciel, notamment via la variété des stations de travail audionumérique à laquelle elle donne accès, ainsi que ceux de Pure Data, à savoir son dynamisme et sa modularité. De la part de l'utilisateur, il s'agira de comprendre comment créer des patchs afin de tirer profit de toutes les fonctionnalités offertes par les plugiciels et les stations de travail. Ce sujet est présenté dans la documentation fournie avec l'outil Camomile et sera traité dans une prochaine publication. Car, avant cela, il semble intéressant de revenir sur les raisons de la mise en œuvre de cet outil ainsi que sur les choix opérés lors de celle-ci. En effet, c'est de cela même que résultent les modalités d'utilisation de l'outil. Cet article revient donc dans la section 2, sur le contexte et les enjeux de cette mise en œuvre, afin de présenter les spécifications techniques fonctionnelles qui ont permis de déterminer les choix de développement. Puis, dans la section 3, les travaux à la fois préexistants et parallèles au projet Camomile seront présentés, afin de mettre en avant les questions et les problématiques importantes qu'il est nécessaire de résoudre. Suite à cela, dans la section 4, les axes importants de la mise en œuvre de Camomile seront explorés, en allant des premiers développements en mai 2015 à la dernière version publiée en décembre 2017 (Figure 2). Enfin dans la section 5, ce travail sera mis en

[1]Les différentes versions du plugiciel sont disponibles en ligne sur le répertoire Github du projet. Le code source est ouvert, libre et gratuit et est disponible sur ce même répertoire. Depuis la version 0.1.0, les sources sont distribuées sous licence GNU GPLv3. Les sources des versions antérieures sont distribuées sous licence BSD 3 github.com/pierreguillot/camomile (site consulté en janvier 2018).

[2]Pure Data est un logiciel de type *patcher* libre et gratuit, créé par Miller Puckette à l'Université de San Diego en Californie msp.ucsd.edu/software.html (site consulté en janvier 2018).

[3]Les formats de plugiciel audionumérique VST (Virtual Studio Technology) 2 et 3 sont développés par la société Steinberg www.steinberg.net (page consultée en janvier 2018).

[4]Le format de plugiciel audionumérique Audio Unit est développé par Apple developer.apple.com/audio/ (page consulté en janvier 2018).

Pierre Guillot, *Camomile, enjeux et développements d'un plugiciel audio embarquant Pure Data*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 15–23, 2018. 

perspective en revenant sur les questions qu'il reste à explorer mais aussi sur le potentiel offert par un tel outil.

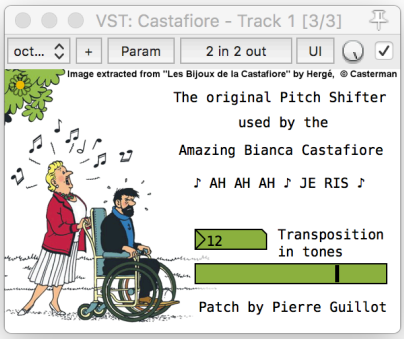



**Figure 2.** Le plugiciel *Castafiore* distribué en exemple avec la dernière version de Camomile et permettant de modifier la hauteur du signal sonore en fonction d'un paramètre de transposition contrôlable via l'interface graphique avec soit une boîte nombre, soit une glissière (*slider*) horizontale.

## 2. CONTEXTES ET ENJEUX

Le plugiciel Camomile a été élaboré afin de répondre à l'envie et la nécessité d'utiliser des patchs Pure Data au sein d'une station de travail audionumérique. Bien que relativement spécifique, ce besoin se manifeste dans des pratiques, des usages et des contextes variés, qui répondent à différents enjeux, desquels découlent des spécifications qui peuvent être complémentaires. Cette partie revient sur ces questions, afin de comprendre et mettre en perspective les travaux préexistants présentés par la suite, ainsi que les choix qui ont été opérés lors de la mise en œuvre de Camomile.

### 2.1. Contextes du projet

En 2015, au début du projet, l'envie d'utiliser des patchs dans une station de travail audionumérique provient du besoin de faciliter le contrôle des traitements audionumériques mis en œuvre dans un contexte de temps différé. L'objectif est notamment de tirer profit de la ligne temporelle généralement présente dans les stations de travail audionumérique[5]. De nombreux praticiens de l'informatique musicale ont d'ailleurs fait part de leurs attentes à ce sujet, ce qui amène notamment en 2013 l'équipe du CICM, dans le cadre des projets HOA [4], à concevoir un plugiciel VST offrant alors la possibilité de spatialiser des sources directionnelles en ambisonie [5][6]. Il a alors été envisagé de proposer d'autres traitements originaux de l'espace et du son – tels que ceux offerts par les versions pour les logiciels Max[7] et Pure Data de la bibliothèque HOA[8] – mais faute de temps et en raison de la complexité des opérations nécessaires à leurs mises en œuvre, cet objectif n'a jamais pu être réalisé. Suite au prolongement des expérimentations et à l'approfondissement des travaux de recherches sur la spatialisation du son, notamment via le logiciel Pure Data [7], la seule solution viable pour répondre à ce problème semble être un plugiciel flexible et dynamique permettant de charger des patchs. Cette approche permet d'assurer une bonne reproduction du rendu sonore des traitements réalisés sous forme de patch au sein des stations de travail audionumérique, tout en limitant le temps de développement. Les logiciels de type *patcher* sont idéals pour l'expérimentation mais prendre en compte et intégrer dans un plugiciel les nombreux et fréquents changements que cela implique est très chronophage. Ainsi, l'enjeu du projet est aussi d'offrir un outil plus adapté à un contexte de développement et recherche. Enfin, la nécessité d'un tel outil s'est fait ressentir dans un cadre pédagogique. Le département de Musique de l'Université Paris 8 propose un cours de 2ème année de Licence d'Introduction à la Programmation avec Max et Pure Data[9] où beaucoup d'étudiants découvrent pour la première fois les logiciels de type *patcher* [9]. Aussi, l'un des défis pédagogiques est de démontrer le potentiel de ces outils via, entre autres, des conditions et des contextes concrets d'utilisations. Les étudiants sont familiers des stations de travail audionumérique et les utilisent généralement déjà fréquemment. Ainsi, proposer un plugiciel permettant d'utiliser un traitement créé sous forme de patch directement dans un de ces logiciels, offre une réponse à ce problème[10]. Ce projet tente donc de répondre à des problématiques provenant d'une approche d'utilisateurs, de développeurs mais aussi d'enseignants.

### 2.2. Spécifications

Ces observations amènent à définir un certain nombre de spécifications fonctionnelles et techniques.

[5]Dans cet article, il ne s'agit pas d'approfondir la question de la ou des représentations du temps offertes de manière native au sein du logiciel Pure Data, d'autant qu'il est possible de combiner Pure Data à d'autres outils d'écriture temporelle tels que les logiciels Ossia développé au LaBRI [2] ou Iannix [3]. Il s'agit simplement de mettre en avant un besoin et une attente qui est plus liée à des questions d'accessibilité et de facilité de mise en œuvre et d'utilisation.

[6]Le plugiciel est à présent non fonctionnel mais il est cependant toujours disponible sur le site du projet HOA www.mshparisnord.fr/hoalibrary (site consulté en janvier 2018).

[7]Le logiciel Max [6], originellement développé à l'IRCAM par Miller Puckette est aujourd'hui développé et distribué par la société Cycling'74 cycling74.com (site consulté en janvier 2018).

[8]Ces opérations sont par exemple construites autour de la synthèse granulaire ou d'un filtre à réponse impulsionnelle infinie réalisés sous forme de patch dans les logiciels Max et Pure Data. La complexité est alors de recréer en C et C++ l'ensemble des traitements audionumériques des objets utilisés, ainsi que leurs réseaux de connexions mais aussi les interfaces graphiques utilisateurs, même basiques, permettant de contrôler ces traitements.

[9]Ce cours a été dispensé depuis de nombreuses années par Anne Sèdes, Alain Bonardi, puis par moi-même en 2016 et 2017 et à présent par E. Maestri qui intègre notamment le logiciel Kiwi [8] à l'apprentissage.

[10]Au cours de ces années d'enseignement, savoir s'ils pouvaient utiliser leurs patchs sur des stations de travail audionumérique était en effet une question récurrente des étudiants.

### *2.2.1. Intégration du moteur patcher*

Afin de répondre à l'ensemble de ces usages dans des contextes compositionnels, expérimentaux, pédagogiques ou de développements, l'idéal serait évidemment que l'outil fonctionne sur le plus grand nombre de plateformes logicielles – et donc le plus grand nombre de formats de plugiciel audionumérique – et de systèmes d'exploitation. L'objectif serait de concevoir un outil compatible avec les systèmes Linux, MacOs et Windows[11], sous forme de plugiciel intégrant les technologie VST, Audio Unit ou encore LV2[12]. Sur un aspect plus fonctionnel, il est primordial que le rendu sonore du patch chargé au sein de la station de travail audionumérique soit identique à celui offert par l'application de bureau originel qui a permis de le créer. Cette spécification amène à réutiliser le moteur sonore originel mais aussi implicitement à éviter les logiciels au code source fermé, qui empêchent donc toute réutilisation[13]. De manière générale, le plugiciel final devrait satisfaire le maximum de spécifications définies par les différents formats et les stations de travail audionumérique, et notamment la capacité de gérer plusieurs instances, mais aussi d'être utilisable dans un contexte de *multithreads*[14]. Le premier enjeu principal du développement est donc, au travers du plugiciel, de réaliser le lien entre la station de travail audionumérique et le moteur de type *patcher*, en faisant concorder leurs spécifications et leurs recommandations respectives.

### *2.2.2. Gestion du moteur patcher et du plugiciel*

Évidemment, afin d'être utilisable, l'outil doit aussi proposer à l'utilisateur un moyen de définir et contrôler l'ensemble des fonctionnalités génériques d'un plugiciel, telles que les paramètres, les préréglages, la latence, etc. L'une des spécifications à ce sujet serait que l'utilisateur souhaitant créer un patch destiné au plugiciel audio n'ait pas à devoir coder en dehors des langages initiaux de type *patcher*, ou du moins n'ait pas à procéder à des opérations qui demandent des notions complexes d'informatique, telles que la compilation d'un code. En effet, si l'outil final est destiné à des étudiants, ce type d'approche peut rapidement décourager son utilisation. Cette spécification liée aux questions d'accessibilité est aussi importante dans un contexte de développement, car elle assure la portabilité d'un même patch sur l'ensemble des plateformes logicielles et des systèmes d'exploitations supportés par le plugiciel. Enfin, les logiciels de type *patcher* étant des applications de programmation graphique, il serait aussi intéressant de pouvoir offrir la possibilité de créer l'interface graphique utilisateur du plugiciel, via l'interface originale du programme. Le deuxième enjeu principal est de créer un système commun de gestion et de communication entre le logiciel de type *patcher* et le plugiciel. Cela afin de pouvoir définir les propriétés du plugiciel à partir du *patcher* et d'adapter le fonctionnement des patchs en fonction des informations envoyées par le plugiciel.

## 3. ÉTAT DE L'ART

Avant de proposer une réponse à ces enjeux et ces spécifications en exposant les choix réalisés lors de la mise en œuvre du plugiciel Camomile, cette partie revient sur les propositions préexistantes ou développées parallèlement, afin de s'en inspirer et d'éviter les problèmes qu'ils ont pu rencontrer.

### 3.1. Des origines de Max à Max for Live

L'idée d'utiliser des patchs en tant que module d'extension au sein d'un système tiers n'est pas nouvelle. Dès 1988, l'un des objectifs du logiciel Max est notamment de pouvoir créer des synthétiseurs audionumériques sous forme de patch, pouvant être utilisés sous forme de carte de traitement du signal[15]. Cette proposition ne restera longtemps qu'à l'état d'idée. En effet, à ses débuts le logiciel était exclusivement orienté sur le traitement de messages et l'intégration du traitement du signal audionumérique dans Max n'apparaîtra qu'en 1991 [10]. C'est à la suite du développement de l'audionumérique avec l'émergence des stations de travail audionumérique telles que Cubase en 1996 du format VST[16], que l'idée pourra voir le jour. Notamment en remplaçant les cartes de traitement du signal par des plugiciels audionumériques. En 1998 la société Cycling'74 propose une extension à Max nommée Pluggo [11], permettant de charger des patchs au sein d'une station de travail audionumérique via un plugiciel VST. Cet outil offre un nombre de fonctionnalités très complètes, telles que la création du moteur sonore, la création d'une interface graphique originale et d'un système permettant de configurer et gérer les paramètres, notamment à l'aide d'objets Max dédiés spécifiquement à cet usage. Le plugiciel de Pluggo fonctionne avec une version d'exécution de Max embarquée[17] qui permet de charger dynamiquement des patchs. Cet outil offre aussi la possibilité de générer des

[11]Il est raisonnable d'affirmer que l'usage de cet outil sur d'autres systèmes d'exploitation ou avec d'autres formats de plugiciel audionumérique se révèle purement anecdotique pour le moment.

[12]Les informations relatives au format LV2 sont disponibles sur le site lv2plug.in (site consulté en janvier 2018).

[13]Cette spécification est d'autant plus en accord avec une approche de recherche et recoupe des questions pédagogiques car les applications libres et ouvertes sont aussi souvent gratuites ou du moins plus accessibles.

[14]Système permettant plusieurs tâches d'exécutions en parallèle, ce qui est souvent le cas sur les ordinateurs modernes et les logiciels actuels.

[15]"*Max has hooks for patching a digital synthesizer, which may just be (as now) an interpreting program in the Macintosh or may (in the future) be a plug-in signal-processing card.*", [9], p. 420.

[16]En 1996, la société Steinberg publie la station de travail audionumérique Cubase 3.02 avec notamment l'intégration de la technologie VST, information d'après artisteaudio.fr/steinberg-cubase/ (page consultée en janvier 2018).

[17]Il est possible exclusivement de jouer le patch. L'édition doit être réalisée au sein du logiciel Max original.

nouveaux plugiciels en embarquant directement les patchs dans des copies indépendantes et distinctes du plugiciel original de la distribution. Le développement du projet prend fin en 2006 avec la version 3.6.1 pour Max 4.6.2, ce qui la rend aujourd'hui dépréciée et restreint fortement toute utilisation. La technologie est cependant réintégrée en 2009 au sein de l'extension Max for Live, permettant de charger des patchs au sein du logiciel Ableton Live[18], dont le fonctionnement y est plutôt similaire[19]. Cet outil puissant, de par ses nombreuses années de développement et d'usage, souffre néanmoins d'une double restriction. Celui-ci n'est plus réservé qu'à une seule station de travail audionumérique, Ableton Live, dont la technologie est fermée tout comme celle du logiciel Max. Aussi, cela freine son appropriation dans un contexte de recherche, où les notions de propriété et d'ouverture sont importantes[20] . De même dans un contexte d'utilisation lié à un cadre pédagogique, où le prix cumulé des technologies devient rapidement un frein à l'appropriation des outils de la part des étudiants universitaires.

### 3.2. De Pure Data au moteur libpd

Le logiciel Pure Data, par sa gratuité, son fonctionnement multiplateforme, son code source ouvert, est accessible à tout un chacun et accepte les ajouts et modifications par des personnes extérieures[21]. Il est ainsi une excellente alternative à l'usage de Max et offre un terrain pour l'émergence d'outils adaptés aux besoins pédagogiques mais aussi de recherche. C'est donc naturellement vers cet outil que les développeurs se tournent lorsqu'il s'agit de s'approprier un moteur audio offrant un modèle de type *patcher*. Pure Data est déjà utilisé par de nombreuses plateformes et de nombreux outils logiciels en tant que moteur audio. Un des premiers projets utilisant cette technologie est PDa (Pure Data Anywhere), réalisé en 2003 par Günter Geiger [13] et visant à faire fonctionner le moteur audio du logiciel sur un PocketPC. Par la suite en 2010, un ensemble de développeurs[22] proposent libpd, une bibliothèque en C embarquant le moteur[23] de Pure Data et offrant des interfaces de programmation en C++, Java, Processing, Objective-C et Python, qui permettent notamment un support pour les plateformes Android and iOS [14]. Cette bibliothèque est utilisée dans de très nombreux projets[24], dont les applications pour téléphones et tablettes tactiles iOS[25] avec PdParty de Dan Wilcox [15] ou son homologue pour Android, dont elle est inspirée avec PdDroidParty[26], développée par Chris McCormick. Dès lors qu'il est possible d'embarquer le moteur de Pure Data au sein d'autre application via libpd, plusieurs projets de plugiciels permettant de charger des patchs Pure Data voient le jour. Par exemple, le plugiciel au format LV2, PdLV2[27] qui fonctionne sur les systèmes d'exploitation Linux et dont le développement date d'au moins 2013, ou plus récemment le plugiciel VST, PdPulp[28] qui fonctionne sur le système d'exploitation MacOs, et dont le développement est contemporain de celui de Camomile. Ces projets sont extrêmement expérimentaux et possèdent sensiblement moins de fonctionnalités qu'un outil tel que Max for Live. Ils sont par conséquent encore difficilement utilisables dans un contexte de production. Il n'est pas donc nécessaire de revenir avec précision sur ces mises en œuvre, mais il est important de relever un problème crucial lié directement à l'architecture du moteur Pure Data. Le logiciel ayant été pensé comme une application autonome, le noyau du code restreint son usage et le résultat concret pour les utilisateurs est que seule une instance d'un plugiciel peut être chargée dans la station de travail audionumérique[29]. Ce problème est primordial à résoudre car l'utilisation multiple d'un même plugiciel dans les chaînes de traitement audio est ancré dans les usages.

### 3.3. Au-delà de libpd

Deux approches ont néanmoins réussi si ce n'est à résoudre le problème, du moins à le contourner. L'une se trouve au sein du plugiciel PdVst~ originellement développé en 2004 par Joseph Jarlo[30] puis repris depuis 2016 par Jean Yves Gratius[31]. Dans ce plugiciel VST,

[18]Le transfert de la technologie de Pluggo vers Max for Live a été annoncé en mai 2009 sur le site de Cyling'74 cycling74.com/newsletters/pluggo-technology-moves-to-max-for-live (page consultée en janvier 2018).

[19]Évidemment de nouvelles fonctionnalités sont apparues et/ou ont été modifiées mais le mécanisme général y est relativement similaire.

[20]La comparaison entre le développement de Pure Data et de jMax est un bon exemple de ce problème [12].

[21]"*Pd was instantly embraced by a huge, and extremely hip, community of users who have taken it far beyond my wildest dreams of it [...] One meaning of Pd was "Public Domain"*, [12], p. 200.

[22]Les créateurs de libpd sont à l'époque une équipe rencontrée sur internet et composée de Peter Brinkmann, Peter Kirn, Richard Lawler, Chris McCormick, Martin Roth & Hans-Christopher Steiner.

[23]Le moteur est ici pris dans son sens le large, avec la partie dédiée à l'audio, aux messages, au MIDI et autres normes, telles que l'OSC.

[24]Il serait vain de vouloir citer l'ensemble de ces projets, surtout qu'une majeure partie semble rester fermée ou inaccessible. Mais un échantillon d'exemples est disponible sur le site du projet libpd.cc (site consulté en janvier 2018).

[25]Système d'exploitation mobile développé par Apple www.apple.com/fr/ios/ios-11 (site consulté en janvier 2018).

[26]L'application est disponible sur le site www.droidparty.net (site consulté en janvier 2018).

[27]Le plugiciel est originellement développé par l'utilisateur de Github "unknownError" et est publié depuis 2013 sur le répertoire Github github.com/unknownError/pdLV2-stereo (consulté en janvier 2018). Depuis 2016, il est mis à jour et maintenu par Alex Norman et accessible sur son répertoire Github github.com/x37v/pdlv2 (consulté en janvier 2018).

[28]Le plugiciel est développé par Karl Pannek and Oliver Greschke depuis juillet 2015 et est disponible sur le site pd-pulp.net (consulté en janvier 2018).

[29]Ce diagnostic est aussi fait lors des premiers essais de mise en œuvre de Camomile, cependant les développements et mises à jours parallèles de Pure Data permettront, comme cela sera présenté dans le suite de l'article, de proposer une solution honorable et intermédiaire avant d'arriver à une solution optimale avec les améliorations plus récentes de Pure Data.

[30]Le lien crca.ucsd.edu/~jsarlo/pdvst permettant d'accéder au projet initial n'est plus valide en 2018.

[31]Le plugiciel et les informations relatives au projet sont disponibles via le répertoire Github de Jean Yves Gratius github.com/jyg/pure-data (site consulté en janvier 2018).

Pure Data n'est pas réellement embarqué. Pour chaque instance du plugiciel, une instance de l'application de Pure Data est démarrée de telle sorte que chaque application possède un espace mémoire qui lui est propre, évitant dès lors des conflits[32]. Aussi, le plugiciel fonctionne en réalité comme un pont entre ces applications Pure Data associées aux plugiciels et la station de travail audionumérique. Le problème majeur reste que cet outil est restreint au système d'exploitation Windows. Il serait donc nécessaire d'étudier son potentiel fonctionnement sur d'autres systèmes d'exploitation, mais aussi de vérifier si cette approche n'implique pas de latence et si la communication entre les instances de Pure Data et la station de travail audionumérique via ce système est réellement stable, notamment dans un contexte de *multithreads*[33]. Une deuxième approche est proposée par la plateforme Heavy[34] distribuée par la société Enzien Audio, qui offre la possibilité de générer directement un plugiciel VST à partir d'un patch Pure Data. La plateforme offre un système de compilation en ligne, mais au lieu d'intégrer Pure Data au plugiciel, Heavy interprète le patch avec son propre moteur qui, quant à lui, offre un fonctionnement multi-instances. La force première de cette approche est aussi son principal défaut. En effet, la plateforme n'utilisant pas réellement Pure Data, elle ne propose qu'un sous ensemble de ses fonctionnalités, ce qui restreint les possibilités, et implique immanquablement un retard par rapport aux mises à jour de la distribution de Pure Data vanilla[35]. De plus, l'exacte similitude du rendu sonore entre le patch dans le logiciel Pure Data et le plugiciel généré n'est pas assurée, du fait de la réécriture totale ou partielle du moteur, afin de le rendre opérable dans un système multi-instances.

## 4. MISE EN ŒUVRE DE CAMOMILE

Lors de la définition des spécifications techniques et fonctionnelles de l'outil Camomile, deux enjeux majeurs sont apparus. Cette partie revient donc sur les choix opérés lors de mise en œuvre de l'outil Camomile d'une part, au niveau de l'intégration du moteur *patcher* dans le plugiciel et d'autre part, au niveau de la création d'un système de gestion réciproque entre le moteur *patcher* et le plugiciel. Cela en les confrontant à l'état de l'art présenté dans la partie précédente. À ce jour, deux versions majeures de Camomile ont été réalisées. Le changement de version correspond à une amélioration de la technologie utilisée. Sans pour autant revenir sur l'ensemble des spécificités de chaque version, les avantages et les conséquences de ces choix seront discutés.

### 4.1. Intégration du moteur dans le plugiciel

Pour les raisons évoquées précédemment, le plugiciel vise à utiliser le moteur de Pure Data qui offre une licence très libre[36] et un usage gratuit, tout en ayant une large communauté d'utilisateurs très actifs [12]. D'un autre côté, l'interface de programmation applicative JUCE[37] a été choisie, car elle offre entre autre la possibilité avec un seul code générique, de créer des plugiciels pour les formats VST et Audio Unit, ainsi que pour les systèmes d'exploitations Windows, Linux et MacOs. L'enjeu est alors de réussir à embarquer le moteur de Pure Data au sein du plugiciel. Or, comme cela a été évoqué précédemment, la difficulté première est de rendre le moteur de Pure Data compatible avec un usage multi-instances dans un contexte de *multithreads*.

#### *4.1.1. L'approche séquentielle*

Les développements de Pure Data pour la version 0.46, avec le début de l'intégration par Miller Puckette de la gestion du multi-instances, ont simplifié la première mise en œuvre de Camomile. Ces modifications du cœur du moteur, bien qu'insuffisantes à une mise en œuvre directe dans le contexte d'un plugiciel audio, ont permis de proposer une première stratégie consistant à forcer le caractère séquentiel des opérations. Pour cela, une interface similaire à libpd a été créée pour répondre spécifiquement à ce problème. La méthode consiste à restreindre l'accès au moteur à une instance à la fois, tout en limitant chaque instance à une seule opération, en utilisant un système de verrous (*locks*). Afin d'offrir un accès dans un contexte de *multithreads*, le système utilise des listes chaînées pour enregistrer les messages et les instructions qui sont lues et exécutées par la suite de manière séquentielle, avant de traiter le signal audionumérique de chaque instance. L'enjeu suivant est de simplement faire concorder l'interface JUCE avec l'interface enveloppant le moteur Pure Data. En avril 2016, une première version fonctionnelle destinée à un large usage[38] a pu être publiée. Elle a été testée sur de nombreuses plateformes logicielles et à chaque fois sur les systèmes d'exploitation pour lesquels elles sont disponibles[39]. Malgré l'approche radicale à forcer le

[32]Cette approche possède aussi l'avantage éventuel comparé aux approches présentées précédemment, de pouvoir utiliser directement l'interface graphique native. Il n'est néanmoins pas forcément souhaitable de vouloir utiliser l'interface graphique originale de Pure Data au sein d'un plugiciel audionumérique. Une interface spécifique et plus adaptée peut potentiellement être plus appropriée. Un autre avantage est la possibilité de charger les bibliothèques d'objets externes.

[33]Faute de moyens techniques et de temps, cela n'a pas pu être réalisé pour le moment.

[34]La plateforme Heavy est proposée par depuis 2016 sur enzienaudio.com (site consulté en janvier 2018).

[35]Distribution principale de l'application réalisée par Miller Puckette.

[36]Pure Data ou le projet libpd sont tous deux sous la même licence définie par "Standard Improved BSD" et distribuée avec les codes sources.

[37]JUCE est une interface de programmation applicative orientée vers le traitement du signal audionumérique distribué par la société Roli juce.com (site consulté en janvier 2018).

[38]La version 0.0.7 est toujours disponible sur le répertoire Github du projet.

[39]Une liste non exhaustive de ces plateformes – comprenant notamment les logiciels Reaper, Cubase, Ableton Live, Tracktion,

caractère séquentiel des opérations, le système était jouable et n'offrait pas d'artefacts en dehors d'une hausse de l'utilisation du processeur – résultat de l'attente de déverrouillage d'une instance par une autre.

#### *4.1.2. L'approche parallèle*

Plus récemment avec la version 0.47 de Pure Data, les derniers problèmes de l'utilisation multi-instances dans un contexte de *multithreads* ont été résolu en utilisant notamment la mémoire locale de *thread*. En gardant la mémoire associée à une instance locale sur chaque *thread*, cela permet d'utiliser en concurrence plusieurs instances sur des *threads* différents[40]. Néanmoins, l'usage d'une telle approche demande de revoir une grande partie de l'interface mise en œuvre précédemment. Aussi, afin que ce travail puisse être utile à une plus large communauté de programmeurs [14], le choix a été fait d'utiliser libpd et d'y intégrer ces améliorations. Les interfaces de programmation élémentaires de libpd pour le multi-instances ont été mises à jours notamment par Miller Puckette et Dan Wilcox. L'enjeu était alors de rendre certains aspects importants – notamment la réception des messages, des événements MIDI ou des notifications de console – compatibles avec le multi-instances. Aussi, la mise à jour pour le multi-instances et le *multithreads* du noyau de Pure Data n'a pas encore été réellement confrontée à un usage concret. Cette mise en œuvre était l'occasion de tester l'approche, et de corriger les quelques cas difficiles à prévoir[41]. En décembre 2017, la deuxième version de Camomile a pu être publiée et grâce au relatif succès de la précédente version, de nombreux retours d'utilisation constructifs ont permis de déployer rapidement les outils sur les différents systèmes d'exploitation[42].

### 4.2. Gestion réciproque entre le moteur et le plugiciel

La gestion du moteur *patcher* par le plugiciel et réciproquement des caractéristiques du plugiciel via le moteur, recouvre plusieurs aspects.

#### *4.2.1. La communication et le partage de données*

Un premier aspect important de la gestion du moteur par le plugiciel concerne la communication, et de manière générale le partage de données audio, MIDI ou encore de type message. La réception et l'envoi du signal audionumérique du plugiciel vers le moteur *patcher* et réciproquement, peuvent être réalisés respectivement avec les objets *adc~* et *dac~*. De manière analogue, les événements MIDI reçus par le plugiciel peuvent être récupérés dans un patch via les objets MIDI natifs de Pure Data. Et des événements MIDI peuvent être générés du patch et envoyés vers le plugiciel via les objets MIDI équivalents pour la sortie. Enfin, la réception de messages du plugiciel vers un patch est quant à elle réalisée avec les objets *receive* et les messages sont transmis du patch vers le plugiciel avec les objets *send*. Ces messages sont notamment utilisés pour modifier dans un patch les valeurs des paramètres mais aussi afin d'être notifié de leurs changements par la station de travail audionumérique (Figure 3)[43].

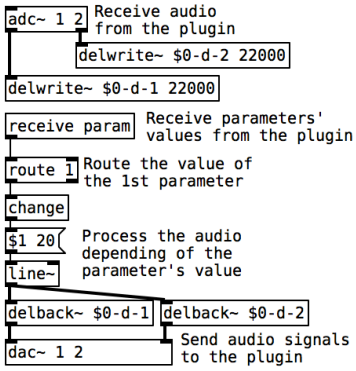



**Figure 3.** Patch Pure Data utilisé dans le plugiciel Bulgroz où le signal en entrée, reçu par l'objet *adc~*, est traité en fonction de la valeur du premier paramètre, reçu par l'objet *receive*, avant d'être renvoyé au plugiciel, via l'objet *dac~*.

L'avantage de cette approche est que, contrairement à Max For Live, aucun objet externe et spécifique à Camomile n'est nécessaire au fonctionnement d'un patch dans le plugiciel. Cela évite une possible obsolescence des patchs et les rend utilisables en dehors du plugiciel. Il est intéressant de remarquer qu'afin d'assurer une bonne distribution des messages entre les instances, la première version nécessitait l'utilisation des indices spécifiques aux patchs dans les symboles de réception et d'envoi[44]. La deuxième version avec sa gestion du multi-instances plus poussée, permet de se libérer de cette contrainte.

#### *4.2.2. Le chargement des patchs*

Un deuxième aspect notable de la gestion du moteur par le plugiciel, concerne la méthode de chargement des patchs. Deux approches peuvent être envisagées et ont été mises en œuvre respectivement dans la première et la deuxième version du plugiciel. La première version offrait la possibilité de charger des patchs dynamiquement via une boîte de dialogue. Cette approche possède l'avantage d'ouvrir rapidement un

Ardour, etc. – et des systèmes d'exploitation – Windows, MacOS et Linux en 32/64bits - est disponible sur le wiki du répertoire Github.

[40]L'accès à une même instance via plusieurs *threads* peut néanmoins toujours poser problème et l'accès sur un *thread* de plusieurs patchs est obligatoirement séquentielle.

[41]Les dernières modifications devraient être acceptées prochainement sur les branches principales de libpd et Pure Data.

[42]Cette version a déjà été testée sur un grand nombre de plateformes logicielles – telles que Reaper, Ableton Live, Tracktion, Bitwig, etc. – et de systèmes d'exploitation – Windows 32/64bits, MacOS 32/64bits et Linux 64bits.

[43]Ces approches peuvent être aussi utilisées pour la gestion des préréglages ou encore pour la configuration des canaux audio.

[44]Ces indices peuvent être intégrés en utilisant les caractères *$0* dans ces symboles.

patch dans la station de travail audionumérique. Associée avec la fonction de rechargement du patch courant, c'est une fonctionnalité très utile pour la création des patchs, mais qui pose néanmoins des problèmes. Plusieurs aspects du fonctionnement du plugiciel dépendent du patch chargé, comme les paramètres, les préréglages, les configurations de canaux supportées, *etc.* Or, changer de patch après le chargement initial du plugiciel par la station de travail audionumérique, implique de modifier ces éléments à la volée, ce qui n'est pas bien supporté par ces logiciels[45]. De plus, la localisation des patchs n'est pas assurée lorsque le projet de la station de travail audionumérique est partagé[46]. Enfin, utiliser l'interface graphique pour le chargement du patch empêche l'utilisation du plugiciel dans les stations de travail qui ne supportent justement pas les interfaces graphiques spécifiques aux plugiciels[47]. Pour répondre à ce problème, la deuxième version de Camomile a choisi d'associer chaque patch – ou un ensemble de patchs – à une nouvelle copie du plugiciel Camomile original. En liant et en intégrant de la sorte les patchs au fichier binaire du plugiciel, le chemin relatif entre les deux est toujours préservé. À la première instanciation du plugiciel dans la station de travail, le ou les patchs associés sont automatiquement chargés et l'ensemble des informations nécessaires au fonctionnement du plugiciel – paramètres, préréglages, configurations des canaux, *etc.* – peut être récupéré. Par la suite, le patch d'une instance du plugiciel peut être rechargé à la volée[48] – permettant donc de modifier le patch, que ce soit le moteur sonore ou l'interface graphique – mais les informations fournies à la station de travail audionumérique, et qui doivent nécessairement rester inchangées, ne sont pas modifiées.

### *4.2.3. Définition des propriétés du plugiciel*

Enfin, un dernier aspect important concerne la manière de définir ces différentes informations liées au fonctionnement du plugiciel. Là encore, l'approche est différente entre les deux versions de Camomile. La première version se limitait aux seules informations des paramètres. Aussi, il avait semblé judicieux de définir les paramètres à l'aide des interfaces graphiques utilisateurs du patch principal. Par exemple, une glissière (*slider*) ou une boîte nombre générait automatiquement un paramètre en fonction du label et de la plage de valeur jouable définie dans les propriétés de l'objet graphique[49]. Cette approche possède l'avantage de savoir dans le plugiciel quand la valeur d'un paramètre est modifiée par l'utilisateur via cette interface, et de simplifier la mise en œuvre du plugiciel et la création du patch. Cependant, un tel choix possède des inconvénients majeurs. L'approche se complexifie dès lors que l'utilisateur souhaite utiliser plusieurs interfaces graphiques pour contrôler un même paramètre (Figure 2). Mais surtout, l'ordre des paramètres[50] dépend de l'ordre de création des interfaces graphiques. Aussi, dans le processus de création d'un patch pour le plugiciel, définir l'ordre des paramètres peut nécessiter de recréer les interfaces graphiques et leurs connexions. Ce qui peut être laborieux et source d'erreurs. Enfin il restait à définir de nouvelles méthodes pour configurer les autres options du plugiciel, telles que les préréglages, les configurations audio supportées, etc.

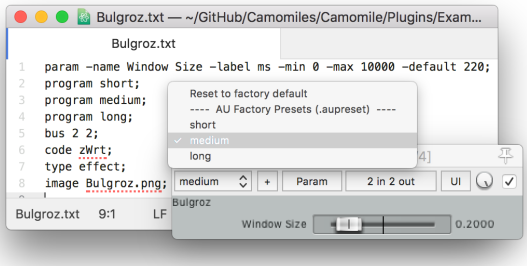



**Figure 4.** En arrière-plan, le fichier texte permettant de définir les propriétés du plugiciel Bulgroz avec Camomile, dont notamment le paramètre de taille de fenêtre et trois préréglages. Au premier-plan, l'interface graphique par défaut du plugiciel dans le logiciel Reaper permet de modifier la valeur du paramètre et de sélectionner les préréglages.

Afin de remédier à ces problèmes, la dernière version de Camomile utilise un fichier texte additionnel pour définir l'ensemble des informations nécessaires à un plugiciel. La syntaxe de ce fichier est simple. Chaque ligne permet de définir une option, de rajouter un paramètre ou un préréglage[51] ce qui ouvre la possibilité d'écrire ou de charger le fichier via l'objet *text* de Pure Data (Figure 4. En arrière-plan, le fichier texte permettant de définir les propriétés du plugiciel Bulgroz avec Camomile, dont notamment le paramètre de taille de fenêtre et trois préréglages. Au premier-plan, l'interface graphique par défaut du plugiciel dans le logiciel Reaper permet de modifier la valeur du paramètre et de sélectionner les préréglages.). Enfin, un système de

[45]Cette approche est normalement interdite par la norme VST. Une restriction est définie dans la documentation du VST 3 Plug-In SDK github.com/steinbergmedia/vst3sdk (page consultée en janvier 2018).

[46]Les patchs n'étant pas directement associés au projet en cours, au fichier binaire du plugiciel ou à la station de travail, son chemin est absolu et lorsqu'un projet est partagé sur un autre ordinateur, le plugiciel ne parvient pas à retrouver le patch et il est nécessaire de redéfinir manuellement tous les chemins.

[47]Certaines stations de travail audionumériques supportent encore mal les interfaces graphiques des plugiciels. C'est le cas notamment du logiciel libre et gratuit Audacity audacity.fr (page consultée en janvier 2018).

[48]Cette fonctionnalité peut être très utile notamment lors de la création du plugiciel. La version 1.0.4 du plugiciel offre, pour cela, la possibilité de recharger le patch manuellement – en cliquant sur un bouton dédié – ou automatiquement à chaque nouvelle sauvegarde du patch.

[49]Cette approche est aussi définie dans la documentation de la version 0.0.7 du plugiciel disponible sur le wiki du répertoire Github.

[50]L'ordre des paramètres est très important, car les stations de travail audionumérique utilisent leurs indices (ou positions) – et non leurs noms – comme identifiants des paramètres pour organiser les informations qui leurs sont relatives, telles que les automations.

[51]Cette approche est définie dans la documentation du plugiciel et sera développée plus longuement dans une prochaine publication.

notifications du plugiciel vers le patch et réciproquement du patch vers le plugiciel, permet d'associer les interfaces graphiques aux paramètres[52]. La version actuelle du plugiciel permet donc un support complet de l'ensemble des fonctionnalités génériques et des spécifications des plugiciels, tout en préservant une cohérence avec le langage et les usages de Pure Data.

## 5. BILAN ET PERSPECTIVES

Camomile offre à présent un outil compatible sur les trois principaux systèmes d'exploitation avec la plupart des stations de travail audionumérique supportant des plugiciels. Les choix opérés dans la dernière version du plugiciel – que ce soit pour le chargement de patchs, la communication entre le moteur et le plugiciel ou encore la création de patchs ainsi que la définition des propriétés du plugiciel – ont permis de supporter pleinement l'ensemble des fonctionnalités usuellement offert par ce type de formats : la gestion des paramètres, des préréglages, des configurations audio, des événements MIDI, des informations de la tête de lecture[53] ou encore la création d'une interface graphique adaptée et fonctionnelle. Ainsi, des patchs correctement configurés et tirant parti de l'ensemble de ces fonctionnalités, peuvent permettre de générer des plugiciels rivalisant que ce soit sur l'aspect sonore ou l'aspect ergonomique avec des plugiciels codés et compilés de manière classique. De plus, l'outil n'est pas seulement limité à un usage expérimental, mais peut très bien être utilisé dans un contexte de production musicale abouti ou dans un contexte pédagogique[54] (Figure 5).

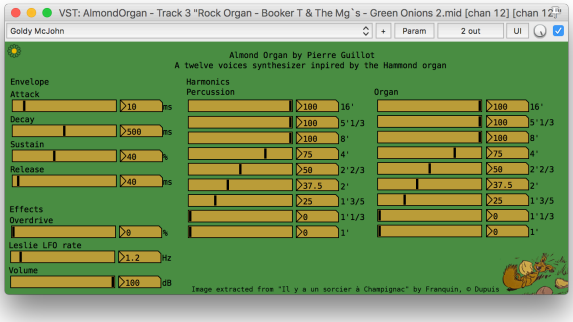



**Figure 5.** Le plugiciel *AlmondOrgan* au format VST, un synthétiseur douze voix inspiré par l'orgue Hammond, créé avec Camomile et intégrant de multiples fonctionnalités comme la gestion des événements MIDI, des paramètres et des préréglages.

Camomile est, en outre, un outil extrêmement intéressant pour les développeurs car il permet de créer rapidement des prototypes de plugiciels en prenant en compte tous les aspects de la mise en œuvre : moteur audio, interfaces graphiques, messages, *etc.* Et au-delà des prototypes, les plugiciels créés avec Camomile peuvent être publiés en tant que tels. Le plugiciel jouit déjà d'une relative notoriété dans la communauté d'utilisateurs de Pure Data et de l'audionumérique libre de manière générale. Mais il serait intéressant d'ouvrir la communauté d'utilisateurs via le partage direct de plugiciels créés avec Camomile. Cela pourrait donner envie à des utilisateurs exclusifs des stations de travail audionumérique de s'intéresser aux approches de type *patcher*. Sur le plan du développement, Camomile aura été un excellent terrain d'expérimentations et de mise à l'épreuve du support *multithreads* et multi-instances de Pure Data. Il aura permis de révéler un certain nombre de problèmes qui restaient à corriger. Enfin, certains points sont néanmoins encore à explorer et d'autres problèmes à débloquer. C'est le cas notamment de la gestion des bibliothèques externes d'objets qui nécessitent *a priori* d'être compilées avec le plugiciel. Il existe deux raisons à cela. D'une part, les stations de travail audionumériques semblent pour la plupart bloquer le chargement dynamique de bibliothèques externes. Et d'autre part, il est nécessaire de vérifier si les bibliothèques externes n'interfèrent pas avec les conditions d'activation du support du multi-instances et du *multithreads* de Pure Data. Il serait aussi intéressant de réfléchir à une amélioration ou une extension de la création et de la gestion de l'interface graphique. Il s'agirait de prendre en compte le système de structure de données graphiques [16] ou d'offrir un système dynamique permettant d'intégrer ses propres graphismes aux interfaces, afin de se rapprocher d'un plugiciel plus classique[55]. Enfin, il serait intéressant d'intégrer la génération de plugiciels au format LV2[56] mais aussi au format Audio Unit v3[57] qui permettrait de charger les plugiciels sur des tablettes et téléphones iOS.

## 6. REMERCIEMENTS

*Je tiens à remercier toute la communauté de développeurs de Pure Data et de libpd, et particulièrement Miller Puckette et Dan Wilcox, pour leurs conseils et leurs explications mais aussi les*

[52]Cette approche est aussi définie dans la documentation du plugiciel et sera aussi développée plus longuement dans une prochaine publication.

[53]Il est en effet possible d'utiliser toutes les informations fournies par la station de travail audionumérique telles que le chiffrage de la mesure, le tempo de la mesure, la position de la tête de lecture, etc.

[54]Le plugiciel a notamment été présenté à l'Université Paris 8 aux étudiants en 2016 et 2017, dans le cadre des cours d'Introduction à la programmation avec Max et Pure Data 1 dispensés par moi-même et en 2018, aux étudiants du cours de Composition électroacoustique 2 dispensé par Alain Bonardi et suite auquel ils ont été invité à créer leurs propres plugins.

[55]Le plugiciel offre déjà à la possibilité de charger des images de fond (Figure 2 & Figure 5) mais il serait intéressant de pouvoir replacer les graphismes natifs de objets de Pure Data, tels que l'interrupteur (*toggle*), la glissière (*slider*) ou encore la boîte nombre (*numbox*), en utilisant des séries d'images données par l'utilisateur.

[56]Le choix a été fait pour le moment d'attendre que JUCE supporte ce format plutôt que coder un plugiciel indépendant spécifique.

[57]Grâce au support de ce format par JUCE, il peut être déjà envisagé de publier une version Audio Unit v3 de Camomile mais faute de temps et de moyen techniques il n'a pas encore été possible de tester l'approche.

*utilisateurs de Camomile pour leurs retours d'utilisation et leurs suggestions. Je tiens aussi à remercier toute l'équipe du CICM pour l'intérêt porté à ce projet, et particulièrement Alain Bonardi et Eliott Paris pour leurs remarques et leurs suggestions.*

## 7. REFERENCES

[1] Puckette M. "Pure Data : Another Integrated Computer Music Environment", Proceedings of the Second Intercollege Computer Music Concerts, p. 37-41, Tachikawa, Japon, 1997.

[2] De la Hogue T., Baltazar P., Catherine M. D., Chao J. et Bossut, C. "Ossia : Open Scenario System For Interactive Applications", actes des Journées d'Informatique Musicale, p.78-84, Bourges, France, mai 2014.

[3] Coduys T., Lefèvre A. et Pape G. "IanniX", actes des Journées d'Informatique Musicale, Montbéliard, France, juin 2003.

[4] Sèdes A., Guillot P. et Paris E. "The HOA library, review and prospect", Proceedings of the ICMC-SMC 2014, p. 855-860, Athènes, Grèce, septembre 2014.

[5] Guillot P., Paris E. et Deneu M. "La bibliothèque de spatialisation HOA pour Max/MSP, Pure Data, VST, FAUST…", Revue Francophone d'Informatique et Musique (En ligne), n° 3, automne 2013, URL : revues.mshparisnord.org/rfim/index.php?id=245.

[6] Favreau E., Fingerhut M., Koechlin O., Potacsek P., Puckette M. et Rowe R., "Software Developments for the 4X Real-time System", Proceedings of the International Computer Music Conference 1986, p. 369-373, La Haye, Pays-Bas, 1986

[7] Guillot P. "La Représentation Intermédiaire et Abstraite de l'espace Comme Outil de Spatialisation du Son", thèse de doctorat, CICM, Université Paris 8, Saint-Denis, France, 2017.

[8] Paris E., Millot J., Guillot P., Bonardi A. et Sèdes A. "Kiwi : Vers un Environnement de Création Musicale Temps-Réel Collaboratif - Premiers Livrables du Projet Musicoll", Actes des Journées d'Informatique Musicale, Paris, France, mai 2017.

[9] Puckette M. "The Patcher", Proceedings of the International Computer Music Conference 1988, p. 420-429, Cologne, Allemagne, 1988.

[10] Puckette M. "Combining Event and Signal Processing in the MAX Graphical Programming Environment", Computer Music Journal, vol. 15, n° 3, p. 68-77, automne 1991.

[11] Zicarelli D. "Developing Plug-ins in Max/MSP for Pluggo - updated information for Pluggo 3", Documentation du plugiciel Pluggo publiée par Cycling'74, révision 4 du 16 mai 2002.

[12] Puckette M. "Who owns our software ? A First-person Case Study", Proceedings of the International Symposium on Electronic Art, p. 200-202, Helsinki, Finlande, 2004.

[13] Geiger G. "PDa : Real Time Signal Processing and Sound Generation on Handheld Devices," Proceedings of the International Computer Music Conference, Singapour, septembre-octobre 2003.

[14] Brinkmann P., Kirn P., Lawler R., McCormick C., Roth M. et Steiner H.-C. "Embedding Pure Data with libpd", Proceedings of the Pure Data Convention, Weimar, Allemagne, aout 2011.

[15] Wilcox D., "PdParty : An iOS Computer Music Platform using libpd", Proceedings of the Pure Data Convention, New-York, États-Unis, novembre 2016.

[16] Puckette M. "Using Pd as a score language", Proceedings of the International Computer Music Conference (ICMC 2002), p. 184-187, Göteborg, Suède, 2002.

# ORDONNANCEMENT ADAPTATIF D'UN GRAPHE AUDIO AVEC DÉGRADATION DE QUALITÉ

*Pierre Donat-Bouillud*
Sorbonne Université/STMS/Inria

**RÉSUMÉ**

Les systèmes interactifs musicaux sont des systèmes particulièrement dynamiques qui combinent du traitement du signal avec du contrôle en temps réel. Ils sont souvent utilisés sur des plateformes où il n'est pas possible d'avoir des garanties temps réel précises. Nous présentons ici un algorithme de dégradation temps réel en ligne pour obtenir un compromis entre la qualité audio et les retards par rapport aux échéances audio, dans un graphe audio dynamique. Nous évaluons expérimentalement les performances de l'algorithme.

## 1. INTRODUCTION

Les systèmes interactifs musicaux (SIM) [15] sont des systèmes programmables de création artistique qui combinent des traitements audio avec du contrôle en temps réel dans un graphe audio. Les graphes audio connectent des nœuds de traitement du signal dans un graphe reconfigurable en ligne et se plient à des contraintes temps réel non critiques, par rapport aux systèmes embarqués dans un avion par exemple, mais plus exigeantes que pour de la vidéo.

Compositeurs et musiciens utilisent généralement ces SIM sur des systèmes d'exploitation grand public comme Windows, macOS, ou Linux, où une estimation fiable du temps d'exécution dans le pire des cas (WCET) est difficile, à cause d'une hiérarchie complexe des caches du processeur, de l'absence d'ordonnanceurs temps-réel, du manque d'isolation temporelle entre tâches, et de la difficulté de prédire quelles tâches seront exécutées à un instant donné.

Par conséquent, nous ne pouvons pas supposer que l'on connaît le WCET des tâches mais nous pouvons réagir aux changements dans l'environnement d'exécution en modifiant le temps d'exécution d'une tâche avec le paradigme de la *programmation approximée* [17], par dégradation du graphe audio. Il ne s'agit pas de seulement dégrader un seul nœud d'un graphe audio, mais de *choisir les nœuds à dégrader*, tout en préservant les contraintes temps réel. Dans ce contexte de traitement du signal, les nœuds sont vus comme des *boites noires* et les dégradations s'effectuent seulement sur les flux de données entre les boites, en *rééchantillonnant* ces flots de données. On pourra aussi envisager de *substituer* un nœud par une autre version du nœud moins demandeuse en ressource de calcul.

Si on considère que le temps est une ressource, dans les systèmes temps-réel, le temps est souvent la seule ressource à être dégradée [7] (en ratant une échéance). Ici, nous dégradons aussi d'autres ressources et nous cherchons à trouver un compromis explicite entre diverses mesures de qualité de la tâche, comme le taux d'échantillonnage du signal.

Dans un premier temps, nous avons étudié comment choisir les nœuds à dégrader au moment de l'exécution, *en ligne.* L'ordonnanceur doit tenir compte de son propre temps de calcul et nous faisons en sorte que les calculs de l'ordonnanceur eux-même soient les plus légers possibles. Un modèle particulièrement bien adapté à décrire des tâches de traitement du signal et les flots entre les nœuds est le modèle *flot de données* [8]. Nous montrerons comment nous pouvons adapter le paradigme de *la programmation approximée* au modèle *flot de données*.

Nos contributions sont les suivantes :

— intégrer les dégradations au modèle *flot de données*
— détermination des nœuds à dégrader (en ligne)
— application de ces dégradations au cas particulier d'un graphe audio

## 2. CONTEXTE ET MOTIVATIONS

### 2.1. Les systèmes interactifs musicaux

Les SIM servent à jouer des pièces de musique, que l'on décrit par des partitions destinées à l'ordinateur, sous la forme de programme. Pour cela, ils manipulent des flux audio, au moment du concert, en temps réel, à l'aide de divers effets audio. Ils effectuent donc un traitement du signal, ils remplissent périodiquement des tampons audio, les envoient à la carte son, et permettent de contrôler les traitements, avec des contrôles apériodiques (comme des changements de l'interface graphique) ou bien périodiques (par exemple via un oscillateur basse fréquence). Les flux audio et les contrôles sont traités dans un graphe audio de nœuds de calcul audio, qui peut être dynamique, c'est-à-dire que des nœuds peuvent être ajoutés ou retirés pendant l'exécution.

Puredata [13] et Max/MSP [20] sont des exemples de SIM, qui affichent graphiquement le graphe audio mais rendent compliquée sa modification de façon dynamique comme résultat d'un calcul. D'autres SIM, comme ChucK [18] ou SuperCollider [11], sont plus dynamiques. Dans Antescofo [6], des musiciens humains et un ordinateur

Pierre Donat-Bouillud, *Ordonnancement adaptatif d'un graphe audio avec dégradation de qualité*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 25–33, 2018. 

peuvent interagir sur scène en concert, à l'aide de stratégies de synchronisation sophistiquées spécifiées par une partition augmentée qui peut aussi décrire des graphes audio dynamiques [5].

### 2.2. Les contraintes temps-réel pour l'audio

La carte son exige que des échantillons audio soient écrits dans son tampon d'entrée périodiquement. Pour le taux d'échantillonnage d'un CD de 44.1 kHz par exemple, et une taille de tampon de 64 échantillons, la période audio est de 1.45 ms. Suivant la latence visée et les ressources de la plateforme considérée, la taille du tampon peut aller de 32 échantillons pour des ordinateurs dédiés au traitement du son, à 2048 échantillons pour certains téléphones tournant sous Android.

Les contraintes temps-réel pour l'audio sont non critiques, mais les exigences sont plus strictes que pour la vidéo. Pour la vidéo, perdre une image parmi les 24 images par seconde n'entraîne pas une baisse visible de la qualité ; de nombreux protocoles de *streaming* [1] se le permettent donc. Au contraire, rater une échéance pour une tâche audio est immédiatement audible.

**Sous-alimentation du tampon** Le pilote audio utilise un tampon circulaire dont la taille est un multiple de la taille du tampon de la carte son. Si la tâche audio rate une échéance, elle ne remplit pas le tampon assez rapidement. Suivant l'implémentation, les tampons précédents sont rejoués (l'effet *mitraillette*) ou du silence est joué, ce qui conduit à de l'audio avec des craquements ou des clics dus aux discontinuités dans le signal, comme on peut le voir sur la Figure 1.

Un tampon de grande taille aide à diminuer ce non-remplissage, mais augmente la latence audio.

**Débordement du tampon** De la même manière, dans certaines implémentations, remplir le tampon audio trop rapidement peut amener à des discontinuités audibles dans le son en sortie si les échantillons de trop ne peuvent pas être stockés pour être utilisés par la suite.

### 2.3. Motivations

Les systèmes d'exploitation grand public comme Windows, macOS ou Linux ne sont pas des systèmes temps-réel et ne donnent aucune garantie forte sur les échéances des traitements audio (voir la Figure 2). Les applications effectuant des traitements audio cohabitent avec de nombreuses autres applications et entrent en compétition pour une part de temps CPU. Par ailleurs, les SIM sont de plus en plus portés vers des cartes embarqués comme le Raspberry Pi et doivent s'adapter aux ressources de calcul limitées de ces plateformes.

Pour relever ces défis, et tenir compte de la complexité croissante des œuvres, les SIMs ont plutôt choisi d'exiger plus de ressources de calcul disponibles et d'augmenter le parallélisme des partitions interactives pour profiter de

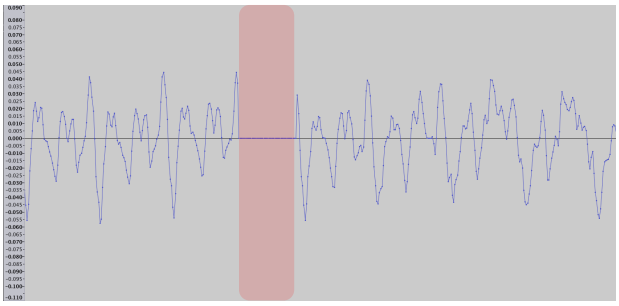


**Figure 1** : Le traitement audio a raté l'échéance. Ainsi, aucun échantillon audio produit par ce traitement audio pour ce cycle audio ne peut être envoyé au tampon audio de la carte son. Dans cette implémentation, le système envoie du silence, c'est-à-dire des échantillons à zéro. Cela entraîne une discontinuité du signal à l'endroit de la bande rouge, et donc un *clic*.

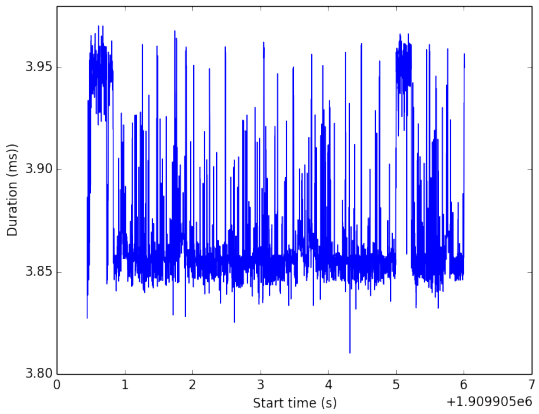



**Figure 2** : La ressource temporelle accordée au *callback* audio sur un Mac Book Pro avec Mac OS X. Bien qu'elle soit centrée autour de 3.94 ms, il y a des valeurs éloignées et elle peut varier jusqu'à 200 $\mu s$.

la généralisation des processeurs multicœurs. Un exemple de système qui s'appuie sur des architectures parallèles est l'ordonnanceur de tâches Supernova [3] pour SuperCollider. Cependant, ces nouveaux ordonnanceurs ne parallélisent pas les partitions automatiquement, requièrent des instructions explicites, comme `ParGroup` pour SuperNova et `poly` pour Max/MSP, et sont donc difficiles à programmer.

Cependant, une autre façon de faire face à ces défis est d'explorer comment les traitements audio peuvent être dégradés sans que les dégradations soient perçues (ou en quantifiant ces dégradations), au moment où les ressources de calcul demandées par les pièces augmentent (chaînes audio à 96 kHz, comme dans la pièce Re Orso de Marco Stroppa [1] ), en même temps que les supports – cartes embarquées, systèmes enfouis, plateformes mobiles – sur lesquels peuvent être jouées les pièces se démocratisent.

1 . http ://brahms.ircam.fr/works/work/27678/

## 2.4. Travaux en rapport

Certaines approches en ordonnancement ont déjà porté sur l'ordonnancement adaptatif, en temps réel critique ou non-critique, soit en supprimant complètement des tâches, soit en les dégradant, avec le paradigme de programmation approximative ou la criticité mixte. Une autre possibilité pour s'adapter aux changements des ressources de calcul disponibles pendant une exécution est d'effectuer des réservations d'une certaine capacité de calcul.

### *2.4.1. Le paradigme de programmation approximative*

Il s'agit d'un paradigme de programmation dans lequel on autorise des erreurs dans les calculs en échange d'une amélioration des performances temporelles. La notion de correction d'un programme est relâchée pour celle d'une correction avec une erreur quantitative. Ce paradigme permet de mettre au point des systèmes dans lesquels on recherche un compromis entre une bonne qualité et les performances ou la consommation en énergie. Venkataramani et al., dans [17], proposent les critères suivants pour décider de la pertinence d'utiliser ce paradigme :

- il n'y a pas une réponse unique, mais un intervalle de réponses est acceptable ;
- les utilisateurs se sont habitués à obtenir des résultats *acceptables*, suffisamment bons mais pas parfaits ;
- les données d'entrée sont bruitées et les algorithmes qui les traitent sont conçus de façon à prendre en compte ce bruit ;
- des schémas de programmation qui diminuent les approximations sont utilisés.

Une stratégie assez basique [10] consiste à diviser les tâches du système entre une partie obligatoire et une partie optionnelle, qui peut ne pas être exécutée en cas de surcharge du processeur. Cette stratégie rend l'ordonnancement temps réel assez simple à mettre en place, avec peu de calculs supplémentaires induits par l'algorithme d'ordonnancement mais ne prend pas en compte les dépendances entre tâches, en particulier dans l'estimation de la qualité.

Un autre modèle de calcul [19] utilise un graphe pour représenter un programme de type *map-reduce*, avec des nœuds de calcul et des nœuds d'agrégation. Il génère hors ligne des versions approximatives étant donnée une certaine marge d'erreur en paramètre. Les transformations qui permettent de dégrader sont de deux types : des transformations de substitution et des transformations par échantillonnage, où l'entrée est sous-échantillonnée aléatoirement. Cependant, ce modèle est destiné à des applications de traitement par lots pour des données massives et ne prend pas en compte les contraintes temps-réel ; il requiert par ailleurs une phase préliminaire de profilage, ce qui empêche de l'appliquer à des graphes reconfigurables et dynamiques.

### *2.4.2. Criticité mixte*

Dans les systèmes temps réel dur à criticité mixte [4], les tâches à haute criticité doivent impérativement être ordonnancées, alors que les tâches moins critiques peuvent être supprimées au cas où cela conduirait à outrepasser des échéances. Dans notre cas cependant, toutes les tâches, liées entre elles dans un graphe, ont la même criticité.

### *2.4.3. La réservation de ressources*

Dans cette approche, une partie des capacités de calcul du processeur est réservée [2] à certaines tâches. Cela est justifié lorsque les tâches en présence sont des tâches différentes en compétition pour les ressources : dans le cas du multimédia par exemple, les tâches vidéo et les tâches audio peuvent se voir donner des réservations différentes, comme les tâches audio sont *plus temps-réel* (plus critiques) que les tâches vidéos. Cependant, la réservation de ressources n'est pas adaptée au cas où les tâches sont identiques (toutes des effets audio) et liées par des relations de dépendance de données.

# 3. MODÈLE DE GRAPHE AUDIO

Dans un graphe audio, des flux audio circulent des entrées aux sorties à des taux différents selon les contraintes des nœuds du graphe. Le modèle *flot de donnée* [8] est particulièrement adapté pour représenter de telles tâches avec des dépendances. Nous nous limiterons ici au modèle *flot de données synchrone*, dans lequel les taux de production et de consommation des données par les tâches sont fixes. Le modèle *flot de données* ne tient pas compte du temps. Nous rajoutons donc aux tâches des informations sur leurs dates de début et leurs durées d'exécution, ce qui constitue le modèle *flot de données temporisé*.

## 3.1. Le modèle *flot de données*

Le modèle *flot de données* [8] est *orienté données*. Un graphe *flots de données* est un graphe dirigé $G = (V, E)$ où les nœuds, dans l'ensemble $V$, représentent des calculs, en l'occurrence, des *effets audio*. Les arcs du graphe, dans $E \subset V \times V$, représentent le chemin des données et connectent les nœuds ensemble via des *ports d'entrée et de sortie*. Si $v_1$ et $v_2$ sont des nœuds, on pourra noter une arête $v_1 \rightarrow v_2$. Les données sont des séquences de *jetons* et un nœud s'exécute quand il y a suffisamment de *jetons* dans ses entrées, comme montré sur la Figure 3. Formellement, une fonction $\mu : E \rightarrow \mathbb{N} \times \mathbb{N}$ est ajoutée au graphe $G$, qui indique le nombre de jetons en entrée, et en sortie, d'une arête donnée. Dans le cas des traitements audio, le signal est échantillonné sur des intervalles périodiques. Les *jetons* du modèle *flot de données* correspondent à ces *échantillons* audio.

Le graphe *flot de données* est dit *synchrone* quand le nombre de jetons nécessaires à l'activation d'un nœud et le nombre de jetons produits en sortie, sont connus à l'avance.

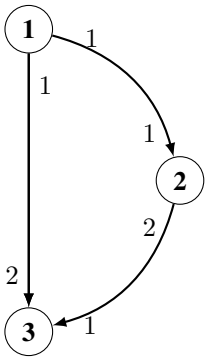



**Figure 3** : Un graphe simple *flot de données synchrone* avec trois nœuds, **1**, **2** and **3**. Les données circulent de **1** à **3**, de **1** à **2** et de **2** à **3**. **1** produit 1 échantillon par exécution, et **3** requiert 2 échantillons pour s'activer. **1** est un nœud *source*, **2** un *effet*, et **3** une *sortie*.

Les nœuds sans port d'entrée sont appelés *sources* ou *entrées* : ce sont les sources sonores, comme l'entrée micro, un fichier son, un synthétiseur. Les nœuds sans port de sortie sont appelés *sorties* ou *puits*, par exemple, la sortie vers la carte son et des haut-parleurs. Les nœuds qui ne sont ni des *entrées* ni des *sorties* sont appelés *effets*.

Le modèle *flot de données dynamique* [9] est un modèle plus riche que le modèle *synchrone*, où le nombre de jetons en entrée et en sortie n'est pas fixé et peut dépendre du nombre de jetons en entrée. Le graphe peut lui-même changer au cours de son exécution.

### 3.2. Le modèle *flot de donnée temporisé*

Un graphe *flot de données* ne décrit pas les instants où les nœuds du graphe s'activent mais seulement leur ordre partiel. Cependant, les graphes *flots de données* représentent souvent des traitements temps réel, par exemple les traitements audio temps réel. Pour un CD, dont le taux d'échantillonnage des morceaux est de 44.1 kHz, il y a un échantillon toutes les 22,6 $\mu s$. Si on donne des dates d'activation aux *sources*, que l'on connaît un temps d'exécution dans le pire des cas ou en moyenne de tous les nœuds, on peut en déduire des temps de déclenchement de chacun des nœuds du graphe. Si par ailleurs les *sorties* ont des échéances, cela permet de vérifier que les échéances sont respectées dans le pire des cas ou en moyenne.

On ajoute donc des dates de déclenchement d'exécution pour chaque *source* $v$, $s_1^v, \ldots, s_n^v$ avec $s_i^v < s_{i+1}^v$ et tous les nœuds du graphe reçoivent un temps d'exécution dans le cas moyen $M_v$, dans le pire cas, $W_v$ et un temps d'exécution actuel $T_v$. On adjoint aussi à chaque *sortie* $v$ des dates *d'échéances* $d_1^v, \ldots, d_n^v$, avec $d_i^v < d_{i+1}^v$, qui correspondent aux moments où la carte son devra avoir reçu un certain nombre d'échantillons. Les traitements audio sont périodiques donc on suppose que pour une source $v$, $s_2^v - s_1^v = s_{i+1}^v - s_i^v$, et pour une sortie $v$, $d_2^v - d_1^v = d_{i+1}^v - d_i^v$. Les taux d'échantillonnage en entrée $f_e^c$ et en sortie $f_e^p$ sur une arête $e = v_1 \to v_2$ et $\mu(e) = (j_1, j_2)$, qui correspondent au nombre de jetons produits et consommés sur une période sur l'arc, est $f_e^c = \frac{j_1}{T_{e_1}}$ and $f_e^p = \frac{j_2}{T_{e_2}}$.

Le temps d'exécution dans le pire cas (respectivement en temps moyen) est, pour un chemin $v_1 \to ... \to v_n$ du graphe, $\sum_1^n W_{v_i}$ (resp. $\sum_1^n M_{v_i}$).

## 4. QUALITÉ

La qualité d'un graphe audio est un concept relatif et subjectif : cette version du graphe sonne-t-elle mieux ou moins bien que telle autre ? Une première façon est de calculer la différence entre une version dégradée et une version considérée comme optimale, une mesure de qualité *a posteriori*. Ce n'est pas souhaitable lors d'une exécution temps réel ; cela reviendrait à calculer simultanément deux versions du graphe, alors que le but est d'alléger les calculs. Nous calculons plutôt une mesure de qualité *a priori* qui se base sur des paramètres des calculs en cours.

### 4.1. Quelques propriétés d'une mesure de qualité

La mesure de qualité doit aussi être une mesure *compositionnelle*, au sens où la qualité du graphe doit être calculable comme une fonction de la qualité de ses nœuds et de ses arêtes. Chaque nœud $v$ se voit attribuer une mesure de qualité $q_v \in [0, 1]$, 0 correspondant à la pire qualité, 1 à la meilleure.

La qualité $q_{v \to v'}$ sur un arc $v \to v'$ suit les propriétés suivantes :

**Associativité** $q_{v \to v'} = q_v \otimes q_{v'}$ où $\otimes$ est un opérateur associatif (et pas commutatif en général).

**Décroissance** $q_{v \to v'} \leq \min\{q_v, q_{v'}\}$, c'est-à-dire que la qualité n'augmente jamais sur un chemin.

De même, pour un graphe $\mathcal{G}$, la qualité $q_\mathcal{G}$ est :

$$q_\mathcal{G} = \bigotimes_{c \in \mathcal{C}} q_c$$

où $\mathcal{C}$ est l'ensemble des chemins de $\mathcal{G}$.

Elle n'est jamais supérieure à la qualité de chacune des chaînes du graphe. Par exemple, pour le graphe de la Figure 3 :

$$q_\mathcal{G} \leq \min\{q_{v_1 \to v_3}, q_{v_1 \to v_2 \to v_3}\}$$

### 4.2. Dégrader la qualité en pratique

On considère ici deux façons de dégrader la qualité du graphe : en insérant des nœuds pour sous-échantillonner des chemins du graphe, ou en modifiant des nœuds du graphe et en les remplaçant par une version de moindre qualité.

**Insertion de nœuds de rééchantillonnage.** Dans un graphe *flot de données*, les nœuds reçoivent des échantillons puis les traitent quand ils en ont reçu suffisamment. Par conséquent, si un nœud reçoit des échantillons moins souvent, il utilisera moins de temps de calcul. Changer le nombre

d'échantillons par unité de temps est une opération habituelle en traitement du signal, et est appelé *rééchantillonnage* : obtenir moins d'échantillons correspond à *sous-échantillonner*, tandis qu'en obtenir plus, à *sur-échantillonner*. Pour rééchantillonner, nous insérons des nœuds dans le graphe qui vont changer les taux de production ou de consommation des échantillons. Ces nœuds insérés sont des nœuds d'effet comme les autres ; ils peuvent interpoler des valeurs, copier des valeurs entre des tampons audio. Ils sont donc aussi munis d'une mesure de qualité et de caractéristiques temporelles, qui permettent de prendre en compte l'impact des calculs menant à la dégradation elle-même dans la décision de dégrader ou pas.

Si l'on insère un nœud sous-échantillonneur, tous les nœuds qui sont sur un chemin qui commence sur ce nœud vont opérer sur des flux sous-échantillonnés. Si le chemin se termine par une *sortie* qui dicte un taux d'échantillonnage particulier, il faut aussi insérer un nœud sur-échantillonneur.

**Remplacement d'un nœud.** Le traitement audio opéré par un nœud peut être remplacé par un autre traitement de qualité inférieure et de temps de calcul dans les cas pire et moyen inférieurs, c'est-à-dire, si on note $v'$ et $v''$ deux versions du nœud $v$, si $q_{v'} < q_{v''}$ alors $M_{v'} < M_{v''}$ et $W_{v'} < W_{v''}$. Par exemple, les nœuds de sous-échantillonnage peuvent utiliser un algorithme de plus ou moins bonne qualité : garder un échantillon sur deux ou bien, en plus de cela, utiliser un filtre passe-bas pour se débarrasser des artefacts fréquentiels dus à la première méthode. Un nœud de réverbération peut être implémenté en haute qualité avec une coût de calcul important à l'aide d'une convolution avec une réponse impulsionnelle d'une salle, ou avec des lignes de retards, ce qui entraîne une moins bonne qualité (une réverbération moins *réaliste*) mais un temps de calcul rapide.

### 4.3. Mesurer la qualité

On choisit comme mesure de qualité le taux d'échantillonnage : plus le taux d'échantillonnage est bas, plus la qualité est basse. Quand l'audio est envoyé trop tard au tampon de sortie, on entend un clic (due une discontinuité du signal, voir Section 2). Au contraire, si le flux audio a été sous-échantillonné, le tampon de sortie comportera des échantillons non-nuls en général. Nous partons donc du principe que sous-échantillonner permet d'obtenir une meilleure qualité que rater une échéance audio.

La qualité $q_v$ d'un nœud inséré d'échantillonnage $v$ est $q_v = \arctan(\alpha f)$ où $f$ est la fréquence d'échantillonnage. Le choix de la fonction $\arctan$ [2] et du coefficient $\alpha$ est permet de prendre en compte que d'un point de vue psychoacoustique [14], au delà d'une certain fréquence, aucune amélioration de qualité n'est perceptible par des humains. En effet, le système auditif de la plupart des êtres humains ne peut pas percevoir des fréquences au-delà de 20 kHz. Le théorème de Shannon indique donc que le taux d'échantillonnage doit être d'au moins le double, donc à peu près la fréquence des CD de 44.1 kHz. Cependant, le suréchantillonnage permet de mieux prendre en compte les erreurs de calcul et d'approximations lors des traitements audio et c'est pourquoi nous tenons quand même compte des fréquences supérieures à 44.1 kHz. On pourra prendre $\alpha = \frac{\tan(0.9)}{44.1 \cdot 10^3}$. On considère par par ailleurs que les autres nœuds sont de qualité optimale, c'est-à-dire, $q = 1$.

On considère une chaîne $\mathcal{C}$ de nœuds $v_1 \to \cdots \to v_n$ à dégrader. On note $v_{\text{sous}}$ et $v_{\text{sur}}$ respectivement le nœud de sous-échantillonnage et celui de sur-échantillonnage. Après insertion de ces nœuds, la chaîne devient : $v_{\text{sous}} \to v_1 \to \cdots \to v_n \to v_{\text{sur}}$. On peut prendre $\otimes$ le produit sur les réels, ou $\otimes = \min$. Comme une qualité $q$ est toujours dans $[0, 1]$, avec $\times$, la qualité ne peut qu'effectivement diminuer sur une chaîne.Pour $\min$, intuitivement, cela indique que la qualité du tout est celle du maillon le plus faible et que l'on peut dégrader beaucoup plus de nœuds pour respecter les échéances tout en conservant la même qualité.

Les traitements audio nécessitent en général de parcourir au moins une fois chacun des échantillons du tampon audio donc ont au moins une complexité linéaire. Le temps d'exécution de la chaîne est donc au moins divisé par le facteur de sous-échantillonnage $r$ (par exemple si l'on passe de 96 kHz à 48 kHz, $r = 2$). Le temps actuel de calcul (resp. moyen, dans le pire des cas) est majoré par :

$$T_{v_{\text{down}}} + \frac{1}{r} \sum_{i}^{n} T_{v_i} + T_{v_{\text{up}}}$$

## 5. CHOISIR LES NŒUDS À DÉGRADER

Le choix des nœuds à dégrader peut être considéré comme un problème d'optimisation. Ici, on s'intéresse au cas moyen, comme les SIM s'exécutent sur des plateformes grand public sans garanties temps-réel fortes. Le raisonnement serait le même dans le cas le pire.

Soit $\mathcal{G}$ est un graphe audio composés des nœuds $v_1, \ldots, v_n$ et $q_{\mathcal{G}}$ sa mesure de qualité associée. L'échéance d'une tâche est notée $d_v$ ; on peut l'obtenir grâce aux dépendances entre tâches, aux dates d'activation des sources et à leur temps d'exécution en moyenne. On suppose donné $\tau : [0, 1] \to \mathbb{R}^+$ qui décrit le temps d'exécution moyen d'une tâche en fonction de sa qualité.

Le problème d'optimisation sous contraintes peut ainsi s'écrire :

*maximiser $q_{\mathcal{G}}$ sous les contraintes :*

$$\tau(v_1) \leq d_{v_1}, \ldots, \tau(v_n) \leq d_{v_n}$$

En supposant que les qualités ont des valeurs dans un ensemble continue, on peut essayer des algorithmes standards d'optimisation pour résoudre le problème. On sait aussi que $\tau(n) = O(n)$ comme indiqué en section 4.3.

2 . $\arctan$ admet la droite $y = 1$ comme asymptote en $+\infty$, ce qui permet de modéliser que l'augmentation de la qualité est de moins en moins grande et ne dépasse pas un palier. D'autres fonctions croissantes avec une asymptote horizontale en $+\infty$ pourraient faire l'affaire.

Cependant, le temps de calcul pour la recherche de solutions exactes n'est pas raisonnable s'il doit être fait au moment de l'exécution du graphe audio, mais est beaucoup plus pertinent pour pré-calculer des version dégradées du graphe. Ce calcul hors-ligne de versions dégradées fera l'objet d'un travail ultérieur. Ici, nous nous attachons plutôt à utiliser des heuristiques simples et rapides à calculer lors de l'exécution du graphe audio.

## Exécution du graphe audio

Les tâches sont des tâches dépendantes et les dépendances entre tâches sont données par le graphe audio. On peut trouver un ordonnancement de ces tâches en effectuant un tri topologique du graphe (ce qui signifie que l'on suppose que le graphe audio est acyclique). Avant d'exécuter chaque tâche, on estime le temps d'exécution qui reste avant la fin de l'exécution du graphe, en utilisant le temps d'exécution dans le cas moyen des nœuds. Il faut ensuite vérifier si le temps restant estimé est plus petit que le temps restant avant l'échéance de la procédure audio qui envoie le signal à la carte son. Le temps d'exécution moyen du nœud est ensuite mis à jour.

Deux situations de surcharge du processeur peuvent se produire, une surcharge du processeur *transitoire*, ou bien une surcharge *permanente*. On dit que le processeur est surchargé de manière permanente quand le processeur est surchargé pendant plus de $k$ cycles audio. La façon dont est déterminé $k$ est hors-sujet dans le cadre de cet article.

**En cas de surcharge *transitoire*.** Étant donnée une chaîne de nœuds de traitement, il est préférable de dégrader les nœuds à la fin de la chaîne plutôt qu'au début, comme la qualité n'augmente jamais sur une chaîne (voir Section 4.1).

Une autre heuristique est de minimiser le nombre de nœuds de rééchantillonnage ajoutés, tout en maximisant le nombre de nœuds dégradés, pour minimiser le temps supplémentaire de calcul dû aux nœuds ajoutés. Cela implique que le nombre de chemins à dégrader doit être minimisé, et c'est pour cela qu'on choisit d'explorer le graphe branche par branche.

A chaque cycle d'exécution du graphe audio, l'algorithme 1 vérifie le temps restant à exécuter avant l'échéance et le compare au temps restant estimé d'exécution. S'il estime qu'il ne reste pas suffisamment de temps, on doit choisir parmi les nœuds restant à exécuter ceux à dégrader, comme décrit dans l'algorithme 2. On part d'un des nœuds *sortie*, on traverse le graphe en arrière jusqu'à ce qu'on a dégradé suffisamment pour que l'estimation du temps restant passe sous le temps restant donné par l'échéance. On peut explorer d'autres branches de même si ce n'est pas suffisant. Finalement, on ajoute les nœuds de sous-échantillonnage et de sur-échantillonnage au début et à la fin de ces branches du graphe.

Cependant, explorer successivement des branches, choisir le nœud à partir duquel dégrader, peut être encore trop coûteux en temps de calcul. Au lieu de l'algorithme 2, on peut plus simplement dégrader directement tous les nœuds restants du graphe, en insérant au début de chaque branche restante et à la fin les nœuds rééchantillonneurs.

**Algorithm 1** Exécution du graphe pendant un cycle, avec éventuelle dégradation.

```
Require : S a schedule, G an audio graph with associated
  execution times, d deadline
  while schedule is not empty do
    node ←pop_first(schedule)
    UPDATE(expectedRemainingTime)
    if expectedRemainingTime ≥ 0 then
      CHOOSENODES(G, deadline, expectedRemainingTime)
    end if
    samples ←GETINCOMINGSAMPLES
    if node.firstToDegrade then
      DOWNSAMPLE(samples)
    end if
    outBuffers ←NODE(samples)
    if node.lastToDegrade then
      UPSAMPLE(outBuffers)
    end if
    update performanceCounters
    node.visited ←true
  end while
```

**En cas de surcharge *permanente*.** Dans le cas d'une détection d'une surcharge permanente du processeur, une version dégradée calculée précédemment peut être utilisée, sans avoir besoin de la calculer à nouveau. Les versions dégradées calculées hors-ligne pourraient être utilisées ici.

## 6. ÉTUDE EXPÉRIMENTALE

Les algorithmes de dégradation en ligne ont été implémentés dans un prototype en Rust. Le rééchantillonnage est effectué avec libsamplerate [3] . Cette bibliothèque *opensource* permet de rééchantillonner avec des ratios arbitraires, compris entre un sous-échantillonnage par 256, jusqu'à un sur-échantillonnage par 256. Le taux d'échantillonnage peut être changé en temps réel. La bibliothèque fournit cinq rééchantillonneurs, *best, medium, fastest quality sinc* rééchantilloneurs (comme dans [16]), *zero order hold* où les valeurs interpolées sont égales à la dernière valeur, et un *rééchantillonneur linéaire*. Ils sont ici rangés en ordre décroissant de qualité.

On considère deux formes extrêmes de graphes audio, l'un où tous les effets sont connectés à un même nœud de sortie, sur la Figure 4a, et l'autre avec une seule branche, sur la Figure 4b. Les effets utilisés sont des oscillateurs sinusoïdaux, des modulateurs par une fréquence sinusoïdale, et des mixeurs.

Les expériences ont été effectuées sur un Mac Book Pro avec un processeur Intel Core i7 à 2.6 GHz, muni de 8

3. http://www.mega-nerd.com/SRC/

**Algorithm 2** Comment choisir les nœuds à dégrader.

```
function CHOOSENODES(graph, budget, expectedRemainingTime)
    expectedDegradedTime ←expectedRemainingTime
    while budget - expectedDegradedTime ≤ 0 do
        currentNode ←LASTNODE(graph)
        LASTNODE(graph).lastToDegrade ← true
        do
            UPDATE(expecteDegradedTime)
            parentNodes ←PARENTS(currentNode)
            currentNode ←
                FIRSTNOTVISITED(parentNodes)
            currentNode.visited ← true
            expectedDegradedTime ← expectedDegradedTime − EXPECTEDTIME(currentNode) + DEGRADEDTIME(currentNode)
        while ¬ currentNode.visited ∧
        currentNode.firstToDegrade ← true
    end while
end function
```

Gb de Ram. Les graphes ont exécutés pendant 5 secondes chacun.

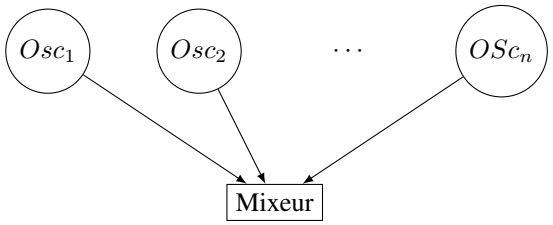
Sur la Figure 5, on montre comment l'algorithme de dégradation utilisant l'heuristique de dégradation de tous les nœuds restants se comporte quand on passe de $2000$ à $3000$ modulateurs, pour le graphe de la Figure 4b. La ressource temporelle restante est le temps qui reste avant l'échéance après que tous les traitements audio ont été effectués dans un cycle. Si elle est négative, cela signifie que l'échéance a été ratée et que le nœud de calcul est en retard. Pour un graphe avec $2000$ modulateurs, même si l'ordonnanceur détecte qu'il doit parfois dégrader, la ressource temporelle est suffisante pour ne pas rater d'échéance en général. Pour le graphe avec $3000$ modulateurs, l'algorithme de dégradation empêche bien en pratique de rater des échéances, et donc d'entendre des clics audio.

Nous avons aussi mesuré les *frais* causés par l'ordonnanceur, c'est-à-dire le temps supplémentaire pris par l'ordonnanceur pour trouver les nœuds à dégrader. Au plus, nous avons trouvé que ce temps supplémentaire s'élevait à $50\mu s$, c'est-à-dire à $1,25\%$ de l'échéance de $4000\mu s$, pour $2000$ nœuds. Cependant, la complexité dans le pire des cas de l'algorithme est linéaire en le nombre de nœuds.

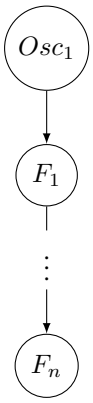
## 7. CONCLUSION ET PERSPECTIVES

Nous avons mis au point un algorithme en ligne de dégradation de nœuds dans un graphe flot de données dédié au son, cherchant un compromis entre qualité sonore dépendant du taux d'échantillonnage sur des branches du graphe et respect des échéances induites par la procédure d'entrée/sortie audio. En cas de surcharge permanente du processeur, une version dégradée pré-calculée du graphe pourrait être utilisée à la volée.

Nous souhaitons dans la suite de notre travail explorer




(a) Graphe audio plat avec $n$ oscillateurs.




(b) Audio graphe chainé avec un oscillateur suivi de $n$ modulateurs.

**Figure 4** : Les graphes utilisés pour les expériences.

plus en détails le calcul de versions dégradées du graphe hors-ligne. L'algorithme en ligne utilise des heuristiques pour choisir les nœuds à dégrader, au lieu de pré-calculer exactement des versions dégradées du graphe pour différents niveaux de qualité. Les versions dégradées pré-calculées du graphe audio qui peuvent être stockées sont en nombre limité. Les surcharges transitoires du processeur peuvent arriver en pleine exécution du graphe audio, et les versions pré-calculées ne sont pas forcément adaptés car elles peuvent exiger de dégrader certains nœuds qui ont pourtant déjà été exécutées en version normale lors du cycle en cours.

Nous voulons aussi évaluer d'autres heuristiques, sur un plus grand nombre de graphes, par exemple des graphes générés aléatoirement, et tester nos algorithmes de dégradation sur de vraies pièces musicales et dans des SIMs existants ; nous avons commencé à étudier la possibilité de les ajouter dans Puredata dans le code source du SIM. Nous pourrions aussi générer directement des versions *optimisés* de patchs Puredata ou Max, ou de programmes Faust [12], en ayant inséré des sous-échantillonneurs dans le graphe audio décrit.

## 8. REMERCIEMENTS

Ce travail a débuté lors d'une visite dans l'équipe du professeur Christoph Kirsch de l'université de Salzbourg en Autriche, que je souhaite remercier pour ses conseils

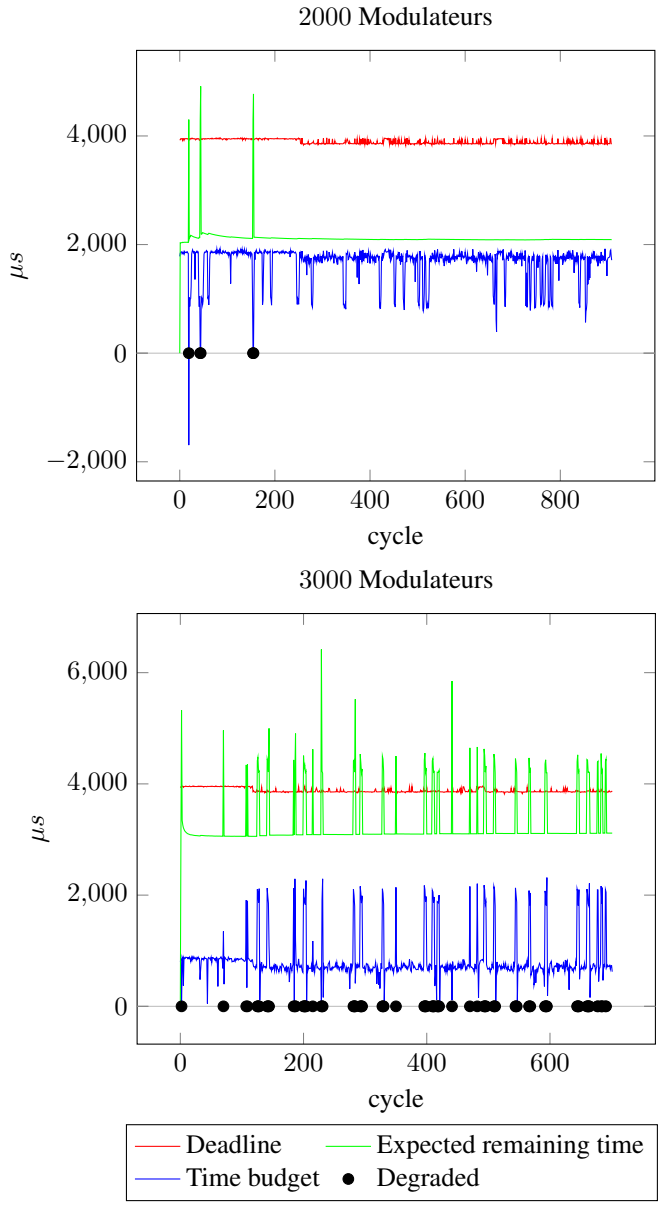



**Figure 5** : Les résultats pour le graphe chainé de la Figure 4b.

avisés. Je remercie aussi mes encadrants de thèse, Florent Jacquemard et Jean-Louis Giavitto, pour leurs conseils.

## 9. REFERENCES

[1] Saamer Akhshabi, Ali C Begen, and Constantine Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 157–168. ACM, 2011.

[2] Karl-Erik Årzén, Vanessa Romero Segovia, Stefan Schorr, and Gerhard Fohler. Adaptive resource management made real. In *3rd Workshop on Adaptive and Reconfigurable Embedded Systems*, 2011.

[3] T. Blechmann. Supernova-a multiprocessor aware real-time audio synthesis engine for supercollider. Master's thesis, Vienna University of Technology, 2011.

[4] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

[5] Pierre Donat-Bouillud, Jean-Louis Giavitto, Arshia Cont, Nicolas Schmidt, and Yann Orlarey. Embedding native audio-processing in a score following system with quasi sample accuracy. In *ICMC 2016-42th International Computer Music Conference*, 2016.

[6] José Echeveste, Arshia Cont, Jean-Louis Giavitto, and Florent Jacquemard. Operational semantics of a domain specific language for real time musician–computer interaction. *Discrete Event Dynamic Systems*, 23.

[7] Christoph M Kirsch. Principles of real-time programming. In *International Workshop on Embedded Software*, pages 61–75. Springer, 2002.

[8] Edward A Lee and David G Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75.

[9] Edward A Lee and Thomas M Parks. Dataflow process networks. *Proceedings of the IEEE*, 83.

[10] Jane WS Liu, Kwei-Jay Lin, Wei Kuan Shih, Albert Chuang-shi Yu, Jen-Yao Chung, and Wei Zhao. *Algorithms for scheduling imprecise computations*. Springer, 1991.

[11] James McCartney. Supercollider : a new real time synthesis language. In *Proceedings of the International Computer Music Conference*, 1996.

[12] Yann Orlarey, Dominique Fober, and Stephane Letz. Syntactical and semantical aspects of faust. *Soft Computing*, 8(9) :623–632, 2004.

[13] M. Puckette. Using pd as a score language. In *Proc. Int. Computer Music Conf.*, pages 184–187, September 2002.

[14] Stuart Rosen and Peter Howell. *Signals and systems for speech and hearing*, volume 29. Brill, 2011.

[15] Robert Rowe. *Interactive Music Systems : Machine Listening and Composing*. AAAI Press, 1993.

[16] Julius O Smith and Phil Gossett. A flexible sampling-rate conversion method. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'84.*, volume 9, pages 112–115. IEEE, 1984.

[17] Swagath Venkataramani, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Computing approximately, and efficiently. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 748–751. EDA Consortium, 2015.

[18] G. Wang. *The ChucK audio programming language." A strongly-timed and on-the-fly environ/mentality"*. PhD thesis, Princeton University, 2009.

[19] Zeyuan Allen Zhu, Sasa Misailovic, Jonathan A Kelner, and Martin Rinard. Randomized accuracy-aware program transformations for efficient approximate computations. In *ACM SIGPLAN Notices*, volume 47, pages 441–454. ACM, 2012.

[20] David Zicarelli. How I learned to love a program that does nothing. *Comput. Music J.*, 26(4) :44–51, 2002.

# SYNCHRONISATION DE DONNÉES INTER-PROCESSUS DANS LES APPLICATIONS AUDIO TEMPS RÉEL : QU'EST-CE QUI DÉBLOQUE ?

*Thibaut Carpentier*
CNRS – Ircam – Sorbonne Université – Ministère de la Culture
Sciences et Technologies de la Musique et du Son (UMR 9912 STMS)
1, place Igor Stravinsky, 75004 Paris
thibaut.carpentier@ircam.fr

**RÉSUMÉ**

Cet article expose des considérations pragmatiques pour le développement d'applications audio temps réel. Il passe en revue plusieurs concepts fondamentaux qui doivent être pris en compte lors de l'élaboration de telles applications. En particulier, les applications audio sont intrinsèquement *multi-thread* et asynchrones, et la synchronisation de ressources partagées entre plusieurs processus concurrents doit donc faire l'objet d'une attention minutieuse. Les concepts exposés ici ne sont pas nouveaux, toutefois leur mise en pratique, dans un contexte concret de production, demeure un défi majeur pour tous les développeurs audio.

Enfin, la thèse soutenue dans cette étude est que les mécanismes de synchronisation dits *lock-free* ne sont généralement pas indispensables et doivent être évités autant que faire se peut. Une approche par *mutex* non-bloquant est proposée comme substitut simple et satisfaisant.

## 1. INTRODUCTION

### 1.1. Anatomie d'une application audio temps réel

L'expression "application audio temps réel" se réfère, dans son acception la plus répandue, à des logiciels [1] qui génèrent, analysent, ou transforment des signaux audio-numériques, en un temps suffisamment court pour que la latence induite soit auditivement imperceptible (de l'ordre de quelques millisecondes). Ces logiciels sont majoritairement mis en œuvre sur des systèmes d'exploitation "conventionnels" tels que macOS, Linux, ou Windows. Il est à noter que lesdits systèmes d'exploitation (*Operating Systems*, OS) ne sont pas des systèmes temps réel au sens strict (*hard real-time*) : le temps maximum entre un stimulus d'entrée et une réponse de sortie ne peut être garanti de façon déterministe. Lorsque ce temps excède les contraintes du temps réel, la "qualité du service" se dégrade, mais le système reste opérationnel et continue de fonctionner ; on parle dans ce cas de temps réel souple (*soft real-time*).

1 . Les hardwares dédiés (DSP, FPGA, etc.) sortent du cadre de cet article.

Les logiciels audio se présentent généralement sous forme d'applications autonomes (*standalones*) ou de *plugins* insérables dans des environnements hôtes, tels que des stations de travail audionumériques (*Digital Audio Workstations*, DAW) ou des *frameworks* multimédia interactifs tels que Max [48], PureData [49], SuperCollider [37], Unity3D, etc. Pour des raisons de performances et d'inter-opérabilité avec leurs environnements hôtes, la vaste majorité des applications audio temps réel est codée en langage `C` ou `C++`. Ce dernier servira d'exemple pour le présent article ; toutefois la plupart des concepts exposés sont transposables à d'autres langages.

### 1.2. Un contexte intrinsèquement concurrent

Les OS et les applications hôtes considérés ici s'appuient fortement sur des paradigmes de programmation concurrente : pour offrir une interaction riche avec l'environnement externe, et pour tirer profit des systèmes multi-cœurs, plusieurs processus (ou *threads*) sont exécutés en parallèle. Les applications audio doivent donc s'intégrer dans de tels contextes intrinsèquement concurrents. Typiquement plusieurs *threads* sont exécutés "simultanément", afin de gérer le traitement des échantillons audio, les flux de données entrantes (instruments MIDI, capteurs, vidéo, etc.), ou des interactions de l'utilisateur (clavier, souris, etc.). Évidemment, la concurrence ici en jeu est de nature *coopérative*, c'est-à-dire que les différents processus ne sont pas isolés : pour le bon fonctionnement de l'application, ils doivent interagir, notamment en s'échangeant des ressources.

Une conséquence induite par la programmation concurrente est le phénomène dit "d'un indéterminisme d'exécution" : l'ordre d'exécution des instructions n'est plus séquentiel, mais il est soumis à une politique d'ordonnancement (*scheduling*) qui est déterminée par le noyau du système d'exploitation et par les règles de l'environnement hôte. Il en résulte que l'accès aux ressources partagées entre les processus doit être protégé : le développeur doit assurer leur intégrité, typiquement en mettant en œuvre des mécanismes de synchronisation (qui seront discutés plus loin).

Thibaut Carpentier, *Synchronisation de données inter-processus dans les applications audio temps réel : qu'est-ce qui débloque ?*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 35–44, 2018.

### 1.3. Ordonnancement

Nous considérons le comportement de l'environnement temps réel Max comme un exemple canonique. Il s'agit certes d'un cas particulier, mais celui-ci tout en étant simple s'avère représentatif de la majorité des environnements audio. Dans Max, les processeurs audio peuvent être pilotés par des événements (messages, clavier MIDI, clic souris, etc.). Comparativement à la cadence audio (*audio-rate*, typiquement 48000 échantillons par seconde), ces événements surviennent de façon sporadique ; ils sont dits *control-rate*. Les événements peuvent être à haute priorité (i.e. nécessitant une grande précision temporelle, par exemple métronome, messages MIDI) ou à basse priorité (interaction avec souris/clavier, ou plus généralement avec des interfaces graphiques). Les événements à basse priorité sont toujours traités dans le *thread* principal (*main thread*) de l'application ; selon les réglages de Max (paramètre *overdrive*) un second *thread* peut être activé pour gérer les événements à haute priorité (si l'*overdrive* est désactivé, tous les événements transitent dans le *thread* principal). Les événements à haute priorité peuvent donc potentiellement interrompre le *thread* principal, et s'exécuter avant les événements à basse priorité, selon un ordre non déterministe. De la même manière, dans une DAW, un *plugin* pourra recevoir des événements depuis le *thread* principal (interface graphique) et depuis un *thread* séparé pour l'automation [2].

Le *scheduler* de Max peut gérer la priorité des événements et les affecter au *thread* haute ou basse priorité (voire à d'autres *threads*). Tous les *threads* restent toutefois tributaires de l'ordonnancement du noyau de l'OS qui régule l'accès des processus au CPU. Enfin, les échantillons audio sont traités dans un *thread* spécifique : le *thread* audio.

### 1.4. Le *thread* audio

Une application audionumérique délivre un flux d'échantillons audio vers le convertisseur numérique/analogique (*digital to analog converter*, DAC) de l'interface audio (carte son). Les échantillons sont produits à une cadence fixe, imposée par la fréquence d'échantillonnage de l'interface. Sur les OS conventionnels, les échantillons ne sont pas délivrés un par un, mais par bloc (*buffer*) typiquement de 32 à 512 échantillons. Les *buffers* sont convoyés vers le driver audio par l'intermédiaire d'une couche de l'OS, e.g. CoreAudio sous macOS, ALSA sous Linux, ASIO sous Windows. Pour ce faire, l'application audio reçoit périodiquement un appel (*callback*) de la couche audio. Ce *callback* survient toutes les $N$ millisecondes ($N$=1000×buffersize/samplerate) ; il fournit le(s) *buffer(s)* des échantillons d'entrée, et l'application audio doit remplir les *buffers* de sortie. Ce mécanisme de traitement par blocs introduit une certaine latence d'entrée/sortie, proportionnelle à la taille de bloc, et généralement ajustable par l'utilisateur final (dans l'application hôte).

Le *callback* survient dans un *thread* spécifique, appelé *thread* audio et parfois qualifié de "temps réel". Sur les OS traditionnels, le *thread* audio est en fait un *thread* à "temps borné" (*time constraint policy*), généralement préemptable, et qui doit, pour garantir un fonctionnement correct, exécuter sa tâche avant une échéance donnée (*deadline*). Typiquement, un tel *thread* indique à l'OS sa périodicité nominale $N$, sa durée nominale d'exécution, et sa contrainte maximale (durée minimale d'exécution). Ces données sont comme une promesse faite à l'OS, et sur cette base, le *scheduler* a la responsabilité d'allouer des ressources et d'ordonnancer les différents processus de sorte à servir au mieux les différents *threads* courants.

Si l'échéance du *thread* audio n'est pas satisfaite, des artefacts (*clicks, glitches*) sont audibles. Plusieurs phénomènes peuvent en être la cause : *a*) un défaut du *scheduler* de l'OS (rare), *b*) le code de traitement audio n'est pas assez rapide pour s'exécuter dans le temps imparti (ne "tient pas" le temps réel), ou *c*) le code utilisé n'a pas un temps d'exécution déterministe ou borné. Le problème *b*) est lié à l'efficacité du code et/ou des algorithmes employés ; des techniques d'optimisation des performances peuvent alors être mises en jeu, mais ceci sort du cadre de cet article. Pour éviter le phénomène *c*), on trouve [3] un certain nombre de règles de bonne conduite à respecter dans le *thread* audio :

- éviter les opérations (potentiellement) bloquantes telles que l'acquisition de verrous (*locks*), la lecture de données sur disque ou sur une *socket* réseau, etc. ;
- proscrire les allocations mémoire (`std::malloc`, `std::free`, opérateurs `new` ou `delete`, etc.) car elles peuvent être bloquantes ou, selon leur implémentation, basée sur des algorithmes à temps non déterministe. La mémoire utilisée dans le *thread* audio doit être pré-allouée à l'avance dans des *threads* non-critiques. Aussi, la réservation de zones mémoire (*memory pool*) est fréquemment recommandée ;
- désactiver si besoin le ramasse-miette (*garbage collector*) ; ceci ne concerne pas `C++`, mais peut affecter d'autres langages tels que `Java` ou `C#` ;
- ne pas appeler de code tiers si son comportement (temps d'exécution) n'est pas connu et prédictible ; ceci concerne également tout appel à des fonctions de l'OS qui, sauf preuve du contraire, doivent être considérées comme bloquantes. De même, tout appel à `Objective-C` ou `Swift` est interdit puisque la résolution des messages durant l'exécution (*dynamic dispatch*) peut déclencher des verrous. Signalons ici qu'il existe des outils d'analyse statique de code permettant de détecter les appels à des fonctions non fiables (voir par exemple [38]) ;
- dans le choix des algorithmes employés, ne pas tenir compte de leur complexité en moyenne (ni leur complexité amortie), mais de leur complexité dans le cas le plus défavorable (*worst-case complexity*). De façon générale, lisser la charge de travail (sur plusieurs *callbacks*), et éviter tout algorithme pouvant manifester des pics de calcul (CPU *spikes*) ;

2 . Les mécanismes peuvent varier sensiblement d'une DAW à l'autre.

3 . essentiellement sur des sites web dédiés au développement `C++`, mais on pourra aussi se référer à [19, 12, 47, 51].

### 1.5. Problématique

Ainsi, les processeurs audio temps réel doivent s'inscrire dans des contextes *multi-thread*, et échanger de façon asynchrone des données avec le *callback* audio qui est soumis à d'importantes contraintes pour s'exécuter en un temps borné. Comment garantir alors un accès *thread-safe* aux ressources partagées par le *thread* audio ? Cette problématique n'est pas nouvelle : les paradigmes de programmation concurrente et les mécanismes de synchronisation inter-processus sont, d'un point de vue théorique, bien maîtrisés et largement discutés dans la littérature [50, 46, 4]. Toutefois leur mise en œuvre pratique, notamment dans un contexte de développement d'applications audio temps réel, demeure un défi et une source de questionnement pour les développeurs.

## 2. MÉCANISMES DE SYNCHRONISATION DE DONNÉES ENTRE *THREADS*

### 2.1. Le modèle de mémoire en C++

C et C++ sont spécifiés comme des langages séquentiels à *thread* unique (*single-threaded languages*), la gestion des *threads* étant reléguée dans des bibliothèques de l'OS (par exemple `pthread` sur les systèmes Posix), ou des APIs dédiées (par exemple `OpenMP` [4]). En conséquence, les compilateurs n'ont pour l'essentiel pas connaissance des *threads*, et ils sont autorisés à effectuer un certain nombre de transformations du code, en particulier ré-ordonner l'affectation de variables indépendantes tant que ces transformations préservent la consistance du programme séquentiel *single-thread*. Dans des contextes *mulit-thread*, l'accès aux données peut engendrer des problèmes de condition de concurrence (*race condition* ou *data race*). Le modèle de mémoire (*memory model*, voir [10, 1, 11, 8, 57]) du langage décrit la sémantique des variables partagées, i.e. qu'il spécifie la ou les valeurs qu'elles peuvent retourner dans un programme *mulit-thread* [5]. Dans le cas de C++, le *memory model* stipule que deux actions (*threads*) sont en conflit si elles accèdent au même emplacement mémoire (e.g. à la même variable) et que l'une au moins des actions est une écriture. Par exemple lire une variable $x$ depuis un *thread* $A$, alors qu'un *thread* $B$ écrit dans $x$ constitue une *data race*. C++ garantit l'exécution séquentielle correcte (*sequential consistency* [33, 1]) des programmes sans *race condition* ; en présence de *data race*, la sémantique et le comportement ne sont pas définis (*undefined behaviour* : tout et n'importe quoi peut alors se produire, éventuellement un crash du programme [9]). Il est à noter que le *memory model* de C++ est sensiblement différent, par exemple, de Java ou C# [36, 44, 1]. Par ailleurs, signalons qu'il existe également un *memory model* au niveau *hardware* car les processeurs, à l'instar des compilateurs, peuvent ré-ordonner les instructions. Évidemment les modèles *software* et *hardware* se doivent d'être compatibles.

Pour permettre l'accès concurrent à des variables normales conformément au *memory model*, c'est-à-dire en évitant les *data races*, plusieurs primitives de synchronisation sont disponibles.

### 2.2. Verrous de protection

#### *2.2.1. Sections critiques*

Le mécanisme le plus communément usité pour éviter les *race conditions* est d'employer des verrous de protection (*locks*) qui permettent de garantir un accès mutuellement exclusif à une donnée (ou structure de données). Les primitives de synchronisation sont appelées *mutex* (*mutual exclusion*). Un *thread* peut verrouiller le *mutex* pendant qu'il accède à la donnée partagée ; on dira qu'il *possède* le *mutex*. Tant qu'il possède le *mutex*, tous les autres *threads* sont bloqués (i.e. mis en attente) s'ils essayent d'accéder à la ressource. La zone de code protégée par le *mutex* est désignée "section critique". Ce mécanisme garantit que tous les *threads* voient toujours une version consistante de la donnée.

Depuis `C++11`, la bibliothèque standard (*Standard Template Library*, STL) fournit plusieurs variantes de *mutex* (`std::mutex`, `std::recursive_mutex`, etc.) ainsi que d'utiles wrappers (par exemple `std::lock_guard`) qui facilitent leur mise en œuvre selon un idiome RAII [6]. Avec `C++14` et `C++17`, la STL s'enrichit encore avec les ajouts de `std::shared_mutex`, `std::shared_lock`, `std::scoped_lock`, etc. L'accès à ces fonctionnalités dans la STL simplifie grandement la tâche des développeurs, les affranchissant des différences d'implémentation entre OS [7].

#### *2.2.2. Inconvénients*

L'utilisation de *mutex* s'accompagne toutefois d'un certain nombre de dangers. En particulier, ils peuvent conduire à une situation d'inter-blocage (*deadlock*) lorsque deux processus concurrents s'attendent mutuellement, essayant d'acquérir deux *mutex* dans un ordre différent [8]. Il existe des recommandations et des algorithmes dédiés pour se prémunir de ce phénomène.

Dans un contexte audio temps réel, d'autres inconvénients des *mutex* sont encore à prendre en considération :

— Si une ressource est bloquée par un *mutex*, les autres *threads* ne peuvent savoir pendant combien de temps elle sera bloquée. En outre, l'acquisition d'un *mutex* est tributaire du *scheduling* opéré par le noyau de l'OS. Le temps d'acquisition d'un *mutex* n'est donc pas prédictible de façon déterministe.

4 . `www.openmp.org`

5 . Il est surprenant de noter que le *memory model multi-thread* n'apparaît qu'à partir de la révision `C++11`.

6 . Resource acquisition is initialization (RAII) : par exemple la classe `std::lock_guard` verrouille le *mutex* dès son constructeur, et le débloque dans son déstructeur donc dès qu'il disparaît du *scope* courant.

7 . Il existe en pratique plusieurs stratégies pour implémenter les *mutex* (e.g. *binary semaphores*, *spinlocks*, *ticket locks*) mais cela sort du cadre de cet article.

8 . Exemple : un *thread* `t1` acquiert un *mutex* `m1`, un *thread* `t2` acquiert un *mutex* `m2`, `t1` attend pour acquérir `m2`, `t2` attend pour acquérir `m1`.

— Les *mutex* peuvent provoquer un scénario d'inversion de priorité (*priority inversion*) : lorsqu'une tâche $H$ à haute priorité attend un *mutex* détenu par une tâche $B$ à basse priorité, $B$ peut se retrouver préemptée par une tierce tâche $M$ de priorité intermédiaire, inversant l'ordonnancement prévu ($M < H$) et bloquant l'exécution de $H$.
— Enfin les *mutex* peuvent générer un problème de performance dit *lock convoy* lorsque des tâches de même priorité tentent d'acquérir de façon répétée un *mutex* ; les *threads* ne sont pas bloqués, mais à chaque tentative avortée d'acquisition du *mutex*, le *thread* doit renoncer à son quantum auprès du *scheduler*, et il provoque une commutation de contexte (*context switch*). Ceci contribue à dégrader les performances du système.

Notons qu'il existe d'autres problèmes à prendre à compte lors de l'usage de *mutex* (e.g. statut d'un *thread* tué tandis qu'il détient un *mutex*) mais ceux-ci ne sont pas directement pertinents pour notre étude.

### 2.3. Les approches non-bloquantes

Pour pallier les inconvénients des verrous, des techniques de programmation non-bloquantes, généralement dites *lock-free*, ont été proposées. La notion de *lock-free* fait référence à l'accès *thread-safe* (i.e. exempt de *race conditions*) à des données partagées, sans le recours à des primitives telles que les *mutex*. On recense trois niveaux de stratégies non-bloquantes [25] :

— Programmation *wait-free* [31] : il s'agit de la garantie "la plus forte". Un système est dit sans attente (*wait-free*) si tous les processus sont garantis de toujours progresser ; aucun *thread* n'est entravé par un autre. Il est donc assuré que toute opération pourra se réaliser un nombre fini d'instructions. En pratique, l'implémentation d'algorithmes *wait-free* est très difficile et assez rare. Ils sont toutefois employés dans des contextes temps réel strict (*hard real-time*) [21].
— Programmation *lock-free* : les algorithmes *lock-free* garantissent qu'au moins l'un des *threads* pourra toujours progresser. En particulier, si l'un des *threads* est préemptés tandis qu'il travaille sur une ressource *lock-free*, il ne bloque pas les autres *threads* qui peuvent accéder à cette ressource et donc converger vers un résultat en un nombre fini d'instructions. Des trois approches non-bloquantes, le paradigme *lock-free* est incontestablement le plus répandu. Notons que tous les algorithmes *wait-free* sont nécessairement *lock-free*.
— Programmation *obstruction-free* [30] : il s'agit de la garantie "la plus faible". Dans ce paradigme, une opération concurrente peut être annulée et ré-essayée ultérieurement, au profit des autres *threads* en compétition. Un algorithme est donc *obstruction-free* si, à un instant donné, un *thread* "isolé" (tous les autres *threads* en concurrence étant suspendus) peut accomplir ses opérations en un nombre fini d'instructions. Cette approche permet d'éviter les situations d'inter-blocage et d'inversion de priorité, mais peut causer des phénomènes de famine (*livelock*) lorsque deux opérations mutuellement en compétition s'annulent l'une l'autre. Notons enfin que tous les algorithmes *lock-free* sont nécessairement *obstruction-free*.

### 2.4. Implémentation pratique des structures *lock-free*

La conception d'algorithmes *lock-free* se révèle très difficile, même d'un point de vue formel. En pratique, on développe plutôt des structures de données *lock-free* (voir paragraphe 2.4.2). Celles-ci sont fréquemment usitées ou recommandées dans le domaine des applications audio temps réel [23, 24, 35, 19, 54, 53, 6, 7, 18, 55]. L'implémentation de telles structures *lock-free* s'appuie sur des variables dites de synchronisation (par opposition aux variables normales) et qui permettent l'exécution d'opération atomiques.

#### *2.4.1. Opérations atomiques*

On dit d'une opération qu'elle est atomique si elle est indivisible, i.e. qu'elle ne peut être interrompue. Soit l'opération s'exécute en entier, soit elle ne s'exécute pas du tout, et il ne peut y avoir d'état intermédiaire.

Depuis `C++11`, des types atomiques `std::atomic` sont disponibles dans la STL. Toutes les opérations qu'ils permettent sont atomiques, et ce sont les seules opérations strictement atomiques offertes par le langage [9]. `std::atomic` offre un nombre restreint de méthodes telles que `load`, `store`, `fetch_add` et `compare_exchange`. Cette dernière primitive est fondamentale et elle est à la base de toutes les implémentations de structures *lock-free*. Comme son nom l'indique, elle réalise un *compare-and-swap* (aussi appelé *compare-and-exchange*) : cette méthode compare la valeur courante $x$ avec une valeur attendue $y$ ; en cas d'égalité, remplace la valeur courante par une nouvelle valeur désirée $z$ (opération *read-modify-write*) ; sinon, la valeur courante $x$ est chargée (*load*) dans $y$. L'atomicité de cet échange est rendue possible par des propriétés du processeur (*hardware feature*) et non de l'OS. Notons que ceci introduit nécessairement une synchronisation (barrière) au niveau CPU ; le traitement de variables atomiques est donc significativement plus lent que celui de variables normales [10].

9 . Les fonctionnalités de `std::atomic` sont peu ou prou équivalentes à la notion de `volatile` en langage `Java`. Notons en revanche que le mot-clé `volatile` en `C++` n'a aucun rapport ! En `C++`, `volatile` ne garantit pas l'atomicité, ne permet pas la synchronisation entre *threads* et n'empêche pas le ré-ordonnancement des instructions (ni au niveau du compilateur, ni au niveau hardware).

10 . Par souci d'exhaustivité, indiquons que les opérations sur `std::atomic` peuvent être paramétrées plus finement en spécifiant le mode d'ordonnancement mémoire (*memory ordering*). En effet, le memory model `C++` offre ici plusieurs alternatives (*relaxed memory ordering*, *acquire/release model*, *consume/release model*) qui permettent de contrôler au bas-niveau les barrières mémoires (*memory fences*) employées par le hardware [5]. On parle de *low-level atomics*. Ce contrôle de bas-niveau peut, en théorie, accroître les performances. Toutefois, ces modes de fonctionnement impliquent un assouplissement du memory model, et en particulier la consistance séquentielle du programme (voir paragraphe 2.1) n'est plus garantie. Sans consistance séquentielle, le code devient extrêmement difficile à écrire et pire encore à déboguer ; c'est pourquoi il est généralement déconseillé de s'y risquer, et nous ne nous étendrons pas

`std::atomic<T>` est une classe template dont le paramètre `T` peut prendre un type scalaire (`bool`, `char`, `int`, etc.) à l'exception des flottants (`float` ou `double`). Les scalaires englobant les pointeurs, `T` peut également être une adresse : `std::atomic<T*>`. Ici, il est important de noter que `T` ne peut être un object (`class`) ou un composé (`struct`, `union`, `array`) ; ceux-ci ne peuvent qu'être passés par leur adresse (i.e. par pointeur).

Enfin, signalons que `std::atomic` n'implique pas nécessairement une implémentation *lock-free*. Sur la plupart des processeurs "courants", `std::atomic<T>` est *lock-free* au moins pour les types `T` de 8 bits ; pour des types plus longs, il est possible que le processeur ne dispose pas du jeu d'instruction nécessaire et, de fait, `std::atomic<T>` sera implémenté de façon bloquante (au niveau du système) [11].

#### *2.4.2. De nombreuses structures existantes*

Les variables atomiques rendent possible le développement de structures de données *lock-free*, notamment des queues (FIFO), des piles (LIFO), ou des listes chaînées (*singly* ou *doubly linked lists*), etc. Les exemples d'implémentations ne manquent pas dans la littérature [56, 42, 23, 24, 50, 57, 25, 43, 2, 3, 30, 26]. Chacune de ces structures de données a des caractéristiques spécifiques, et le choix d'un *container* approprié dépend de l'application visée. Dans le cas de structures *lock-free*, ce choix est rendu plus difficile par l'existence de nombreuses variantes, fonction du contexte *multi-thread* : il faut en effet distinguer les *threads* qui écrivent dans le *container* (*producers*) de ceux qui y lisent (*consumers*). On trouve ainsi des déclinaisons *single producer-single consumer* (SPSC : cas le plus simple où un *thread* écrit et un autre lit), *multiple producers-multiple consumers* (MPMC : plusieurs *threads* concurrents peuvent écrire tandis que plusieurs *threads* peuvent lire), et les combinaisons SPMC et MPSC. On comprend aisément qu'une structure SPSC requiert moins d'efforts de synchronisation qu'une MPMC ; on peut donc supputer que cette dernière sera moins performante.

Les containers *lock-free* sont le plus souvent implémentés sous forme de *buffers* circulaires de taille fixe, de sorte à pré-allouer l'espace de stockage (statiquement ou durant l'initialisation) et à éviter les allocations dynamiques durant l'exécution.

### 2.5. Avantages et inconvénients des structures non-bloquantes

La principale motivation et l'avantage putatif de la programmation *lock-free* est la scalabilité, c'est-à-dire la capacité d'exploiter au mieux le parallélisme en augmentant le nombre d'opérations sur une structure de données tout en minimisant le temps d'attente de chacun des *threads*, et en évitant les problèmes de deadlocks et d'inversion de priorité.

Cependant cela s'accompagne d'un certain nombre de difficultés, voire d'inconvénients par rapport aux techniques bloquantes :

— La conception et le développement de structures *lock-free* est considérablement plus difficile que la programmation à base de *mutex*. Il est communément admis que cela doit être strictement réservé aux développeurs les plus aguerris [12]. Bjarne Stroustrup et Herb Sutter ont récemment publié un ensemble de recommandations pour l'usage pertinent et efficace du langage `C++` (Core Guidelines [13]) ; la première règle (CP.100) de la section *lock-free* stipule : "Don't use lock-free programming unless you absolutely have to. Reason : It's error-prone and requires expert level knowledge of language features, machine architecture, and data structures."

— En corollaire, les programmes *lock-free* sont très difficiles à maintenir et à débogguer, d'autant qu'il est compliqué sinon impossible de prouver formellement leur exactitude [34]. Par ailleurs, il est délicat d'évaluer les performances de la programmation non-bloquante et de démontrer ses bénéfices.

— Un autre inconvénient crucial des structures *lock-free* concerne la gestion de la mémoire et de la durée de vie des objets : comme expliqué au paragraphe 2.4.1, les containers *lock-free* de données "complexes" (composées) nécessitent l'usage de `std::atomic<T*>`. L'utilisation d'un pointeur nu `T*` contrevient à toutes les bonnes règles de codage en `C++` moderne, et délègue à l'appelant la gestion mémoire (allocation/libération) des éléments du container. Ceci augmente considérablement la complexité du code et le risque d'erreurs [14]. Pour "pallier" cela, les structures *lock-free* doivent généralement être accompagnées d'un dispositif ad-hoc de gestion de la mémoire, ce qui constitue en soi un défi considérable. Ont été proposés à cet effet : des *pools* d'objets [32], des ramasse-miettes avec compteur de références [22, 29, 27], des *hazard pointers* [40, 3], des approches d'accès optimistes [17], etc.

— Elles peuvent être soumises au problème dit d'échange d'états ABA [20, 21].

— Elles ne constituent pas un remède au problème de contention.

— Certaines implémentations font l'objet de brevets [52, 45].

sur ce sujet.

11 . La méthode `std::atomic<T>::is_lock_free()` permet de tester si l'implémentation est *lock-free* pour le type `T` et l'architecture courante. (Bizarrement, cette méthode est une fonction membre et non une fonction statique de classe.)

12 . Herb Sutter, éminent spécialiste du langage et président du comité de standardisation ISO `C++`, a donné de nombreux keynotes et tenu une colonne sur `www.drdobbs.com` au sujet de la programmation concurrente. Il explique en substance que la programmation *lock-free* est réellement maîtrisée par environ "fifty people in the world (and that's a generous idea)", car "writing lock-free code can confound anyone – even expert programmers" (Lock-Free Code : A False Sense of Security – publié en septembre 2008).

13 . `https://github.com/isocpp/CppCoreGuidelines`

14 . Tempérons un peu en signalant que des propositions concernant `std::atomic_shared_ptr` font l'objet de discussions pour `C++20`. Ceci pourrait grandement simplifier l'usage de structure de données atomiques.

## 3. UNE APPROCHE SOUS-ESTIMÉE : LES *MUTEX* NON-BLOQUANTS

Revenons à la problématique des applications audio temps réel. Nous l'avons dit dans la section 1, il est généralement inévitable de se confronter à la concurrence *multi-thread*, car le *callback* audio a besoin de communiquer avec un *thread* graphique, des *threads* haute priorité (interfaces MIDI, sockets réseau, etc.) ou basse priorité (accès de fichiers sur disque, etc.). Dans ce contexte concurrent, une queue d'événements est requise. Deux cas d'usage typiques se présentent, et ils couvrent une vaste majorité des applications audio :

— Queue MPSC : différents *threads* (GUI et/ou haute priorité) sont employés pour piloter un moteur audio vers qui ils envoient des messages. Le *callback* audio est dans ce cas le seul et unique *consumer*, et plusieurs *threads* sont *producers* (généralement en nombre assez restreint – deux ou guère plus).
— Queue SPMC : le *thread* audio analyse les signaux et renvoie des valeurs par exemple vers une interface graphique (e.g. spectroscope, mesure de niveaux RMS et vu-mètres, etc.). Dans ce cas le *callback* audio est l'unique *producer*, et les autres *threads* sont *consumers*.

Dans la plupart des usages courants, il est en outre légitime de considérer que le timing des événements n'est pas critique. Typiquement, une réactivité de l'ordre de quelques millisecondes (supérieur à la taille de bloc audio) est satisfaisante (e.g. pour changer les paramètres d'un processeur audio ou rafraîchir la visualisation d'une analyse).

Sous ces hypothèses, nous proposons d'utiliser une technique de synchronisation inter-processus basée sur des *mutex* non-bloquants. Cette technique n'est pas nouvelle, mais elle semble largement sous-estimée et inexploitée. Elle repose simplement sur la méthode `try_lock()` de `std::mutex` qui, comme son nom l'indique, tente d'acquérir le *mutex*. En cas d'échec (si le *mutex* est déjà verrouillé par ailleurs), la méthode retourne immédiatement, sans bloquer. En cas de succès, elle obtient la lock et les autres *threads* ne pourront entrer dans la section critique. La méthode est susceptible de retourner des faux négatifs.

Par exemple, dans le scénario de queue MPSC évoqué, l'écriture des événements par les *threads producers* est protégée par `std::mutex::lock()`, tandis que la lecture de la queue dans le *thread* audio se contente d'un `try_lock`. Si le `try_lock` échoue, l'opération sera re-tentée ultérieurement, par exemple au prochain bloc audio [15]. Le scénario de queue SPMC peut être traité de façon analogue.

Il est en théorie possible que le `try_lock` échoue systématiquement et que les paramètres dans le *thread* audio ne soient donc jamais mis à jour. Nous spéculons que ce phénomène n'a, en pratique, que peu de risque de se produire : le nombre de *threads* concurrents est très restreint, les événements ne surviennent que sporadiquement, et les sections critiques sont très "courtes" (il s'agit de lire ou écrire un élément – e.g. message MIDI ou OSC – dans la queue, ce qui ne requiert que quelques instructions). Pour les mêmes raisons, on peut espérer que les *threads producers* pâtiront peu des blocages de *mutex*, de sorte que le programme demeure réactif.

La technique de *mutex* non-bloquants ici proposée permet donc une synchronisation *thread-safe* qui tire profit de la grande simplicité des *mutex* (l'effort de codage est minimal), sans risquer d'inversion de priorité (le *callback* audio n'étant jamais bloqué). Elle garantit la progression *obstruction-free* du *thread* audio [41].

### 3.1. Évaluation des performances

La notion de "progression garantie" est indépendante des performances du sytème. Il convient donc d'évaluer l'efficacité de la solution `try_lock`. Les mérites des structures non-bloquantes s'évaluent essentiellement selon deux critères [39] : leur capacité de traitement (*throughput*), c'est-à-dire la quantité de données pouvant être produites–consommées par la structure par unité de temps, et leur scalabilité, autrement dit leur capacité à "fournir plus de travail" lorsque plus de *threads* sont mobilisés. La scalabilité constitue un facteur crucial pour les applications massivement parallèles, mais, comme évoqué précédemment, ce critère est assez peu pertinent pour les programmes audio où le nombre de *threads* concurrents est très faible. Nous nous focaliserons donc sur le *throughput*.

### 3.2. Protocole

Nous réalisons un test comparatif entre la technique de *mutex* non-bloquants et une approche *lock-free*. Nous considérons le cas d'une queue de capacité fixe (de taille arbitraire $2^{15} = 32768$ éléments). Afin de s'affranchir des problèmes de gestion mémoire (voir paragraphe 2.5), nous testons une queue d'éléments entiers, c'est-à-dire que la queue *lock-free* s'appuie sur `std::atomic<int>`. Pour l'implémentation *lock-free*, nous utilisons une bibliothèque développée par Erik Rigtorp [16]. Celle-ci est simple d'usage (fichiers d'entête seulement), écrite en utilisant des idiomes `C++` modernes, et optimisée (utilisation d'atomiques de bas-niveau, alignement de cache, etc.). Elle propose une queue SPSC et une queue MPMC (il est plus rare de trouver des implémentations MSPC ou SPMC). Les tests sont conduits dans l'environnement Max (7.3.5 en 64 bit), sous macOS (10.13.4), processeur Intel Core i7 (4 Mo de cache L3). Nous évaluons le *throughput* des différentes implémentations en mesurant le nombre d'événements qui peuvent être écrits (*push*) et lus (*pop*) dans la queue pendant une unité de temps. Les événements sont simplement des messages Max convoyant un nombre entier tiré aléatoirement. Le *thread* audio est le seul et unique

15 . Il est également possible d'obtenir une granularité plus fine en re-découpant le bloc audio en sous-blocs de taille plus petite.

16 . `https://github.com/rigtorp/SPSCQueue` `https://github.com/rigtorp/MPMCQueue`

*consumer* de ces messages. À chaque entrée dans le *callback* audio, l'approche par *mutex* opère un `try_lock` ; s'il est concluant, tous les événements de la queue sont dépilés un par un. Sinon l'exécution continue et un nouvel essai sera réalisé au bloc suivant. Les tests sont menés pour différentes tailles de bloc audio. La répétabilité n'étant pas garantie, les résultats sont moyennés sur trois réalisations de dix secondes chacune.

Dans le test #1, un seul *thread producer* (à haute priorité) est considéré, les messages étant générés par un métronome (objet `metro`). Le test #2 est identique, sinon que la contention globale du programme est accrue, en déclenchant un nombre considérable d'événements concurrents (*stress test*) dans les *threads* haute priorité, basse priorité, et d'autres *threads* de priorité normale. Le test #3 est similaire à #1, mais deux *threads producers* sont mobilisés, en utilisant à la fois `metro` et `qmetro`. Le test #4 est similaire à #3, mais la contention est accrue selon un protocole similaire à #2. Le test #4 représente la situation la plus réaliste d'un processeur audio temps réel en situation de performance. L'implémentation SPSC n'est pas valide lorsque plusieurs *threads* sont *producers*, aussi n'est-elle évaluée que dans les tests #1 et #2.

### 3.3. Analyse des résultats

Les résultats sont présentés dans la figure 1. Les données absolues n'ayant guère d'utilité ici, nous nous limiterons à des analyses en relatif.

Test #1 : les performances des queues *lock-free* SPSC et MPMC sont sensiblement identiques. Elles se dégradent notablement pour des tailles de bloc de $1024$ et $2048$. Nous émettons l'hypothèse que ce phénomène pourrait être lié à des problèmes de cache CPU ou d'erreur de page (*cache miss* ou *page fault*). Les performances de l'implémentation `try_lock` ne dépendent (quasiment) pas de la taille de *buffer*. Pour des *buffers* de taille $\leq 512$, les queues *lock-free* surpassent nettement le *mutex*, et la variante SPSC est marginalement meilleure que MPMC (comme conjecturé paragraphe 2.4.2).

Test #2 : En présence de contention, les performances des trois implémentations se dégradent nettement. Toutefois la méthode *mutex* (avec une diminution de throughput d'environ $20\%$) semble plus robuste que les approches *lock-free* (perte d'environ $35\%$). Le comportement "anormal" des structures *lock-free* pour un bloc de $2048$ (et $1024$) est similaire au test #1 et demeure inexpliqué.

Test #3 : En présence de *threads producers* concurrents, les performances générales diminuent encore. L'avantage de la queue MPMC sur le `try_lock` devient plus marginal.

Test #4 : Le throughput des implémentations testées est globalement moindre que dans #3. Le gain de MPMC par rapport au *mutex* non-bloquant est approximativement de $10\%$.

En définitive, les implémentations *lock-free* sont globalement plus performantes que l'approche par *mutex* `try_lock`, toutefois elles sont plus sensibles à la contention. Dans le cas le plus réaliste (test #4), le gain de performances de MPMC est minime, d'autant que ces benchmarks ont été réalisés sur des queues d'entier donc ne tenant pas compte du possible surcoût de gestion dynamique de la mémoire. Les *mutex* font partie des primitives de base de tous les OS courants ; il est donc permis de croire qu'ils sont extrêmement optimisés pour tous les usages "courants", justifiant ainsi leur bon rendement ici observé.

Ces remarques sont proches des conclusions dressées dans [13], dans lequel l'auteur évalue les mérites respectifs des queues *lock-free* (implémentation de la bibliothèque Boost) et à base de lock : "[...] both the lock-based and lock-free queue perform just as well. This is contrary to [19], which suggests that audio programs should use lock-free data structures. These patterns, however, were based on the concurrency primitives that were available to programmers in 2005. The assumption now is that the newer `C++` concurrency primitives perform much better than their older counterparts [17] , meaning that for some applications, the implementation overheads of a lock-free queue may not warrant its use."

## 4. CONCLUSION

Le développement d'applications audio temps réel nécessite la synchronisation de données partagées entre différents processus et le *thread* audio. En raison des contraintes spécifiques auxquelles est soumis ce dernier, on recommande fréquemment la mise en œuvre de mécanismes *lock-free*, afin d'éviter les phénomènes d'inter-blocage ou d'inversion de priorité. La programmation de structures *lock-free*, basée sur des variables atomiques, est particulièrement ardue et propice aux erreurs car elle requiert une connaissance experte du langage, du *memory model* et de l'architecture *hardware*. Elle complexifie en outre la gestion dynamique des ressources mémoire.

Dans cet article, nous avançons que les approches *lock-free*, si elles sont parfois les plus performantes, ne sont généralement pas indispensables dans un contexte d'audio temps réel, et elles devraient être évitées au profit de l'hygiène du code. Comme substitut simple et pragmatique, nous proposons un paradigme *thread-safe* non-bloquant s'appuyant sur le `try_lock` des *mutex*. Cette méthode garantit une progression *obstruction-free* du *thread* audio. Nous montrons que les performances obtenues sont proches des structures *lock-free*, et viables pour une exploitation en production [18] .

17 . N. Pipenbrinck. Online discussion on lock-free data structures. `http://stackoverflow.com/questions/27738660`

18 . La technique `try_lock` est effectivement mise en œuvre dans plusieurs applications temps réel développées par l'auteur [16, 15, 28, 14].

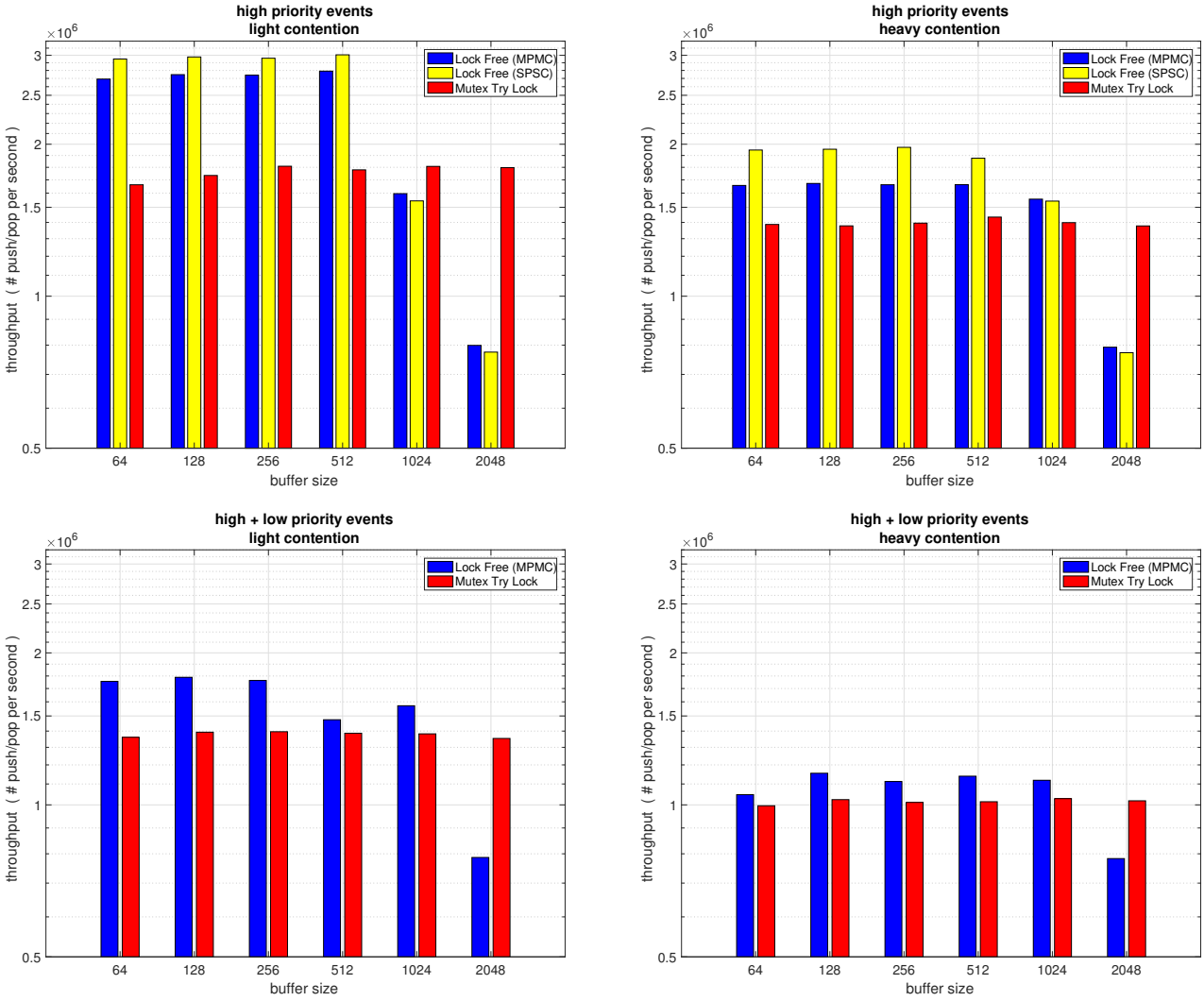



**Figure 1**. En haut à gauche : test #1. En haut à droite : test #2. En bas à gauche : test #3. En bas à droite : test #4.

## 5. REFERENCES


[1] Adve, S.V., Boehm, H.J., "Memory Models : A Case for Rethinking Parallel Languages and Hardware", *Communications of the ACM*, volume 53(8), pages 90 – 101, Aug 2010.

[2] Alexandrescu, A., "Lock-Free Data Structures", *C/C++ Users Journal*, Oct 2004.

[3] Alexandrescu, A., Michael, M.M., "Lock-Free Data Structures with Hazard Pointers", *C/C++ Users Journal*, Dec 2004.

[4] Andrews, G.R., *Foundations of Multithreaded, Parallel, and Distributed Programming*, Pearson, 1999.

[5] Batty, M., Owens, S., Sarkar, S., Sewell, P., Weber, T., "Mathematizing C++ Concurrency", *Proceedings of the 38th annual Symposium on Principles of programming languages (ACM SIGPLAN-SIGACT)*, pages 55 – 66, Austin, TX, USA, Jan 2011.

[6] Bencina, R., "Interfacing real-time audio and file I/O", *Proc. of the Australasian Computer Music Conference (ACMC)*, pages 21 – 28, Melbourne, July 2014.

[7] Blechmann, T., "Supernova – A scalable parallel audio synthesis server for SuperCollider", *Proc. International Computer Music Conference (ICMC)*, Huddersfield, UK, August 2011.

[8] Boehm, H.J., "Threads Cannot Be Implemented As a Library", *Proc of the Conference on Programming Language Design and Implementation (ACM SIGPLAN)*, volume 40, pages 261 – 268, Chicago, IL, USA, June 2005.

[9] Boehm, H.J., "Position Paper : Nondeterminism is unavoidable, but data races are pure evil", *Proc of the ACM workshop on Relaxing synchronization for multicore and manycore scalability*, pages 9 – 14, Tucson, AZ, USA, Oct 2012.

[10] Boehm, H.J., Adve, S.V., "Foundations of the C++ Concurrency Memory Model", *Proc. of the Conference on Programming Language Design and Imple-*

*mentation (ACM SIGPLAN)*, volume 43, pages 68 – 78, Tucson, AZ, USA, June 2008.

[11] Boehm, H.J., Adve, S.V., "You Don't Know Jack about Shared Variables or Memory Models", *ACM Queue*, volume 9(12), Dec 2011.

[12] Boulanger, R., Lazzarini, V., *The Audio Programming Book*, MIT Press, 2011.

[13] Cameron, E., *Parallelizing the ALSA modular audio synthesizer*, Master's thesis, Concordia University, Montreal, Canada, 2015.

[14] Carpentier, T., "ToscA : An OSC Communication Plugin for Object-Oriented Spatialization Authoring", *Proc. of the 41st International Computer Music Conference*, pages 368 – 371, Denton, TX, USA, Sept. 2015.

[15] Carpentier, T., "Panoramix : 3D mixing and post-production workstation", *Proc. 42nd International Computer Music Conference (ICMC)*, pages 122 – 127, Utrecht, Netherlands, Sept 2016.

[16] Carpentier, T., Noisternig, M., Warusfel, O., "Twenty Years of Ircam Spat : Looking Back, Looking Forward", *Proc. of the 41st International Computer Music Conference*, pages 270 – 277, Denton, TX, USA, Sept. 2015.

[17] Cohen, N., Petrank, E., "Efficient Memory Management for Lock-Free Data Structures with Optimistic Access", *Proc. of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 254 – 263, Portland, OR, USA, June 2015.

[18] Colmenares, J.A., Saxton, I., Battenberg, E., Avizienis, R., Peters, N., Asanovic, K., Kubiatowicz, J.D., Wessel, D., "Real-time Musical Applications on an Experimental Operating System for Multi-Core Processors", *Proc. International Computer Music Conference (ICMC)*, Huddersfield, UK, August 2011.

[19] Dannenberg, R.B., Bencina, R., "Design Patterns for Real-Time Computer Music Systems", *Workshop on Real Time Systems Concepts for Computer Music (ICMC)*, Sept 2005.

[20] Dechev, D., Pirkelbauer, P., Stroustrup, B., "Understanding and Effectively Preventing the ABA Problem in Descriptor-based Lock-free Designs", *Proc. 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, pages 185– 192, Seville, Spain, May 2010.

[21] Dechev, D., Stroustrup, B., "Scalable Nonblocking Concurrent Objects for Mission Critical Code", *Proc. ACM Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, Orlando, FL, USA, Oct 2009.

[22] Detlefs, D.L., Martin, P.A., Moir, M., Jr., G.L.S., "Lock-free reference counting", *Distributed Computing*, volume 15(4), pages 255 — 271, 2002.

[23] Fober, D., Orlarey, Y., Letz, S., "Optimised Lock-Free FIFO Queue", Technical report, GRAME - Computer Music Research Lab, Lyon, France, Jan 2001.

[24] Fober, D., Orlarey, Y., Letz, S., "Lock-Free Techniques for Concurrent Access to Shared Objects", *Actes des Journées d'Informatique musicale (JIM)*, pages 143 – 150, Marseille, France, 2002.

[25] Fraser, K., "Practical lock-freedom", Technical report, University of Cambridge Computer Laboratory, Feb 2004.

[26] Fraser, K., Harris, T., "Concurrent Programming Without Locks", *ACM Transactions on Computer Systems (TOCS)*, volume 25(2), May 2007.

[27] Gao, H., Groote, J., Hesselink, W., "Lock-free parallel and concurrent garbage collection by mark & sweep", *Science of Computer Programming*, volume 64, pages 341 — 374, 2007.

[28] Geier, M., Carpentier, T., Noisternig, M., Warusfel, O., "Software tools for object-based audio production using the Audio Definition Model", *Proc. of the 4th International Conference on Spatial Audio (ICSA)*, Graz, Austria, Sept 2017.

[29] Gidenstam, A., Papatriantafilou, M., Sundell, H., Tsigas, P., "Practical and Efficient Lock-Free Garbage Collection Based on Reference Counting", Technical Report no. 2005-04, Chalmers University of Technology and Göteborg University, Göteborg, Sweden, 2005.

[30] Herlihy, M., Luchangco, V., Moir, M., "Obstruction-free synchronization : double-ended queues as an example", *Proc. of the 23rd International Conference on Distributed Computing Systems*, pages 522 – 529, May 2003.

[31] Herlihy, M.P., "Impossibility and universality results for wait-free synchronization", *Proc. of the seventh annual ACM Symposium on Principles of distributed computing (POD)*, pages 276 – 290, Toronto, Canada, August 1988.

[32] Herlihy, M.P., "A methodology for implementing highly concurrent data objects", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, volume 15(5), pages 745 — 770, Nov 1993.

[33] Lamport, L., "How to make a multi-processor computer that correctly executes multiprocess programs", *IEEE Transactions on Computers*, volume C-28(9), pages 690 – 691, Sept 1979.

[34] Lamport, L., "The temporal logic of actions", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, volume 16(3), pages 872 – 923, May 1994.

[35] Letz, S., Fober, D., Orlarey, Y., "Jack audio server for multi-processor machines", *Proc. of the International Computer Music Conference (ICMC)*, pages 1 – 4, Barcelona, Spain, Sept 2005.

[36] Manson, J., Pugh, W., Adve, S.V., "The Java Memory Model", *Proc of the 32nd Symposium on Principles of programming languages (ACM SIGPLAN-SIGACT)*, volume 40, pages 378 – 391, Long Beach, CA, USA, Jan 2005.

[37] McCartney, J., "Rethinking the Computer Music Language : SuperCollider", *Computer Music Journal*, volume 26(4), pages 61 – 68, Winter 2002.

[38] McCurry, M., "STatic (LLVM) Object Analysis Tool : Stoat", *Proc. of the Linux Audio Conference (LAC)*, Saint-Etienne, France, May 2017.

[39] Meneghin, M., Pasetto, D., Franke, H., Petrini, F., Xenidis, J., "Performance evaluation of inter-thread communication mechanisms on multicore/multithreaded architectures", *Proc. of the 21st international symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2012.

[40] Michael, M., "Hazard pointers : Safe memory reclamation for lock-free objects", *IEEE Transactions on Parallel and Distributed Systems*, volume 15(6), pages 491 — 504, 2004.

[41] Michael, M.M., "The Balancing Act of Choosing Nonblocking Features", *Communications of the ACM*, volume 56(9), pages 46 – 53, Sept 2013.

[42] Michael, M.M., Scott, M.L., "Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms", *Proc. of the fifteenth annual ACM symposium on Principles of distributed computing (PODC)*, pages 267 – 275, 1996.

[43] Michael, M.M., Scott, M.L., "Nonblocking Algorithms and Preemption-Safe Locking on Multiprogrammed Shared Memory Multiprocessors", *Journal of Parallel and Distributed Computing*, volume 51(1), pages 1 – 26, May 1998.

[44] Ostrovsky, I., "The C# Memory Model in Theory and Practice", Dec 2012.

[45] Otenko, O., "System and method for efficient concurrent queue implementation – US Patent 8,607,249 B2", Technical report, Oracle International Corporation, Dec 2013.

[46] Pacheco, P., *An Introduction to Parallel Programming*, Morgan Kaufmann, 2011.

[47] Pirkle, W., *Designing Audio Effect Plug-Ins in C++ : With Digital Audio Signal Processing Theory*, Focal Press, 2012.

[48] Puckette, M., "The Patcher", *Proc. International Computer Music Conference (ICMC)*, pages 420 – 429, San Francisco, CA, USA, 1988.

[49] Puckette, M., "Pure Data", *Proc. of the International Computer Music Conference (ICMC)*, pages 224 – 227, Thessaloniki, Greece, 1997.

[50] Raynal, M., *Concurrent Programming : Algorithms, Principles, and Foundations*, Springer, 2013.

[51] Robinson, M., *Getting Started with JUCE*, Packt Publishing, 2013.

[52] Rushworth, T.B., Telfer, A.R., "Multi-reader, multi-writer lock-free ring buffer", Technical report, Inetco Systems Limited, 2012.

[53] Schmeder, A., Freed, A., Wessel, D., "Best Practices for Open Sound Control", *Proc. of the Linux Audio Conference (LAC)*, Utrecht, Netherlands, May 2010.

[54] Schnell, N., Roebel, A., Schwarz, D., Peeters, G., Borghesi, R., "MuBu and friends–assembling tools for content based real-time interactive audio processing in Max/MSP", *Proc. International Computer Music Conference (ICMC)*, pages 423 – 426, Montreal, Canada, Aug 2009.

[55] Shelton, R.J., *A Lock-Free Environment for Computer Music : Concurrent Components for Computer Supported Cooperative Work*, Ph.D. thesis, University of Melbourne, 2011.

[56] Valois, J.D., "Lock-Free Linked Lists Using Compare-and Swap", *Proc. of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 214 – 222, 1995.

[57] Williams, A., *C++ Concurrency in Action Practical Multithreading*, Manning Publications, 2010.

# UNE NOUVELLE IMPLÉMENTATION DU SPATIALISATEUR DANS MAX

*Thibaut Carpentier*
CNRS – Ircam – Sorbonne Université – Ministère de la Culture
Sciences et Technologies de la Musique et du Son (UMR 9912 STMS)
1, place Igor Stravinsky, 75004 Paris
thibaut.carpentier@ircam.fr

**RÉSUMÉ**

Le Spatialisateur (`spat~`) est une bibliothèque d'outils dédiés à la spatialisation sonore, la réverbération artificielle, et la diffusion multicanale. Cet article présente une révision majeure de l'environnement (version 5) et son intégration dans Max.

## 1. INTRODUCTION

Le Spatialisateur [20, 22], communément appelé `spat~`, est un outil temps réel dédié au traitement de spatialisation sonore, à la réverbération artificielle, et à la diffusion multicanale. Il est développé à l'Ircam depuis le début des années 1990, et s'incarne principalement dans l'environnement Max [27]. Il se présente sous la forme d'une bibliothèque de processeurs articulés autour d'un moteur de réverbération à réseau de retards rebouclés [21] et de modules de panoramique. Ces processeurs sont pilotés par le biais d'une interface de haut-niveau qui permet de moduler la qualité acoustique de l'effet de salle synthétisé selon plusieurs attributs perceptivement pertinents [23, 24].

Visant des domaines d'application variés (concerts, mixage, post-production, réalité virtuelle, installations interactives, etc.), le logiciel est développé selon des méthodologies agiles [26] et il agrège en continu les principaux résultats de recherche de l'équipe Espaces Acoustiques et Cognitifs (anciennement Acoustique des Salles). L'outil a donc connu au fil des ans diverses évolutions, et la dernière révision majeure, `spat~` 4, fut diffusée en 2009 et présentée dans [13, 4].

Cet article présente `spat~` 5, une nouvelle mouture de l'environnement. Comme tout logiciel exploité au long cours, son cycle de développement s'oriente selon différentes directions : l'amélioration des fonctionnalités existantes (incluant maintenance corrective, adaptative et évolutive), l'ajout de nouvelles fonctionnalités, et le réusinage (*refactoring*) de l'infrastructure logicielle (afin d'assainir l'architecture et assurer sa pérennité). Nous présentons dans les sections 2 et 3 une profonde refonte architecturale, et des nouvelles fonctionnalités sont décrites dans la section 4.

Le modèle interne du `spat~` se fonde historiquement sur une représentation orientée objet de l'effet de salle, constitué de quatre sections temporelles filtrées et pannées indépendamment : son direct, réflexions précoces, réflexions tardives et queue de réverbération (voir par exemple la section 3 dans [4]). Cette architecture est perpétuée dans `spat~` 5 ; en revanche, l'une des principales évolutions introduites dans cette nouvelle version concerne l'interfaçage de l'outil avec son (ou ses) environnement(s) hôte(s).

## 2. INTERFAÇAGE OSC

### 2.1. Motivations

Sorti en 2008, Max 5 introduisit pour la première fois la notion "d'attributs". Par analogie avec les paradigmes de programmation orientée objet, on peut dire que chaque objet (*external*) dans Max est une instance de classe, et les attributs sont des variables membres de cette classe. Grâce aux fonctionnalités offertes dans l'API, il est possible d'exposer publiquement (i.e. à l'utilisateur du patch Max) les attributs et les manipuler via des accesseurs (*getter*) et mutateurs (*setter*). `spat~` 4 utilise intensivement ce concept, puisque les paramètres utilisateur des modules du `spat~` sont exposés majoritairement via leurs attributs. Ce choix de conception présente des avantages et, rétrospectivement, un certain nombre d'inconvénients. Les avantages sont clairs : le mécanisme d'attributs est très bien intégré dans Max et il est aisé de manipuler (`attrui`, `getattr`, *inspector*) ou sauvegarder (`pattr`, `pattrstorage`) les attributs d'objets ; leur syntaxe est explicite (par opposition aux "arguments"), et leur mise en œuvre (pour le développeur) via l'API est aisée. Dans le contexte spécifique du `spat~`, il peut cependant s'avérer que les attributs fussent inadéquats pour plusieurs raisons :

— Les objets `spat~`, multicanaux par essence, présentent généralement un grand nombre d'attributs (souvent proportionnel au nombre de canaux). Dans les menus contextuels de Max, ceci peut conduire à une navigation mal commode et lente. En outre, les attributs dans Max sont stockés sous forme de listes (*array*) ce qui s'avère assez inapproprié aux paramètres du `spat~` ; une structure hiérarchique serait plus pertinente pour refléter l'arborescence des données [1]. En outre, stocker

1 . Il est en théorie possible de créer des attributs d'attributs, et donc d'élaborer une forme d'arborescence ; toutefois cela ne résout pas vraiment la question et ne va pas dans le sens de la simplicité.

Thibaut Carpentier, *Une nouvelle implémentation du Spatialisateur dans Max*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 45–52, 2018. 

les attributs sous forme de listes peut nuire aux performances : par exemple `pattr` (mécanisme de sauvegarde des attributs) aura besoin de copier l'ensemble de la liste dès lors qu'un des éléments constitutif est modifié.

— Les objets `spat~` sont souvent polymorphiques : le nombre et la nature des paramètres exposés peuvent évoluer au cours du temps. Un exemple est l'objet `spat.pan~` qui présente des propriétés différentes selon qu'il opère en stéréo, binaural, Ambisonics, etc. Les attributs de l'API Max sont mal adaptés pour refléter ce polymorphisme [2].

— Les attributs sont spécifiques à l'API de Max ; or la bibliothèque logicielle `spat~` est *bindée* vers de nombreux autres environnements (Matlab [3], Spat Revolution [4], plugins Ircam Tools [5], Open Music [17], Pure Data [6], etc.). Chaque nouveau binding requiert une couche de code de liaison (*glue code*) pour s'interfacer avec l'environnement hôte. Côté développeur, il est important de minimiser l'effort induit par ce *glue code* ; côté utilisateur, il est pertinent d'avoir un interfaçage similaire (e.g. même syntaxe) dans les différents hôtes.

Ces considérations nous ont amené à "changer notre fusil d'épaule" et adopter dans `spat~` 5 un interfaçage selon le protocole et la syntaxe Open Sound Control (OSC [31]). Les attributs sont désormais abandonnés.

Le choix de l'OSC s'est imposé de façon somme toute évidente : ce protocole est largement répandu et maîtrisé dans la communauté audio, il encourage et facilite la communication (e.g. via UDP/IP) inter-applications ou avec des interfaces de contrôle compatibles ; son implémentation est très simple (de nombreuses bibliothèques sont par ailleurs disponibles dans plusieurs langages). L'espace d'adressage hiérarchique se prête bien à la plupart des applications d'informatique musicale et notamment du `spat~`. Outre les messages conventionnels, le protocole permet également d'encapsuler plusieurs messages au sein d'un *bundle*, simplifiant la transmission synchrone de grandes quantités d'événements. Enfin, il offre un puissant mécanisme de distribution (*dispatching*) de messages grâce à sa sémantique de *pattern matching*.

## 2.2. Intégration dans Max

### *2.2.1. Syntaxe*

Les *externals* utilisent donc une syntaxe OSC. À l'interface avec Max, les messages sont convertis de/vers le format natif de Max, les `atoms`. Cette conversion est triviale puisque le typage des arguments OSC est proche de celui des `atoms` (int, float, symbole, etc.). Les *bundles* OSC sont quant à eux transmis sous forme de `FullPacket` véhiculant uniquement un pointeur vers la mémoire du *bundle* (à l'instar des `dict` de Max). Ceci peut donc permettre de transmettre d'importantes quantités de données de façon efficace (le scheduler de Max n'est sollicité qu'une fois par bundle, et non pour chaque message).

La syntaxe adoptée s'inspire du style REST [29] (Representational State Transfer), et s'avère très proche de la syntaxe de `spat~` 4 (en y ajoutant le séparateur `/`). Par exemple pour spécifier la position cartésienne d'une source dans `spat~` 5, on pourra utiliser le message :

```
/source/1/xy [float][float]
```

Les *externals* supportent les sémantiques habituelles de *pattern matching*, favorisant le *grouping* d'éléments :

```
/source/*/mute [boolean]
/source/[2-5]/mute [boolean]
/source/{3,6,7}/mute [boolean]
```

Le routage et la distribution de *patterns* d'adresses OSC dans Max peut nécessiter la manipulation d'expressions régulières (`regexp`) ce qui est généralement mal commode et peu performant ; aussi avons-nous développé une bibliothèque d'outils utilitaires (environ 25 *externals*) pour simplifier les opérations usuelles (voir Figure 1).

D'un point de vue du développement logiciel, les adresses OSC les plus couramment utilisées dans `spat~` sont stockées (lors de la compilation) dans une table de hash. Ceci évite toute manipulation (coûteuse) de *strings* lors de l'exécution et garantit ainsi une résolution dynamique aussi performante que les tables de symboles traditionnelles de Max.

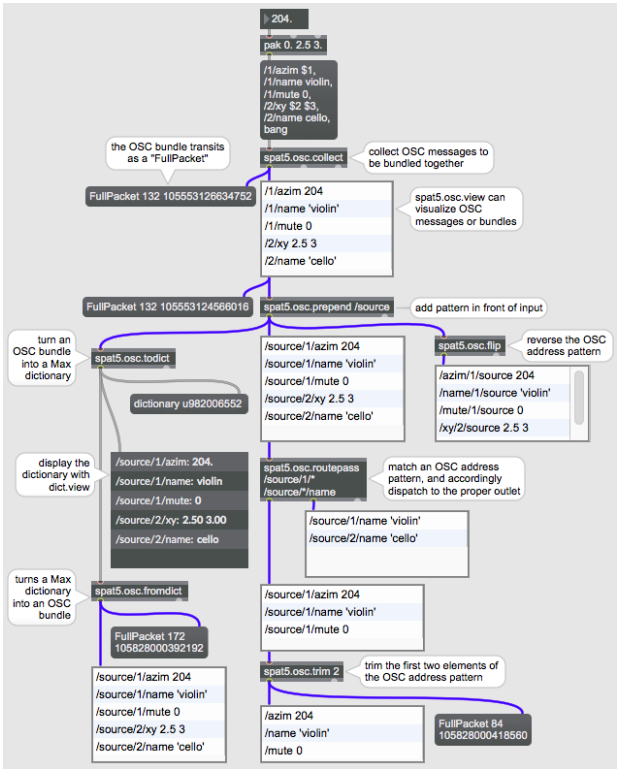



**Figure 1**. Exemple de manipulations OSC usuelles. Les *patchcords* représentées en bleu véhiculent des bundles OSC (`FullPacket`).

2 . Certes, il existe des attributs d'objet – par opposition aux attributs de classe – qui peuvent être ajoutés/supprimés dynamiquement, mais cela contrevient également à la simplicité de mise en œuvre.

3 . Non diffusé publiquement au moment de la rédaction de cet article.

4 . `www.spatrevolution.com`

5 . `www.ircamtools.com`

6 . Non diffusé publiquement au moment de la rédaction de cet article.

### 2.2.2. *Inter-opérabilité et compatibilité*

Un avantage potentiel de l'interfaçage OSC est que les utilisateurs peuvent également bénéficier des bibliothèques et outils OSC existants, notamment :

— le *package* `odot` [16] qui offre un langage de manipulation d'expressions (parseur et interpréteur) opérant sur des bundles OSC. Typiquement, `odot` pourra être utilisé pour la génération et la transformation algorithmiques de données de spatialisation (trajectoires).
— ToscA [6, 5], un plugin insérable dans les stations de travail et qui permet la transmission via OSC de données d'automation. Bien que générique, ce plugin a, dès sa conception, été pensé pour un usage couplé avec `spat~`.

En revanche, il convient de noter que la syntaxe OSC ne préserve pas la compatibilité descendante avec `spat~` 4. Cet aspect n'est évidemment pas négligeable puisque plusieurs centaines d'œuvres s'appuient sur `spat~` 4. Il n'existe à l'heure actuelle pas de mécanisme de "portage automatique" de `spat~` 4 vers `spat~` 5 ; toutefois nous estimons que, dans la majorité des cas, la transition pourra se faire de façon indolore (il s'agit essentiellement d'adaptation syntaxique mineure). Des exemples de portage de patchers canoniques sont également fournis dans la distribution.
Notons enfin que `spat~` 4 et `spat~` 5 peuvent cohabiter sans conflit (notamment pour permettre le portage progressif des pièces) puisque les *externals* `spat~` 5 s'inscrivent dans un espace d'adressage (*namespace*) dédié (préfixe `spat5.*`).

## 3. AUTRES REFONTES DE L'ENVIRONNEMENT

La refonte de l'environnement engagée dans `spat~` 5 ne se restreint pas uniquement à la syntaxe des objets, mais a une incidence à plusieurs niveaux : ergonomie générale, processeurs audio, et interfaces graphiques de contrôle.

### 3.1. Ergonomie

Comme signalé dans la section 2, les *externals* de `spat~` 5 n'utilisent plus d'attributs. En conséquence, ils ne bénéficient plus de certaines fonctionnalités de Max telles que l'inspecteur ou les mécanismes de documentation automatique (*hints*). Pour pallier cela, des mécanismes "de substitution" ont été introduits : chaque objet dispose d'une fenêtre de *status* et de *help* (voir Figure 2). La fenêtre de *status* affiche l'état courant de tous les paramètres de l'objet ; à l'instar de l'inspecteur Max, il est possible de filtrer les requêtes, et sélectionner et copier des messages depuis cette fenêtre vers le patcher. La fenêtre *help* affiche une documentation textuelle de tous les messages OSC supportés. Des pages de référence sont également générées automatiquement et sont accessibles via le navigateur standard de documentation de Max.

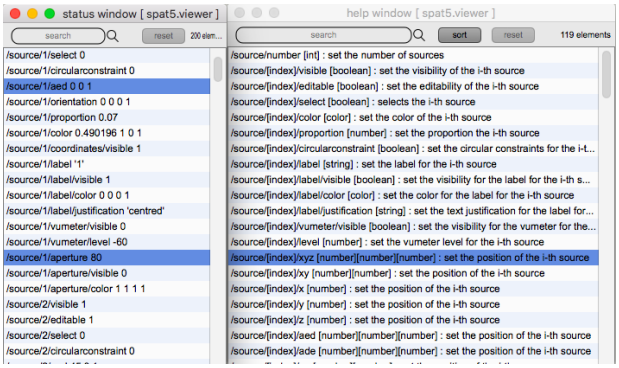



**Figure 2**. Fenêtres de statut (à gauche) et de documentation (à droite) de l'objet `spat5.viewer`.

Si l'on compare avec l'inspecteur d'attributs dans `spat~` 4 (Figure 3), on peut comprendre certains bénéfices de la nouvelle architecture : la fenêtre de *status* propose une vision hiérarchique des paramètres qui est plus claire que les listes (*array*) de valeurs dans l'inspecteur ; en outre ceci permet une granularité beaucoup plus fine : par exemple l'attribut `sourceseditable [boolean]` de `spat~` 4 (qui autorise ou non l'édition des sources) agit globalement sur l'ensemble des sources, tandis que `spat~` 5 offre un contrôle indépendant pour chaque élément :
`/source/i/editable [boolean]`

L'actualisation de paramètres indépendants plutôt que de (longues) listes de données peut également améliorer les performances générales de l'outil.

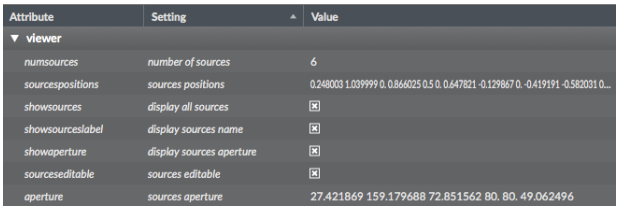



**Figure 3**. Inspecteur du `spat.viewer` dans `spat~` 4.

### 3.2. Ordonnancement et *thread-safety*

Par rapport aux attributs, l'encapsulation des événements dans des messages OSC simplifie considérablement le développement d'une queue *thread-safe* pour la synchronisation de données partagées entre les différents processus impliqués dans Max (*thread* audio, *thread* message de l'application, *thread* d'événements à haute priorité, etc.), épurant ainsi l'hygiène du code et limitant significativement le risque de bogues : les événements entrants (messages ou bundles) sont empilés dans une queue FIFO puis traités au moment et dans le *thread* opportuns. L'accès à cette queue est garanti *thread-safe* et non-bloquant [7]. A contra-

7. Ce mécanisme est décrit en détail dans [11].

rio, seuls quelques objets dans `spat~` 4 étaient garantis *thread-safe*.

Pour les objets DSP, le comportement par défaut est que la queue est dépilée dans le *thread* audio, au début du *callback* (voir Figure 4). Ce comportement correspond peu ou prou au mécanisme dit *scheduler in audio interrupt* dans Max. Il contribue également à une meilleure sécurité (*thread-safety*) du code. Notons que tous les objets `spat~` 5 fonctionnent selon ce mécanisme, indépendamment des réglages *overdrive* ou *interrupt* de l'environnement hôte.

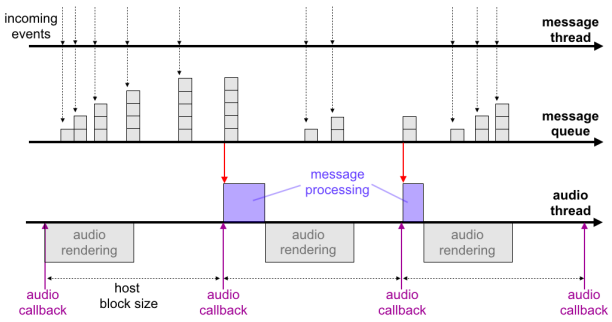



**Figure 4**. Ordonnancement des événements selon un mécanisme *scheduler in audio interrupt*.

### 3.3. Interfaces de contrôle

Le *package* `spat~` contient une vingtaine d'*externals* d'interfaces graphiques de contrôle (voir par exemple Figure 5). Ces interfaces ont été "rafraîchies" pour une meilleure lisibilité et ergonomie, et de nombreuses options de personnalisation y ont été ajoutées. Ont également été ajoutés un grand nombre de raccourcis clavier pour un accès immédiat aux fonctionnalités usuelles ; ces raccourcis sont par ailleurs configurables par l'utilisateur (Figure 6).

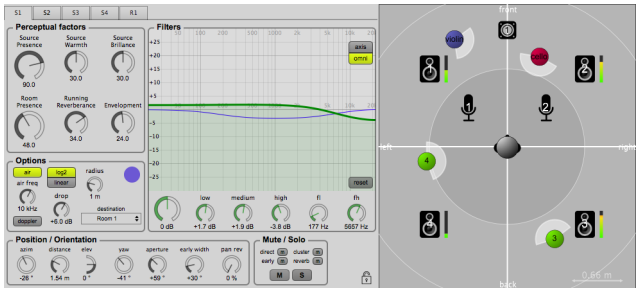



**Figure 5**. Interface du `spat5.oper` (opérateur perceptif de contrôle haut-niveau du `spat~`). Facteurs perceptifs de contrôle de l'effet de salle (gauche) ; filtrages (centre) ; représentation schématique 2D de la scène sonore (droite).

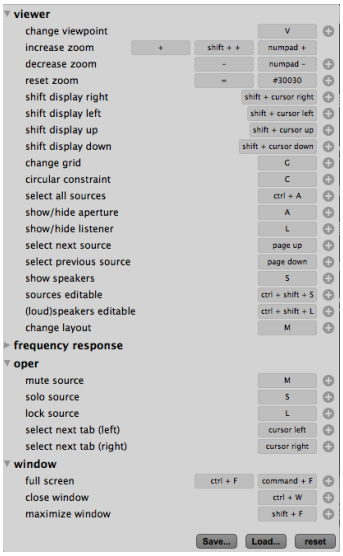



**Figure 6**. Fenêtre de personnalisation des raccourcis clavier pour `spat5.oper`.

Les *externals* portant ces interfaces graphiques bénéficient par ailleurs du mécanisme de queue *thread-safe* présenté au paragraphe précédent. Ceci a permis de les implémenter sans recourir à *deferlow* (renvoi dans le *thread* à basse priorité), offrant ainsi une réactivité bien supérieure à `spat~` 4 (dans lequel la plupart des GUIs utilisaient *deferlow*, ce qui conduit, en cas de charge importante, à un empilement des messages et un ralentissement global de l'interface et du scheduler).

L'état global (i.e. l'ensemble des paramètres) d'un *external* GUI est représenté sous forme d'un bundle OSC (voir par exemple Figure 2). Ce bundle peut facilement être exporté sous forme de fichier texte (donc éditable), puis ré-importé. Ceci peut donc être utilisé comme mécanisme simple de sauvegarde/chargement de presets. Le bundle peut également être stocké directement dans le patcher (on dira qu'il est *embedded*) ou via les *snapshots* de Max (en activant le "Parameter Enable Mode") ; dans ces cas, le bundle OSC est préalablement converti sous forme de *blob* binaire (non éditable) sauvé avec le patcher.

### 3.4. Aspects de génie logiciel

Le code `C++` des bibliothèques sous-jacentes a été considérablement modernisé, en utilisant les nouveaux idiomes de programmation offerts par les standards `C++11` à `C++17` [8] : expressions constantes à la compilation (`constexpr`), déduction de typage automatique (`auto`), *lambda* fonctions, *range-based for loop*, etc. Ces nouvelles fonctionnalités permettent d'assainir l'hygiène du code, de diminuer les risques de bogues, et incidemment de réduire la taille du code source ; par exemple, le *glue code* requis pour l'interfaçage avec l'API de Max (voir section 2) a pu

8 . ISO International Standard ISO/IEC 14882 :2017(E) – Programming Language `C++`

être réduit de 80%.

Les algorithmes de traitement du signal, déjà vectorisés et optimisés grâce au *framework* `Accelerate` [9], ont encore été améliorés par l'utilisation des primitives hautes performances Intel® Integrated Performance Primitives (IPP) [10].

## 4. NOUVELLES FONCTIONNALITÉS

Cette section recense, sans souci d'exhaustivité, plusieurs nouveautés marquantes du *package* `spat~` 5.

### 4.1. Higher Order Ambisonics

Concernant les processeurs audio pour l'analyse/synthèse de scènes sonores spatialisées, de nombreuses améliorations et nouvelles fonctionnalités ont été introduites dans la bibliothèque `spat~` 5. Il serait fastidieux d'en tenir une liste complète. Toutefois les recherches des dernières années ont mis un accent particulier sur la technique Higher Order Ambisonics (HOA [14]), et nous énumérons ici quelques nouveautés afférentes :

— Les différents schémas de normalisation des composantes HOA en vigueur (FuMa, MaxN, SN3D, N3D, etc.) sont une source fréquente de confusion pour les utilisateurs, et ils nuisent à l'inter-opérabilité des outils. Un travail de formalisation [10] et de documentation a été mené afin de clarifier leur impact dans une chaîne de production Ambisonics, et de limiter les risques d'erreur.

— Plusieurs stratégies de décodage HOA sont offertes dans `spat~` (voir [4]). Dans `spat~` 5, nous y avons ajouté les décodeurs dits "all-rad" (All-Round Ambisonic Panning and Decoding [32]) et "constant-spread" (Constant Angular Spread Ambisonic Decoding [15]) qui s'appuient sur un décodage HOA régulier, sur une sphère virtuelle de type *t-design*, projeté via VBAP ou MDIP sur le dispositif de haut-parleurs physiques.

— Les différents décodeurs HOA disponibles (*sampling decoder*, *mode-matching* [14], *energy-preserving* [33], *all-rad* [32], *constant-spread* [15]) peuvent conduire à des champs sonores de puissances significativement différentes (jusqu'à plusieurs dizaines de dB d'écart selon les configurations). Ceci empêche toute étude/écoute comparative des différents décodages, aussi avons nous développé une technique de compensation énergétique qui permet d'aligner les décodeurs entre eux (ou sur une référence arbitraire). La méthode s'appuie sur une estimation de la puissance rendue en condition de champ diffus (voir par exemple la section 4.4 dans [33]).

— `spat5.hoa.blur~` est un nouvel outil permettant de manipuler la "résolution spatiale" d'un champ encodé HOA. Il permet de modifier continûment l'ordre du flux HOA (simulant de la sorte des ordres fractionnaires), tout en maintenant la puissance globale constante [9]. Il peut être utilisé pour adapter l'ordre de contenus existants, ou comme effet créatif (typiquement en faisant varier dynamiquement le facteur de "flou").

— `spat5.hoa.focus~` est un autre effet opérant dans le domaine HOA. Inspiré de [25], il permet de synthétiser des diagrammes de directivité virtuels qui "filtrent" un flux HOA. L'orientation et la sélectivité des diagrammes virtuels peuvent être éditées via une interface graphique dédiée (Figure 7). L'outil trouve un intérêt aussi bien dans le domaine de la post-production ("zoom" dans une scène sonore enregistrée) que de la création.

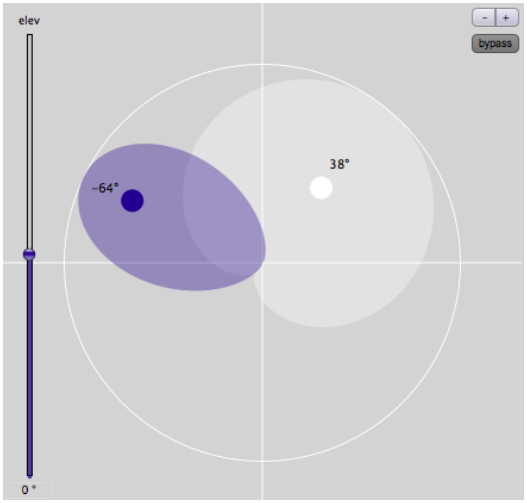



**Figure 7**. Module `spat5.hoa.focus` pour la synthèse de directivités virtuelles dans le domaine HOA.

### 4.2. Audio Definition Model

Ces dernières années ont vu un regain d'intérêt pour les modèles orientés objet pour la production et la diffusion de contenus multicanaux. Plusieurs formats d'échange ont ainsi été proposés ; en particulier Audio Definition Model (ADM [19]) est un standard ouvert publié par ITU et EBU [11] pour la description de médias orientés objet et encapsulés dans un conteneur Broadcast Wave Format (BWF). ADM spécifie un ensemble de métadonnées (notamment positions et gains d'objets sonores évoluant dans le temps et l'espace) codées selon une représentation XML. `spat~` est l'une des premières bibliothèques proposant une solution complète pour la création et le rendu de fichiers BWF-ADM : `spat5.adm.record~` permet l'écriture de fichier BWF embarquant des métadonnées de spatialisation, et `spat5.adm.renderer~` assure le rendu temps réel de médias ADM sur différents dispositifs de restitution (casque ou haut-parleurs) ; d'autres *externals* permettent

9. `http://developer.apple.com`
10. `http://software.intel.com`
11. International Telecommunication Union et European Broadcasting Union

également de gérer l'interactivité des objets. Ces outils sont décrits plus en détail dans [18]. Notons toutefois que seul un sous-ensemble des spécifications ADM est actuellement supporté (couvrant cependant la plupart des cas d'usage classiques), et qu'une intégration plus étroite du format au sein de l'architecture du spat∼ reste à faire (e.g. import/export de fichiers ADM directement depuis les processeurs tels que spat5.spat∼). Ceci fait l'objet de travaux en cours.

### 4.3. Panoramix

panoramix est une station de travail pour la spatialisation et la réverbération en contexte de mixage et de post-production 3D. L'outil a été présenté dans plusieurs publications [7, 12, 8]. Il s'appuie de façon sous-jacente sur la bibliothèque C++ spat∼, et son développement rapide n'a été possible que grâce à la profonde refonte logicielle exposée dans les sections 2 et 3. panoramix peut être vu comme une déclinaison spécialisée des outils spat5.oper et spat5.spat∼ (qui sont plus génériques), dotée d'un *frontend* ad-hoc pour les situations de mixage multicanal. Bien que distribué séparément sous forme d'exécutable standalone, panoramix est également inclus dans le *package* spat∼ 5, sous forme d'*externals* pouvant être utilisés pour des applications moins conventionnelles. Le lecteur est invité à consulter [7, 12, 8] pour plus de détails sur la conception et l'exploitation de panoramix.

### 4.4. Quaternions

Avec la démocratisation des équipements de réalité virtuelle, spat∼ est de plus en plus employé pour assurer le rendu audio d'applications multimédia immersives, typiquement présentées selon une modalité binaurale. Ces environnements requièrent des manipulations géométriques 3D plus ou moins complexes. En particulier, la gestion de l'orientation des entités (auditeur ou objets sonores/visuels) dans l'espace est une source fréquente de confusion pour les utilisateurs, en raison des nombreuses conventions en vigueur (et incompatibles entre elles). Pour pallier cette difficulté, nous avons développé une bibliothèque d'*externals* permettant de manipuler quaternions, angles d'Euler et matrices de rotation 3D. Ils assurent notamment les conversions entre ces différentes représentations. Les processeurs audio de spat∼ (par exemple spat5.binaural∼) peuvent être pilotés indifféremment par des quaternions ou des angles d'Euler, simplifiant l'inter-opérabilité avec les différents SDK de réalité virtuelle.

### 4.5. Time Code

spat∼ est fréquemment utilisé dans des contextes audiovisuels ; lors du couplage avec des environnements vidéo/graphiques, il est nécessaire de mettre en place des mécanismes de synchronisation entre les flux audio et vidéo. Une des techniques les plus populaires consiste alors à utiliser un *Linear Timecode* (LTC [28]) encodant des *frames* SMPTE et transmis sous forme de signal audio longitudinal. Ce standard n'est malheureusement pas supporté nativement par Max, aussi avons nous développé des outils permettant la réception (spat5.ltc.decode∼), et la génération (spat5.ltc.encode∼) de tels *timecodes*. En outre, l'objet spat5.ltc.trigger∼ peut être utilisé comme gestionnaire d'événements (*cue manager*) déclenchant des actions en fonction du timecode courant ; la granularité temporelle est certes assez faible (typiquement 30 fps $\approx$ 33 millisecondes), mais suffisante pour la plupart des applications en spatialisation sonore.

### 4.6. Intégration dans Open Music

La spatialisation sonore n'est pas seulement déployée lors de l'exécution des œuvres, mais elle doit également être intégrée au processus compositionnel. Aussi est-il pertinent de proposer des bindings de spat∼ dans des environnements de composition assistée par ordinateur. Comme discuté au paragraphe 2, une des motivations pour l'adoption d'une interface OSC est de s'abstraire des spécificités de Max, et d'accélérer l'intégration de spat∼ dans différentes applications hôtes, en offrant une syntaxe unifiée. Dans le cadre du projet EFFICACe [3], nous avons tiré profit de cette nouvelle architecture OSC pour insérer plusieurs modules de la bibliothèque spat∼ dans les environnements Open Music et o7. Les premiers fruits de cette intégration ont été présentés dans [17, 1, 2], et ils ouvrent la porte à de nouveaux champs d'expérimentation faisant l'objet de travaux et de résidences artistiques en cours.

## 5. CONCLUSION

Cet article présente spat∼ 5, une nouvelle mouture de l'environnement de spatialisation et réverbération spat∼ implémentée dans Max. La bibliothèque contient près de 200 *externals* couvrant toutes les activités liées à la diffusion multicanale. Par rapport aux versions précédentes, cette implémentation propose un nouvel interfaçage avec l'application hôte, basé sur le protocole OSC. Hormis les aspects syntaxiques, cette nouvelle structure OSC s'accompagne d'une refonte plus profonde de la bibliothèque, visant à améliorer son ergonomie, sa stabilité, ses performances et son inter-opérabilité. La distribution spat∼ 5 inclut par ailleurs nombre de nouveaux objets (contrôle, DSP, et GUI) qui élargissent le spectre des traitements réalisables. Signalons encore que plusieurs "sous-ensembles" du *package* (e.g. les modules OSC, LTC, quaternions, etc.) sont indépendants du modèle du spatialisateur à proprement parler, et peuvent être d'un intérêt plus général pour la communauté d'informatique musicale.

Impulsé tant par les innovations technologiques que par les productions artistiques, spat∼ demeure un outil en constante évolution ; les principaux travaux en cours et les

perspectives à court terme concernent :

— l'intégration de formats orientés objet, tel que discuté au paragraphe 4.2 ;
— la compatibilité avec les protocoles d'interrogation du *namespace* OSC ; de nombreuses propositions ont été faites à cet égard (OSC Query[30], Minuit [12] , libmapper [13] , OSNIP [14] , OSCQueryProposal [15] ), et il n'existe pour l'heure pas de consensus dans la communauté. L'intégration de l'un ou l'autre de ces protocoles dans `spat~` 5 est en cours d'évaluation ;
— l'extension des formalismes d'analyse/synthèse de scènes sonores spatiales ;
— et l'exploitation dans des environnements de composition assistée par ordinateur.

## 6. REFERENCES


[1] Agger, S., Bresson, J., Carpentier, T., "*Landschaften* – Visualization, Control and Processing of Sounds in 3D Spaces", *Proc. of the International Computer Music Conference (ICMC)*, Shanghai, China, Oct 2017.

[2] Bresson, J., Bouche, D., Carpentier, T., Schwarz, D., Garcia, J., "Next-generation Computer-aided Composition Environment : A New Implementation of OpenMusic", *Proc. of the International Computer Music Conference (ICMC)*, Shanghai, China, Oct 2017.

[3] Bresson, J., Bouche, D., Garcia, J., Carpentier, T., Jacquemard, F., MacCallum, J., Schwarz, D., "Projet EFFICACe : Développements et perspectives en composition assistée par ordinateur", *Journées d'Informatique Musicale (JIM)*, Montreal, Canada, May 2015.

[4] Carpentier, T., "Récents développements du Spatialisateur", *Journées d'Informatique Musicale (JIM)*, Montreal, Canada, May 2015.

[5] Carpentier, T., "ToscA : Un plugin de communication OSC pour le mixage spatialisé orienté objet", *Journées d'Informatique Musicale (JIM)*, Montreal, Canada, May 2015.

[6] Carpentier, T., "ToscA : An OSC Communication Plugin for Object-Oriented Spatialization Authoring", *Proc. of the 41st International Computer Music Conference (ICMC)*, pages 368 – 371, Denton, TX, USA, Sept. 2015.

[7] Carpentier, T., "Panoramix : 3D mixing and post-production workstation", *Proc. 42nd International Computer Music Conference (ICMC)*, pages 122 – 127, Utrecht, Netherlands, Sept 2016.

[8] Carpentier, T., "A versatile workstation for the diffusion, mixing, and post-production of spatial audio", *Proc. of the Linux Audio Conference (LAC)*, Saint-Etienne, France, May 2017.

[9] Carpentier, T., "Ambisonic spatial blur", *Proc of the 142nd Convention of the Audio Engineering Society (AES)*, Berlin, Germany, May 2017.

[10] Carpentier, T., "Normalization schemes in Ambisonic : does it matter ?", *Proc of the 142nd Convention of the Audio Engineering Society (AES)*, Berlin, Germany, May 2017.

[11] Carpentier, T., "Synchronisation de données interprocessus dans les applications audio temps réel : qu'est-ce qui débloque ?", *Journées d'Informatique Musicale*, Amiens, France, May 2018.

[12] Carpentier, T., Cornuau, C., "panoramix : station de mixage et post-production 3D", *Journées d'Informatique Musicale*, pages 162 – 169, Albi, France, April 2016.

[13] Carpentier, T., Noisternig, M., Warusfel, O., "Twenty Years of Ircam Spat : Looking Back, Looking Forward", *Proc. of the 41st International Computer Music Conference (ICMC)*, pages 270 – 277, Denton, TX, USA, Sept. 2015.

[14] Daniel, J., *Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia*, Ph.D. thesis, Université de Paris VI, 2001.

[15] Epain, N., Jin, C., Zotter, F., "Ambisonic Decoding With Constant Angular Spread", *Acta Acustica united with Acustica*, volume 100, pages 928 — 936, 2014.

[16] Freed, A., MacCallum, J., Schmeder, A., "Dynamic, instance-based, object-oriented programming in Max/MSP using Open Sound Control message delegation", *Proc. of the 37th International Computer Music Conference (ICMC)*, pages 491 – 498, Huddersfield, Aug. 2011.

[17] Garcia, J., Carpentier, T., Bresson, J., "Interactive-compositional authoring of sound spatialization", *Journal of New Music Research – Special Issue on Interactive Composition*, volume 46(1), pages 74 – 86, 2017.

[18] Geier, M., Carpentier, T., Noisternig, M., Warusfel, O., "Software tools for object-based audio production using the Audio Definition Model", *Proc. of the 4th International Conference on Spatial Audio (ICSA)*, Graz, Austria, Sept 2017.

[19] ITU, "ITU-R BS.2076 (ADM Audio Definition Model)", Technical report, www.itu.int/rec/R-REC-BS.2076, 2015.

[20] Jot, J.M., "Real-time spatial processing of sounds for music, multimedia and interactive human-computer interfaces", *ACM Multimedia Systems Journal (Special issue on Audio and Multimedia)*, volume 7(1), pages 55 – 69, 1999.

[21] Jot, J.M., Chaigne, A., "Digital Delay Networks for Designing Artificial Reverberators", *Proc. of the 90th Convention of the Audio Engineering Society (AES)*, Paris, France, Feb 1991.


12 . `https://github.com/Minuit`
13 . `http://libmapper.github.io`
14 . `https://github.com/jamoma/osnip/wiki`
15 . `https://github.com/mrRay/OSCQueryProposal`

[22] Jot, J.M., Warusfel, O., “A Real-Time Spatial Sound Processor for Music and Virtual Reality Applications”, *Proc. of the International Computer Music Conference (ICMC)*, pages 294 – 295, Banff, Canada, 1995.

[23] Jullien, J.P., “Structured model for the representation and the control of room acoustic quality”, *Proc. of the 15$^{th}$ International Congress on Acoustics (ICA)*, pages 517 — 520, Trondheim, Norway, June 1995.

[24] Kahle, E., Jullien, J.P., “Subjective Listening Tests in Concert Halls : Methodology and Results”, *Proc. of the 15$^{th}$ International Congress on Acoustics (ICA)*, pages 521 – 524, Trondheim, Norway, June 1995.

[25] Kronlachner, M., Zotter, F., “Spatial transformations for the enhancement of Ambisonic recordings”, *2$^{nd}$ International Conference on Spatial Audio (ICSA)*, Erlangen, Germany, February 2014.

[26] Larman, C., *Agile and Iterative Development : A Manager’s Guide*, Addison Wesley, 2003.

[27] Puckette, M., “The Patcher”, *Proc. of the International Computer Music Conference (ICMC)*, pages 420 – 429, San Francisco, CA, USA, 1988.

[28] Ratcliff, J., *Timecode : A user’s guide, 3$^{rd}$ Edition*, Focal Press, 1999.

[29] Schmeder, A., Freed, A., Wessel, D., “Best Practices for Open Sound Control”, *Proc. of the Linux Audio Conference (LAC)*, Utrecht, Netherlands, May 2010.

[30] Schmeder, A.W., Wright, M., “A Query System for Open Sound Control (Draft Proposal)”, Technical report, Center for New Music and Audio Technology (CNMAT), UC Berkeley, 2004.

[31] Wright, M., “Open Sound Control : an enabling technology for musical networking”, *Organised Sound*, volume 10(3), pages 193 – 200, Dec 2005.

[32] Zotter, F., Frank, M., “All-Round Ambisonic Panning and Decoding”, *Journal of the Audio Engineering Society*, volume 60(10), pages 807 – 820, 2012.

[33] Zotter, F., Pomberger, H., Noisternig, M., “Energy-Preserving Ambisonic Decoding”, *Acta Acustica united with Acustica*, volume 98, pages 37 – 47, 2012.

# ÉTUDE ET APPROPRIATION DE L'AUGMENTATION INSTRUMENTALE PROPOSÉE PAR JEAN-CLAUDE RISSET DANS L'ÉLABORATION DE SES DUOS POUR UN PIANISTE

*Sébastien Clara*
Université Lyon – Saint-Étienne
sebastien.clara@univ-st-etienne.fr

**RÉSUMÉ**

En 1989, Jean-Claude Risset est invité au Media Lab du M.I.T. en tant que compositeur en résidence. Il élabore un dispositif réactif autour d'un piano reproducteur Disklavier commercialisé par la firme Yamaha. Avec cette gamme de piano, Yamaha effectuait une mise à jour au standard de la fin du XX[e] siècle du piano dit mécanique. Les pianos mécaniques génèrent de la musique automatiquement et ils ont bénéficié d'une grande popularité au début du XX[e] siècle. Le piano reproducteur est quant à lui capable d'enregistrer l'exécution d'une pièce en temps réel et de la reproduire fidèlement. Jean-Claude Risset a détourné la fonction de ce piano reproducteur pour construire un instrument augmenté et compose avec celui-ci une série de pièces qu'il intitule *Huit esquisses en duo pour un pianiste*.

Dans cet article, nous retraçons les différents développements que nous avons réalisés autour des pièces composées par Jean-Claude Risset durant cette résidence. Nous étudierons en premier l'augmentation instrumentale établie par Jean-Claude Risset pour personnaliser et varier les réponses de son automate suivant les particularités d'une interprétation. Puis, nous relaterons notre appropriation de l'augmentation de son instrument.

## 1. INTRODUCTION

En 1989, Jean-Claude Risset expérimente plusieurs modalités d'interaction entre un musicien et un automate lors de sa résidence au M.I.T. Il conclut ce travail de recherche par la composition d'une série de pièces qu'il intitule *Huit esquisses en duo pour un pianiste*. Son dispositif se compose d'un ordinateur et d'un piano acoustique, mais de type automatophone.

Un automatophone est un instrument de musique couplé à un mécanisme qui permet de simuler le jeu d'un instrumentiste. Il peut être actionné automatiquement ou par un humain. Dans l'imaginaire occidental, l'automatophone correspond à l'orgue de Barbarie des joueurs de rue ou au piano dit mécanique. La culture populaire a utilisé ces pianos comme un personnage des saloons de l'ouest sauvage des États-Unis du début du XX[e] siècle, jouant sans peur et inlassablement des Ragtimes très en vogue à ce moment-là.

Le principe du codage de la musique des automatophones voit le jour à Bagdad au IX[e] siècle. La fratrie Banu Musa met au point un automate joueur de flûte, mû par la force hydraulique. Les leviers qui obstruent les trous de la flûte sont commandés par un cylindre à picots. La mélodie est programmée suivant la disposition spatiale des picots sur le cylindre. L'écoulement de l'eau dans le système met en mouvement le cylindre. Les picots déclenchent alors la levée et la descente des leviers et modifient la note émise par la flûte [3].

De nombreux compositeurs[1] se sont approprié ces automates, mais ces machines génèrent de la musique régulière, inapte à entamer un dialogue expressif avec un musicien. Les automatophones étaient généralement utilisés pour reproduire la musique populaire ou religieuse, mis à part quelques exemples créatifs notables durant le XX[e] siècle [5].

L'essor de l'électronique pendant le XX[e] siècle a augmenté significativement l'importance de l'automatisation dans nos sociétés et a entraîné la création de nouveaux outils destinés à la production sonore, notamment le synthétiseur, le séquenceur ou l'arpégiateur[2]. Cependant, le principe de codage de la musique du cylindre à picots a été transposé[3] sur ces nouveaux instruments, avec sa rigueur métronomique jusqu'à aujourd'hui dans nos stations audionumériques.

À la fin des années 1950, Max Mathews crée le premier logiciel de synthèse sonore. Toutefois, les ordinateurs de cette période demandent plusieurs minutes pour calculer une seconde de son. Dans ces

[1] Joseph Haydn (Hob. XIX: 1–32), Wolfgang Amadeus Mozart (K. 594, 608 et 616), Ludwig van Beethoven (Op. 91, arrangé ultérieurement pour orchestre), Igor Stravinsky (Étude pour Pianola), György Ligeti (certaines études pour piano), etc. À noter que Conlon Nancarrow a consacré sa production au piano automatique.

[2] La fonction de l'arpégiateur est de transformer un accord en arpège suivant un tempo donné. Il fonctionne en temps réel et il dispose de plusieurs paramètres pour varier ses réponses. À partir des années 1980, cette fonction est régulièrement implantée dans les synthétiseurs. Aujourd'hui, elle est disponible par défaut dans la majorité des stations audionumériques ou par l'adjonction de logiciels tiers.

[3] De même pour l'industrie avec l'automatisation du métier à tisser par rubans perforés ou de l'informatique avec son système d'entrée-sortie par cartes perforées.

Sébastien Clara, *Étude et appropriation de l'augmentation instrumentale proposée par Jean-Claude Risset dans l'élaboration de ses duos pour un pianiste*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 53–61, 2018. 

conditions, l'interaction musicale entre une machine numérique et un humain est impossible. À la fin des années 1960, Mathews et Moore mettent au point le système Groove pour pallier cette carence. Cependant, Groove est un système hybride. En effet, l'ordinateur n'est toujours pas capable de calculer des sons en temps réel, mais il peut générer du contrôle en temps réel. L'ordinateur commande alors un synthétiseur analogique [9].

Les moyens technologiques limitent les possibilités artistiques et c'est à partir des années 1980 que la création de dispositifs interactifs ou réactifs [1] se démocratise. Les ordinateurs deviennent plus puissances, plus petits, plus abordables et ils ont à leur disposition de plus en plus de mémoire. La norme MIDI permet de découpler les contrôleurs et les modules de production sonore et de créer des dispositifs hétérogènes dont l'ordinateur peut devenir un maillon.

L'avènement de la norme MIDI ravive aussi l'intérêt pour les interfaces homme-machines alternatives – quête qui démarre avec les premiers instruments électroniques comme les efforts fournis par Léon Theremin, Maurice Martenot ou encore Donald Buchla, mais pour le moment aucune n'a supplanté la popularité des contrôleurs qui reproduisent la forme des instruments acoustiques ou des équipements de studio [4].

Une autre approche de contrôle d'un dispositif consiste à augmenter les capacités d'un instrument acoustique ou électronique. Au M.I.T., Tod Machover fait partie des pionniers de ce type de démarche avec son projet d'hyperinstrument.

> *The hyperinstrument project was started in 1986 with the goal of designing expanded musical instruments, using technology to give extra power and finesse to virtuosic performers. Such hyperinstruments were designed to augment guitars and keyboards, percussion and strings, and even conducting. […] The research focus of all this work is on designing computer systems (sensors, signal processing, and software) that measure and interpret human expression and feeling, as well as on exploring the appropriate modalities and innovative content*[4].

Deux modalités de contrôle sont possibles pour augmenter un instrument. La première consiste à ajouter des contrôleurs à l'instrument. Cette méthode implique l'apprentissage de nouveaux gestes de la part du musicien pour maîtriser ce type d'augmentation. L'exemple emblématique de cette méthode est l'augmentation de la guitare. L'amplification électrique a permis à cet instrument acoustique de faible intensité sonore de devenir un instrument soliste de premier plan dans l'orchestre. Mais l'électricité a aussi permis de modifier radicalement son timbre. L'insertion d'une ou plusieurs pédales d'effet entre la guitare et l'amplificateur fournit des contrôles en temps réel pour altérer le timbre par des gestes inusités par la technique instrumentale traditionnelle de cet instrument.

La seconde méthode pour réaliser un instrument augmenté est transparente pour le musicien. À partir de différents moyens techniques, un système capture une performance instrumentale classique pour s'alimenter en données et générer des réponses sonores. Jean-Claude Risset a utilisé cette dernière méthode d'augmentation pour construire son dispositif. En effet, Yamaha commercialise un nouveau type de piano augmenté de capteurs et de moteurs à partir de 1987. Le standard MIDI permet à ce piano de dialoguer avec d'autres machines équipées de ce protocole de communication. Les capteurs indiquent les touches et les pédales actionnées par un pianiste et les moteurs entraînent les touches et les pédales du piano. Un ordinateur connecté à ce type de piano peut donc « écouter » un pianiste jouer ou commander le piano.

En 1989, les champs technique et conceptuel étaient donc propices pour que Jean-Claude Risset puisse réaliser son dispositif et composer sa série de pièces. Le Disklavier de Yamaha reprend le principe du piano reproducteur élaboré au début du XX$^{e}$ siècle. Le but de ce piano est de saisir l'expressivité[5] d'une interprétation en temps réel et de la restituer le plus fidèlement possible. La norme MIDI a permis à Jean-Claude Risset de détourner la fonction de ce type de piano pour fabriquer un instrument augmenté. Dans cet article, nous nous focalisons exclusivement sur la solution développée par Jean-Claude Risset. Toutefois, des compositeurs ou des Réalisateurs en Informatique Musicale ont proposé des dispositifs instrumentaux alternatifs [6].

## 2. PRÉSENTATION DES DUOS POUR UN PIANISTE

Jean-Claude Risset compose ses différents duos au cours de deux résidences au M.I.T. La première date de 1989 et il nomme ces études *Duo pour un pianiste : huit esquisses pour piano MIDI et ordinateur*. Le titre des pièces évoque l'opération d'augmentation mise en œuvre.

> Double. *Le pianiste joue seul, puis à la reprise l'ordinateur ajoute des ornements. Ces ornements pré-enregistrés interviennent quand le pianiste joue certaines notes ; leur tempo peut être influencé par le tempo du pianiste.*

[4] http://opera.media.mit.edu/projects/hyperinstruments.html, dernière consultation le 20/03/18.

[5] Cette problématique est déjà présente au XVIII$^{e}$. Dans son traité sur *l'art de noter les cylindres*, Marie-Dominique-Joseph Engramelle signale l'écart entre la notation musicale et la pratique des musiciens et focalise son attention sur la transcription de ces détails d'expression [2].

Miroirs. *À chaque note jouée par le pianiste répond la note symétrique par rapport à certaine note du clavier – un procédé utilisé dans la seconde* Variation *op. 27 de Webern, citée au début (et aussi à la fin, à l'écrevisse). Les centres de symétrie et les retards de réponse sont variés au cours de la pièce.*

Extensions. *Aux arpèges joués par le pianiste, l'ordinateur ajoute des arpèges transposés.*

Fractals. *À chaque note jouée, l'ordinateur ajoute cinq notes espacées d'une octave altérée. Alors les mélodies jouées par le pianiste sont étrangement distordues : une montée d'une octave est perçue comme une descente d'un demi-ton.*

Agrandissements. *Comme dans Extensions, l'ordinateur ajoute des notes, mais les intervalles sont non pas transposés, mais agrandis dans des rapports allant de 1,3 à 2,7, ce qui agrandit les intervalles mélodiques et harmoniques.*

Métronomes. *Au début, l'ordinateur répond en canon, sur des hauteurs transposées et à des tempos plus rapides. Puis il joue simultanément plusieurs séquences à des tempos différents. Enfin il répète les mêmes hauteurs, mais encore à des tempos métronomiques différents, soit établis à l'avance, soit décidés par le pianiste.*

Up-down. *Des arpèges d'octaves altérées sont déclenchés par le pianiste, qui voit ses notes proliférer. Le tempo des arpèges est établi d'abord par le tempo du pianiste ; puis la note qu'il joue ; enfin par l'intensité du jeu.*

Résonances. *Au début et à la fin, l'ordinateur tient de longs accords. Dans la section médiane, le pianiste joue des accords muets : les cordes sont mises en résonance par les séquences jouées par l'ordinateur [7].*

La seconde résidence date de 1991 et Risset l'intitule *Trois études en duo*. Le principe et le dispositif sont identiques à ceux de la première résidence. Cependant, il symbolise les opérations d'augmentation utilisées dans ces études par des personnages tirés de la mythologie grecque.

Echo. *L'ordinateur fait écho au pianiste, mais il ne s'agit pas d'une simple répétition : les échos sont transposés en hauteur et en tempo, et ils interviennent avec des retards variés. Cette étude tire parti des résonances sur la table d'harmonie des notes jouées par le pianiste aussi bien que par son partenaire virtuel.*

Narcisse. *Ici la relation s'apparente à la réflexion par un miroir : les intervalles mélodiques sont renversés – une quarte devient une quinte et vice-versa. Le centre de symétrie est une note du clavier qui varie dans le courant de la pièce. La réflexion peut aussi intervenir avec un retard variable.*

Mercure. *Il s'agit d'une sorte de scherzo dans lequel le pianiste déclenche des arpèges à différentes vitesses. Le tempo des arpèges dépend soit du tempo de certains groupes de notes jouées par le pianiste, soit de la hauteur, soit de l'intensité de son jeu. Les arpèges investissent l'espace des hauteurs un peu à la façon des formes d'un kaléidoscope.*[6]

Le glissement entre le mythe d'Écho et de Narcisse et les opérations musicales de canon et de renversement est clair. Toutefois, l'invocation de Mercure est moins évidente à saisir. En effet, cette divinité est le protecteur du commerce, des voleurs, des voyages et le messager des dieux. Une technique de communication longue distance, ancêtre du sémaphore que l'on trouve durant l'Antiquité grecque, transmet une information par un signal visuel de relais en relais jusqu'à son destinataire.

> *Clytemnestre : Héphaistos, en lançant de l'Ida une flamme brillante, puis, de feu en feu, le message igné est venu jusqu'à nous. L'Ida l'a envoyé au rocher d'Hermès, à Lemnos ; de l'île, l'éclatante lumière a été captée au troisième relais, au mont Athos voué à Zeus, puis franchissant d'un bond vigoureux les crêtes de la mer, l'ardent flambeau voyageur s'est élancé avec joie <...> et la torche a transmis sa clarté dorée, soleil de nuit, au poste de guet du Makistos.*[7]

La propagation d'une note par les arpèges générés automatiquement pourrait se comparer à la propagation de l'information de la chute de Troie de la tragédie d'Eschyle. De torche en torche, ce message court au-

[6] Note de programme, http://brahms.ircam.fr/works/work/11507/#program, dernière consultation le 20/03/18.

[7] ESCHYLE, *Agamemnon*, La Différence, Paris, 2004, p. 177.

dessus des montagnes et de la mer jusqu'à Clytemnestre. Toutefois, notre proposition sur la raison de Jean-Claude Risset d'intituler cette pièce Mercure reste une hypothèse. Aucun des documents que nous avons consultés n'apporte de précision sur cette question.

Par la suite, Jean-Claude Risset fusionne les travaux produits durant ces deux résidences lorsqu'il interprète ces pièces. Le nom générique *Duo pour un pianiste* fait donc référence indistinctement aux pièces de ces deux résidences.

## 3. LES PRINCIPAUX ALGORITHMES DES DUOS

Nous avons recensé dans cette section les principales briques algorithmiques utilisées par Jean-Claude Risset pour élaborer son augmentation instrumentale. Pour effectuer ce travail, nous avons consulté les patchs Max réalisés par le compositeur. Pour ce faire, nous avons utilisé l'émulateur Basilisk II des versions 7 et 8 du système d'exploitation de Macintosh. Sous cet émulateur, nous avons utilisé Max 3 pour sauvegarder les patchs en version texte. Nous avons pu alors les ouvrir et les adapter avec Max/MSP 7 sous un système d'exploitation récent et entièrement fonctionnel.

Pour alimenter les algorithmes de ces duos, Jean-Claude Risset exploite les données générées en temps réel par le pianiste. Les renseignements transmis par une note MIDI sont sa hauteur, sa vélocité, le moment de son l'attaque et de son relâchement. Une horloge est couplée aux transitoires des notes et elle permet de déterminer la durée des notes et la durée entre les attaques des notes. Ces données peuvent être sauvegardées pour réaliser des échos par exemple ou être consommées dans l'instant.

Risset programme principalement des algorithmes qui transforment la hauteur et la durée des notes jouées par un pianiste. Mais il parallélise aussi les traitements tout en les configurant différemment. Il peut ainsi varier la densité des notes générées par l'automate. De plus, l'action de la pédale una corda du piano stoppe tous les algorithmes en cours d'exécution et oblige l'automate au silence. Nous classons ces algorithmes en trois catégories avec pour critère de discrimination les traitements réalisés sur la hauteur, le temps et les traitements conditionnels.

### 3.1. Traitement de la hauteur

- Transposition :

```
noteMidi + transpoDemiTon
```

- Renversement :

```
(pointDeSymétrie * 2) -  noteMidi
```

- Accord et arpège à partir d'une note :

Exécution parallèle de l'algorithme de transposition avec un contrôle du délai d'exécution sur chaque branche. Si les délais sont identiques, les notes seront jouées simultanément. Sinon le programme égrainera l'énoncé des notes dans le temps et produira un arpège.

- Agrandissement :

```
noteMidi * coefAgrandissement
```

### 3.2. Traitement temporel

- Agrandissement d'une durée :

```
duréeNoteMidi * coefAgrandissement
```

- Détermination d'un tempo après trois notes attaquées :

```
120 / (dateNoteOnT3 – dateNoteOnT1)
```

- Répétition d'une note :

```
Tant Que répétitionPorte est
     ouverte Faire ...
```

### 3.3. Traitement conditionnel

- Sensibilisation d'une réponse :

```
Si vélocitéNoteMidi <  vélocitéSeuil

 Alors coefAgrandissement <-- 1.3

 Sinon coefAgrandissement <-- 0.7
```

- Déclenchement d'une action suivant un groupement de notes jouées :

```
Si (noteMidi == noteMidiPivot) &&
(noteMidi-1 == noteMidiPivot-1)

 Alors déclenchement d'une action
```

Dans le vocabulaire de Jean-Claude Risset, ce traitement conditionnel est encapsulé dans l'objet : *trig2 noteMidiPivot-1 noteMidiPivot* (cf. figure 1).

## 4. MISE EN ŒUVRE DES OPÉRATIONS

La figure 1 représente la première partie du patch Max de la pièce *Miroirs* et elle nous indique la progression du paramétrage de l'opération de renversement. À l'initialisation du programme, le point de symétrie de l'opération reçoit la note MIDI 69. Le délai avant que l'automate joue les notes renversées est de 175 ms. La première note pivot (la note MIDI 27) est activée. À ce moment, toutes les notes jouées par le pianiste seront renversées suivant la note point de symétrie avec une latence d'exécution de 175 ms. Toutes les notes, sauf la note MIDI 27. Lorsque celle-ci sera jouée par le pianiste, un nouveau paramétrage

algorithmique sera mis en place. Le point de symétrie deviendra la note MIDI 85 et la prochaine note pivot sera la note MIDI 64. Le délai avant de jouer les notes renversées n'est pour le moment pas modifié.

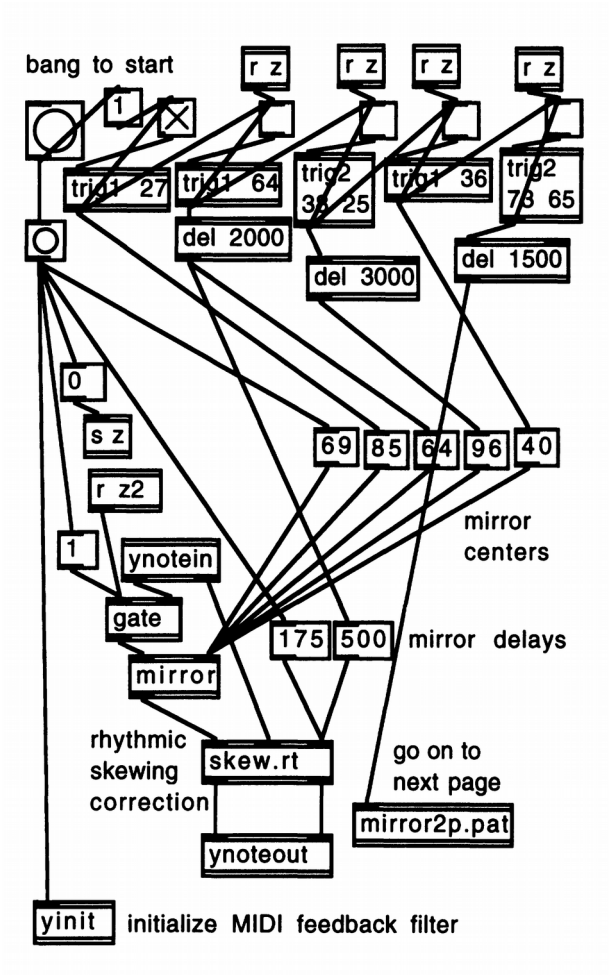



**Figure 1**. Première partie du patch Max de la pièce Miroirs [8].

La pulsation de cette pièce est de 120 à la noire. Jean-Claude Risset note uniquement les barres de mesures du premier système. Nous sommes à 4/4, mais il ne l'écrit pas. Ces premières barres de mesures montrent la carrure générale des prochains systèmes, mais l'interprète n'a pas à suivre cette contrainte. Le premier déclenchement est placé sur le premier temps de la quatrième mesure. La première modification des paramètres de l'algorithme est donc programmée six secondes après le début de la pièce. La deuxième est prévue dix secondes après cette première modification et ainsi de suite.

Dans *Miroirs*, le paramétrage de l'opération est fixé par le compositeur, mais il évolue durant toute la pièce à une fréquence assez soutenue comme nous venons de le voir. Ces modifications récurrentes altèrent la régularité des réponses générées par l'automate. Nous représentons cette réalisation par la figure 2. À l'initialisation du système, le traitement reçoit des arguments opératoires puis les notes à traiter, mais certaines notes provoquent des changements de paramètres.

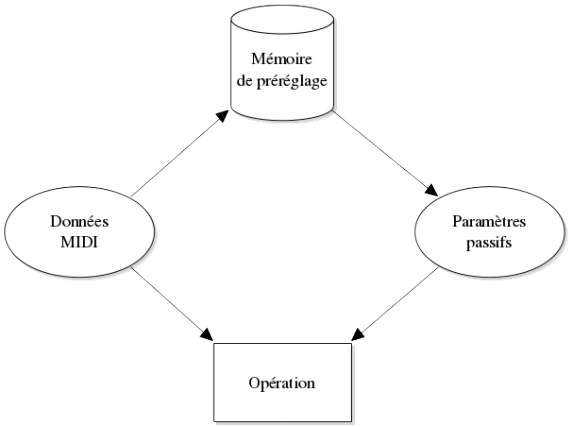



**Figure 2**. Automatisation passive, mais avec une évolution des paramètres du traitement.

D'autre part, pour rendre les réponses de l'automate actives et sensibles au jeu du pianiste, Jean-Claude Risset conditionne les valeurs du paramétrage d'un algorithme aux caractéristiques des notes jouées en temps réel. De même que pour un paramétrage passif, ces conditions évoluent durant le déroulement de la pièce. Par exemple, dans la pièce *Mercure* « le tempo des arpèges dépend soit du tempo de certains groupes de notes jouées par le pianiste, soit de la hauteur, soit de l'intensité de son jeu ».

Deux types de configurations alimentent les opérations, les arguments passifs et actifs. Mais en plus, ces deux types d'arguments évoluent durant l'exécution de la pièce. La figure 3 représente l'ensemble des possibilités de paramétrage d'une opération utilisé par Jean-Claude Risset pour élaborer la réactivité de son système.

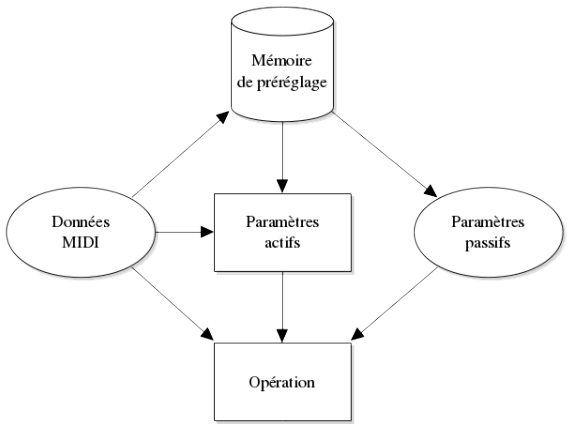



**Figure 3**. Automatisation active.

De plus, Jean-Claude Risset emploie des rapports toujours complexes entre les paramètres d'une opération afin de pulvériser la plupart des relations synchrones dans ses duos. Dans la première partie de *Miroirs* (figure 1), le rapport entre le premier délai et le deuxième avant que l'automate produise une réponse est de 20/7. Par ailleurs, le premier délai de réponse de l'automate n'est pas synchronisé au tempo du pianiste, noté 120 à la noire et provoque un écart entre la pulsation du pianiste et celle de l'automate. À la deuxième note pivot, le délai de réponse de l'automate (de la durée d'une noire) et le tempo du pianiste se synchronisent. Mais dans la seconde partie du programme, Risset restaure cette dichotomie entre les pulsations et il l'entretiendra jusqu'à la fin de la pièce.

Dans les différents duos, l'automate génère généralement des réponses asynchrones par rapport aux données produites par le pianiste. Néanmoins, les protagonistes de ces duos utilisent le même instrument. L'auditeur ne perçoit pas un dialogue ou le développement d'une opération compositionnelle, mais un monologue riche, plus ou moins dense. Dans le dispositif établi par Jean-Claude Risset, l'automate rempli la fonction d'augmentation instrumentale en accompagnant ou en amplifiant le jeu du pianiste. Pour varier le discours musical et briser la monotonie des réponses de son système, Risset fait fluctuer par différentes méthodes les relations qui lient l'automate au pianiste.

## 5. APPROPRIATION DES DUOS

Nous nous sommes approprié les principes établis par Jean-Claude Risset dans l'élaboration de ces duos pour présenter un instrument augmenté lors de l'exposition IA[2] organisée dans les locaux de l'université de Saint-Étienne pour animer les événements alternatifs (*Off*) de la Biennale Internationale Design 2017 de Saint-Étienne. Pour ce faire, nous disposions d'un piano Schimmel TwinTone. Ce modèle est un piano droit équipé d'un synthétiseur. Les entrées MIDI du synthétiseur ne contrôlent pas la partie mécanique du piano. Cette caractéristique technique implique que notre automate peut produire des sons uniquement synthétiques.

L'avantage de cette contrainte technique est que nous n'avons pas été confrontés au problème de réinjection des données dans la boucle établie entre les entrées/sorties MIDI de l'ordinateur et du piano ni aux distorsions rythmiques induites par des latences variables des événements pour commander le piano. Ces problèmes et leur solution sont discutés dans l'article rédigé par Jean-Claude Risset et Scott Van Duyne [8]. L'inconvénient de cette contrainte technique est la piètre mixité entre les parties acoustique et synthétique du piano. Pour homogénéiser le résultat sonore de notre installation, nous avons décidé d'utiliser uniquement le son du synthétiseur.

Notre augmentation instrumentale propose au public d'explorer six modes opératoires sensibles. Les notes les plus graves du piano permettent de changer de mode et l'action de la pédale *una corda* stoppe l'automate. Pour faciliter l'interaction entre le public et l'automate, nous affichons son état sur un écran. Nous utilisons ce retour visuel pour présenter les différents contrôles, stipuler et donner une explication du mode opératoire sélectionné et indiquer les notes jouées par l'automate (cf. figure 4).

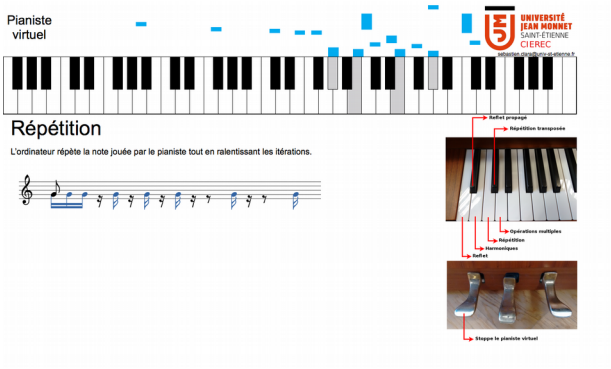



**Figure 4**. Retour visuel de l'état de notre automate.

Pour stabiliser l'expérience des utilisateurs, nous avons utilisé uniquement une configuration active pour conduire l'augmentation de notre instrument. Fixer le paramétrage du système participe à la compréhension globale des utilisateurs et accélère l'apprentissage du fonctionnement du dispositif. Sensibiliser les réponses de l'automate décuple l'expérience récréative de l'installation et procure une plus grande satisfaction lorsqu'on maîtrise musicalement cette augmentation. Pour les personnes qui possèdent aucune notion musicale, le retour visuel donne une clé d'écoute pour comprendre le principe d'un instrument augmenté.

De plus, si l'installation n'est pas utilisée un certain temps, elle émet un appel. Suivant le lieu d'exposition, nous avons implanté deux types de signaux. Le premier joue des extraits du répertoire occidental classique pour piano. La seconde solution enregistre le jeu des utilisateurs de l'installation et transforme ces données pour produire un appel.

Nous avons réalisé notre installation avec Max/MSP. Cependant, la majeure partie de notre système est développée en JavaScript dans des objets Max dédiés. Le fonctionnement asynchrone des tâches JavaScript nous a semblé approprié pour répondre à nos attentes. En effet, une tâche JavaScript est une fonction que l'on gère dans le temps. On peut lui spécifier un nombre d'exécutions, une condition d'arrêt particulière, la stopper manuellement, etc[8]. Pour ne pas saturer l'espace sonore par une accumulation des interventions de notre automate, nous avons fixé le nombre de tâche pouvant fonctionner en parallèle à trois.

[8] https://docs.cycling74.com/max5/vignettes/js/jstaskobject.html, dernière consultation le 20/03/18.

Les principales actions de notre automate sont déclenchées lors de l'attaque ou du relâchement d'une note et suivant l'événement, notre automate réagit distinctement. Selon le mode opératoire sélectionné et ses conditions d'exécution, une réponse peut être délivrée (cf. figure 5).

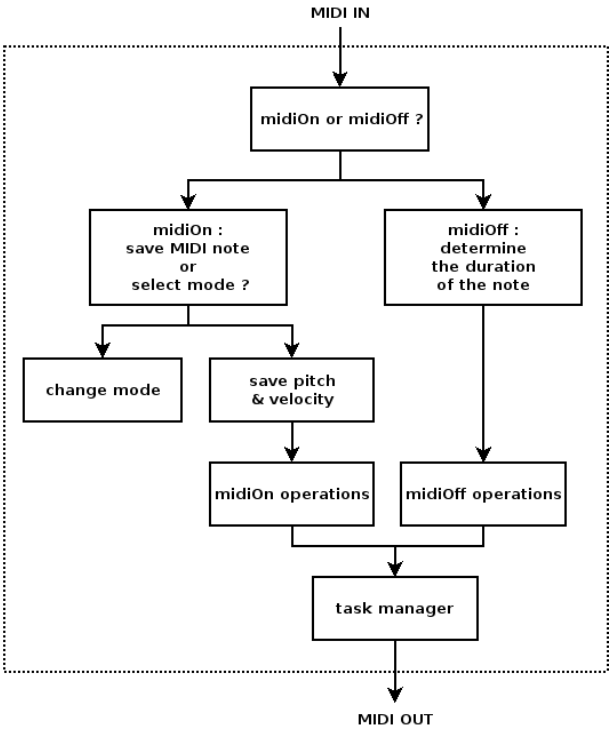



**Figure 5**. Schéma du fonctionnement de notre automate.

### 5.1. Exemple d'implémentation d'une opération

Pour traiter cet exemple d'implémentation, nous choisissons l'opération d'extension par ajout d'harmoniques. Cette opération est représentative de l'architecture de notre système. En effet, nous dédions un mode à cette opération (cf. la section 5.4). Nous l'exécutons dans ce mode lorsqu'une note est attaquée. Mais nous l'utilisons aussi dans notre dernier mode, où différentes opérations sont employées (cf. la section 5.7). Dans ce mode, nous l'exécutons au relâchement d'une note, si la durée de cette dernière est supérieure à une seconde et demi. Ces conditions d'exécution sont déterminées dans les blocs *midiOn operations* et *midiOff operations* de la figure 5.

Pour cette opération, le but de la tâche est de jouer une note de la série harmonique de la note jouée par le pianiste à chaque fois qu'elle est appelée. Par exemple, nous devons l'exécuter dix fois pour jouer les dix premières harmoniques de la note jouée par un pianiste. Une des propriétés des tâches JavaScript nous fournit le nombre de fois que la tâche a été appelée. Nous utilisons cette propriété pour déterminer le rang de l'harmonique à jouer (cf. lignes 4 et 5 de la figure 6). Nous avons décidé que l'intensité des harmoniques diminuera suivant leur rang. Toutefois, l'amplitude de cette décroissance est corrélée à la vélocité de la note jouée par le pianiste (cf. ligne 6 de la figure 6). La tâche dispose d'un contrôle pour vérifier la vraisemblance des données à émettre (cf. ligne 8 de la figure 6). Si le déroulement de cette tâche aboutit à produire des données invalides, elle met fin à son exécution (cf. ligne 11 de la figure 6).

```
1  function overtonesOpe() {
2    var note = arguments.callee.task.arguments[0][0];
3    var velocity = arguments.callee.task.arguments[0][1];
4    var coef = arguments.callee.task.iterations + 1;
5    var drift = freq2midi(midi2freq(note)*coef);
6    var amp = Math.round(velocity -
*    (arguments.callee.task.iterations*midi2data(velocity, 0.01, 3)));
7
8    if((drift<128)&&(amp>0)) {
9      outlet(0, [ drift, amp, midi2data(velocity, 1000, 150) ]);
10   } else {
11     arguments.callee.task.cancel ();
12   }
13 }
```

**Figure 6**. Tâche de l'opération d'extension par ajout d'harmoniques.

Nous gérons l'exécution de nos tâches avec un tableau. Nous l'initialisons pour qu'il contienne trois cellules, le nombre maximal de tâches pouvant fonctionner en parallèle. Nous l'utilisons comme une liste circulaire. Nous naviguons dans celui-ci avec un index que nous incrémentons de un à chaque appel d'une nouvelle tâche, modulo le nombre de cellules du tableau.

Avant d'affecter une nouvelle tâche à une cellule de notre tableau, nous nous assurons que la précédente est achevée. Si ce n'est pas le cas, nous l'interrompons (cf. ligne 2 de la figure 7, le test est réalisé dans la fonction *stopTask*). Nous affectons notre nouvelle tâche dans cette cellule pour qu'elle exécute l'opération d'extension par ajout d'harmoniques et nous la paramétrons avec les données MIDI fournies par le pianiste (cf. ligne 4 de la figure 7). Nous fixons une durée d'attente entre chaque itération de cette tâche. Toutefois, nous lions la valeur de cette durée à la vélocité de la note jouée par le pianiste (cf. ligne 5 de la figure 7). Enfin, nous exécutons et planifions quinze itérations de cette tâche (cf. ligne 6 de la figure 7). Cependant, l'exécution des quinze itérations de cette tâche sera menée à son terme seulement si les données qu'elle produit sont vraisemblables (cf. lignes 8 à 12 de la figure 6).

Nous pouvons exécuter une tâche JavaScript par différentes méthodes. Nous pouvons contrôler la vie d'une tâche par différents facteurs. Nous pouvons agir sur le fonctionnement des tâches en modifiant différentes propriétés. Cette variété d'usage favorise la flexibilité de notre système. Pour sensibiliser les réponses délivrées par notre automate, nous utilisons une mise à l'échelle classique entre les données MIDI fournies par le pianiste en temps réel et les paramètres de nos tâches. Dans les prochaines sections, nous

détaillons succinctement les différentes possibilités d'augmentation présentées au public.

```
function playOvertones(note, velocity, index) {
  stopTask(taskRun[index]);

  taskRun[index] = new Task(overtonesOpe, this, [note, velocity]);
  taskRun[index].interval = midi2data(velocity, 1300, 25);
  taskRun[index].repeat(15);
}
```

**Figure 7**. Gestion de la tâche de l'opération d'extension par ajout d'harmoniques.

### 5.2. Reflet

L'automate accompagne en miroir le pianiste. Le reflet est déterminé par le do central, point de symétrie de cette opération.

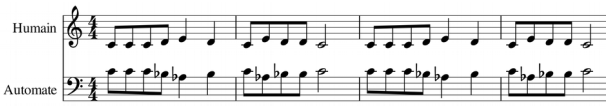



**Figure 8**. Opération de reflet.

### 5.3. Reflet propagé

L'automate accompagne en miroir le pianiste. Cependant, le reflet dérive d'octave en octave. La vitesse de la dérive est corrélée à l'intensité du jeu pianistique. Le reflet est déterminé par le do central, point de symétrie de cette opération.

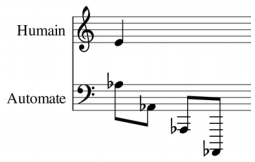



**Figure 9**. Opération de reflet propagé.

### 5.4. Harmoniques

L'automate génère la série harmonique de la note jouée par le pianiste. La vitesse d'apparition des harmoniques est corrélée à l'intensité du jeu pianistique.

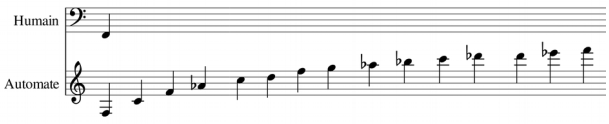



**Figure 10**. Opération d'extension par ajout d'harmoniques.

### 5.5. Répétition

L'automate répète la note jouée par le pianiste tout en ralentissant les itérations.

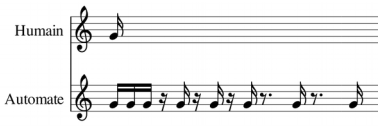



**Figure 11**. Opération de répétition.

### 5.6. Répétition transposée

L'automate répète une note plus grave que la note jouée par le pianiste tout en ralentissant les itérations. L'intervalle entre la note jouée et la note répétée est déterminé suivant l'intensité du jeu pianistique.

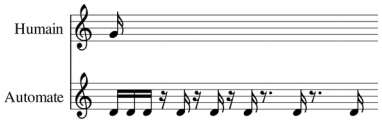



**Figure 12**. Opération de répétition transposée.

### 5.7. Opération multiple

Ce mode regroupe différentes opérations. Les échos sont exécutés si un intervalle particulier est joué et si le dispositif a enregistré plus de cinq notes.

Si l'intensité d'une note dépasse un certain seuil alors une répétition transposée est exécutée. Si la durée d'une note dépasse une seconde et demi alors l'ordinateur joue sa série harmonique. Si un intervalle d'octave est joué alors un double reflet (propagation dans les aigus et les graves) est exécuté. Si un intervalle montant de quarte est joué alors l'automate répète une section de notes jouée précédemment par le pianiste. L'écho est accéléré, rétrogradé et transposé symétriquement par rapport au do central.

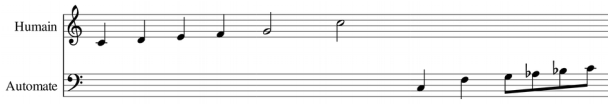



**Figure 13**. Opération rétrograde et de symétrie d'un écho.

Si un intervalle de triton est joué alors l'automate répète une section de notes jouée précédemment par le pianiste. L'écho est accéléré, rétrogradé et transposé d'un demi-ton plus grave.

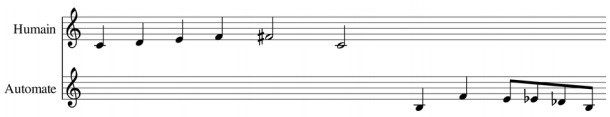



**Figure 14**. Opération rétrograde, d'accélération et de transposition d'un demi-ton plus grave d'un écho.

Si un intervalle descendant de quinte est joué alors l'automate répète une section de notes jouée précédemment par le pianiste. L'écho est juste accéléré.

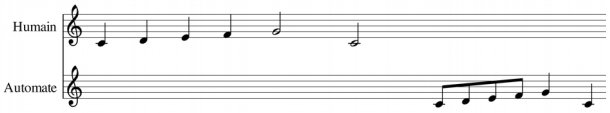



**Figure 15**. Opération d'accélération d'un écho.

Si un intervalle de sixte mineure est joué alors l'automate répète une section de notes jouée précédemment par le pianiste. L'écho est accéléré et transposé symétriquement par rapport au do central.

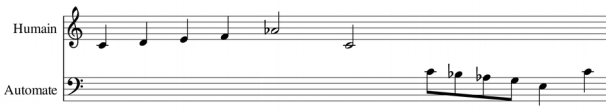



**Figure 16**. Opération d'accélération et de symétrie d'un écho.

## 6. CONCLUSION

L'objectif de notre projet de thèse est de réaliser une panoplie de captation du geste et de dispositifs de production de sons électroniques pour la musique vivante. L'étude et l'appropriation de l'augmentation instrumentale proposée par Jean-Claude Risset dans l'élaboration de ses duos pour un pianiste rentrent dans ce cadre. Suivant le même principe présenté dans cet article, nous avons programmé un arpégiateur en JavaScript dans Max/MSP. Nous avons développé aussi un certain nombre de synthétiseurs sous FAUST[9]. L'un des objectifs du langage de programmation FAUST est de pérenniser les traitements du signal et de synthèses sonores temps réel face à la volatilité des technologies et de les rendre ubiquitaires, exportables sur tous types de plates-formes, allant des extensions de différents formats à des applications pour téléphone portable.

Depuis l'introduction de la norme HTML5 et de la balise <audio>, il n'est plus nécessaire d'user d'extensions tiers pour gérer le son dans un navigateur web. De plus, la *Web Audio API* fournit aux développeurs des outils pour générer et transformer le son. En 2012, FAUST ajoute une brique à son projet pour traduire un code source DSP FAUST en fichier JavaScript. Nos développements d'augmentation instrumentale en JavaScript peuvent stimuler nos synthétiseurs FAUST dans Max/MSP, mais aussi dans un navigateur web[10]. Ces possibilités technologiques concourent à pérenniser notre travail. Nous pouvons le rendre accessible au plus grand nombre, sans qu'il soit restreint à l'usage d'un logiciel ou d'un système d'exploitation et cela pour des projets artistiques ou pédagogiques.

Nos prochains efforts se focaliseront sur la captation du geste afin d'irriguer de données nos différents modules d'augmentation et de synthèse. Nous validerons enfin nos différents développements par un travail artistique.

## 7. RÉFÉRENCES

[1] Drummond, J. "Understanding interactive systems", Organised Sound 14/2, Cambridge University Press, 2009.

[2] Engramelle, M.-D.-J. *Tonotechnie, ou l'art de noter les cylindres*, P. M. Delaguette, Paris, 1775.

[3] Koetsier, T. "On the prehistory of programmable machines: musical automata, looms, calculators", Mechanism and Machine Theory 36, 2001.

[4] Paradiso, J. A., O'Modhrain, S. "Current Trends in Electronic Music Interfaces. Guest Editors' Introduction", Journal of New Music Research 32/4, 2003.

[5] Patteson, T. W. *Instruments for New Music, Sound, Technology, and Modernism,* University of California Press, Oakland, 2016.

[6] Pottier, L. "Les musiques mixtes temps réel : pour une interprétation du son électronique en concert", colloque Soixante ans de musiques mixtes, Paris, 2012.

[7] Risset, J.-C. "Composer le son : expériences avec l'ordinateur, 1964-1989", Contrechamps n°11 – Musiques Électroniques, l'Age d'Homme, 1990.

[8] Risset, J.-C, Duyne, S. V. "Real-Time Performance Interaction with a Computer-Controlled Acoustic Piano", Computer Music Journal 20/1, 1996.

[9] Roads, C., Mathews, M. "Interview with Max Mathews", Computer Music Journal 4/4, 1980.

[9] La librairie FAUST fournit plusieurs traitements et synthétiseurs opérationnels : http://faust.grame.fr/library.html.

[10] Nous avons réalisé une transposition de cette installation pour fonctionner dans un navigateur web pour la communication de Pottier, L. « Jean-Claude Risset : autour de la synthèse sonore, de ses œuvres, de la façon de les (re)présenter graphiquement, interactivement, sur le WEB », Rencontres internationales du Collegium Musicæ, Jean-Claude Risset : interdisciplinarités, Paris, 2018.

# OUTILS INFORMATIQUES ET ANALYSE MUSICALE : REPRÉSENTER LE ROUTING DE *JUPITER*

*Maxence Larrieu*
LISAA
Université Paris-Est Marne-la-Vallée
maxence@larri.eu
orcid.org/0000-0002-1834-3007

**RÉSUMÉ**

Cet article traite du routing de la pièce *Jupiter* (1987), pour flûte et électronique en temps-réel, composée par Philippe Manoury avec l'assistance de Miller Puckette. Le routing de cette pièce, l'interconnexion entre les modules audio, pose un problème particulier pour l'analyste qui souhaite s'appuyer sur le « document électronique de la pièce », le *patch*. Le problème tient dans le fait qu'il n'est pas possible de savoir quels sont les modules actifs et leur embranchement à un passage donné. Dans le cadre de notre thèse nous avons été confrontés à ce problème et nous avons développé une solution *ad hoc,* laquelle est l'objet du présent article. Nous avons développé des programmes informatiques qui simulent le fonctionnement du patch, et cela exclusivement à partir des *qlist* – les fichiers alphanumériques qui intègrent les données de synthèse. De cette simulation nous avons obtenu des représentations qui rendent compte de l'état du routing sur toute la durée de la pièce. Après avoir présenté la pièce et ces programmes informatiques, nous expliquons comment nous avons utilisé ces résultats dans le cadre de notre analyse musicale.

## 1. INTRODUCTION

*Jupiter*, pour flûte et électronique en temps-réel, est une pièce pionnière du répertoire d'informatique musicale. Elle ouvre en effet le célèbre cycle *Sonus Ex Machina*, avec lequel Philippe Manoury a développé les premières réflexions d'une « musique en temps-réel » [7][1]. Composée dans les années 1985 avec l'aide de Miller Puckette, elle est créée en 1987 par Pierre-André Valade dans l'espace de projection de l'Ircam, lors d'un concert qui célèbre les 10 ans de l'institut[2].

Dans notre doctorat portant sur l'analyse des musiques d'informatique [4], nous avons travaillé conséquemment sur cette pièce. Celle-ci a notamment été choisie pour ses possibilités d'interconnexion entre modules, ce que nous désignons avec le terme *routing*, emprunté à l'ingénierie audio. En effet la composition de *Jupiter* se fait avec la possibilité de configurer un réseau d'une dizaine de modules tout au long de la pièce. Cette possibilité engendre néanmoins un contre-coup important : même avec le patch « sous les yeux », il n'est pas possible pour l'analyste de définir précisément quels sont les modules actifs à un passage donné. Ce problème n'est pas nouveau pour la communauté d'informatique musicale puisqu'il a été pointé en 2013 par Alain Bonardi sur une autre pièce du compositeur, *En Écho* (2003), dans le cadre d'un travail de reconstruction (cf. [1] et [2]).

Afin de mener à bien notre analyse nous avons dû trouver une solution à ce problème de routing. L'idée a été de simuler le fonctionnement du patch à l'aide uniquement des fichiers *qlist* et, durant cette simulation, d'en extraire les informations qui nous sont nécessaires. Nous avons conçu des programmes informatiques *ad hoc* qui scannent les 12 fichiers *qlist* de l'archive et qui simulent, à l'aide d'une forte compréhension du patch[3], le fonctionnement du routing. De cette simulation nous avons ensuite extrait des *chaînes de routing,* des interconnections entre modules, avec lesquelles il nous a été possible de répondre à la question : quels sont les modules qui produisent ce qui est perçu ? De là, nous avons pu réaliser notre analyse de *Jupiter* en croisant l'écoute et les connaissances acquises sur la production.

Dans cet article nous présentons le routing de *Jupiter* et le problème qui se pose à l'analyste, puis nous décrivons les trois programmes informatiques réalisés qui permettent de faire face à ce problème. Enfin nous présentons l'usage que nous avons fait des résultats dans le cadre de notre analyse musicale.

[1] Pour une introduction sur *Jupiter* voir [4], [6] et [8].

[2] Pour ses 40 ans en 2017, l'Ircam présente et diffuse les pièces de ce concert sur son site web (https://www.ircam.fr/article/detail/sons-dessus-dessous-28-retour-sur-le-10e-anniversaire/, 18 janvier 2018)

[3] Cette compréhension a abouti par exemple à une table qui décrit toutes les varibles du patch (242). cf. datapipes.okfnlabs.org/csv/html/?url=https://raw.githubusercontent.com/Podatus/Jupiter/master/jupiterVars/jupiterVars.csv, (visité le 30 mars 2018).

Maxence Larrieu, *Outils informatiques et analyse musicale : représenter le routing de Jupiter*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 63–70, 2018. 

## 2. MODULES ET ROUTING DE *JUPITER*

### 2.1. Le patch, les modules

Pour analyser *Jupiter* nous avons utilisé un patch Pure Data[4], diffusé dans un article ambitieux qui a fait date, *New Public-Domain Realizations of Standard Pieces for Instruments and Live Electronics* de Miller Puckette [9]. Dans ce patch nous pouvons différencier plusieurs modules audio, dont les noms sont, pour la plupart, familiers. Il y a d'abord des modules de synthèse : Sampler, Additive, Chapo, Paf[5] ; puis des modules de traitements : Harmonizer, Frequency Shifter, Reverb, Noise et un Spatialisateur. La table suivante donne un aperçu de ces modules avec différentes données que nous utilisons par la suite.

| Module | Type | Groupe | Nb signaux | Abbréviation |
|---|---|---|---|---|
| Flûte | synth | | 1 | d |
| Sampler | synth | | 4 | t |
| Additive | synth | osc | 4 | o |
| Chapo | synth | osc | 4 | o |
| Paf | synth | osc | 1 | o |
| Harmonizer | treat | | 1 | h |
| FreqShift | treat | | 2 | f |
| Reverb | treat | | 2 | r |
| Noise | treat | | 1 | n |
| Spat | | | 4 | 4 |

**Table 1**. Données sur les modules du routing[6]

### 2.2. Le routing

*Jupiter* est une pièce singulière notamment pour le routing : il est flexible et la composition se fait en partie par sa définition dans le temps de la pièce. Cette possibilité de configurer à loisir le routing est permise d'abord par le « suiveur de partition ». Centrale à *Jupiter* – et plus largement à la musique en temps-réel – il permet d'articuler des événements électroniques avec l'interprétation qu'effectue le flûtiste. De plus, la flexibilité du routing s'explique par son implémentation : les 5 modules de synthèse sont tous reliés aux 4 modules de traitement, et parallèlement ces derniers sont tous reliés entre eux ; enfin, tous ces modules sont reliés au Spat. La configuration du routing passe alors par un ensemble de variables qui permet de doser la quantité de signal (en valeur MIDI) dans chacune de ces connexions. Cette flexibilité a un prix : une configuration totale du routing nécessite en théorie[7] 41 variables[8].

## 3. LE PROBLÈME DU ROUTING

### 3.1. Aperçu

Nous pouvons présenter le problème en imaginant la situation suivante : soit un musicologue qui souhaite analyser *Jupiter* à l'aide (notamment) du patch. Dans le travail à réaliser nous pouvons différencier deux étapes : une où il va effectuer des lectures du patch – de sorte à identifier et comprendre les 9 modules implémentés ; et une autre où il va se confronter à la manifestation sonore de la pièce. Le problème survient dans la seconde étape : confronté au sonore de la pièce il va avoir besoin de savoir quels sont les modules qui produisent le signal perçu ; quels sont ceux qui expliquent ce qui est écouté ? Or, comme nous allons le voir, il n'est pas possible de répondre directement à cette question.

### 3.2. Utilisation du routing

Le compositeur a utilisé le routing dans le temps de la pièce, « au fil des événements », et non de configuration type en configuration type. Par exemple le Frequency Shifter (FS par la suite) est souvent utilisé pour venir perturber un « tapis » maintenu par le Paf et/ou la Reverb (*e.g.* le début de la section IV) : sur un passage le compositeur va placer le FS sur la production de ces derniers modules puis quelques événements après cette connexion va être supprimée, puis, encore quelques événements après, il va par exemple activer l'envoi de la Reverb vers l'Harmonizer, puis de ce dernier vers le FS, etc. Avec cet exemple nous souhaitons montrer que le routing possède une écriture *persistante* : une connexion reste dans son état tant qu'elle n'est pas modifiée (ré-écrite). Ainsi, l'analyste qui souhaite savoir la configuration du routing à tel événement devra remonter la vingtaine de variables[9] du routing. Ceci reste envisageable dans les premières minutes, mais devient impossible sur la demi-heure de la pièce.

Le problème auquel est confronté l'analyste vient d'une part des 21 variables utilisées pour configurer le routing, et de l'autre de leur utilisation intensive ; la flexibilité du routing appelle en contre-partie une *difficulté de représentation.*

[4] Ce patch est couramment nommé « la version de Miller Puckette » par la communauté. Soulignons que Miller Puckette comme Philippe Manoury nous ont assuré que ce patch a été utilisé en concert.

[5] Nous utilisons des majuscules afin de différencier ces modules des méthodes de synthèse qu'ils sous-tendent.

[6] Le lecteur spécialiste aura remarqué que nous n'intégrons pas le module de traitement nommé phasor. En effet celui-ci n'est pas relié au routing : il est appliqué tout au long de la pièce sur la flûte et diffusé dans la stéréophonie.

[7] En théorie, car tous ce que contient le patch n'est pas nécessairement utilisé.

[8] 5 modules de synthèse reliés aux 4 modules de traitements ; 4 modules de traitement reliés entre eux ; et les 9 modules reliés au Spat.

[9] Toutes les connexions entre modules ne sont pas utilisées par le compositeur, nous avons ainsi trouvé dans les *qlist* l'usage de 21 variables dédiées au routing.

## 4. UNE SOLUTION

Puisque le patch et les *qlist* sont seules responsables de l'électronique, de ce qui est perçu, il doit être possible de rendre compte de ces embranchements entre modules. Une telle prise de conscience est par exemple présente chez Alain Bonardi, dans le cadre d'un travail de reconstruction de la pièce *En Echo* (1993). Le patch est exécuté, écouté, et les connexions entre modules sont représentées dans le patch, graphiquement, à l'aide d'un « diagramme fonctionnel » [2]. Cette solution convient, mais elle ne nous permet pas de connaître, hors temps, des connexions présentes à un événement donné ; elle n'amène pas un support qui rende compte du routing en dehors du temps de la pièce.

Nous présentons ci-après une solution voisine où le patch à été simulé uniquement à partir des *qlist* – *i.e.* sans computation audio – et d'un ensemble de règles. Cette solution est faite de trois étapes : la première relève les pré-états des modules ; la deuxième les connexions prescrites ; la dernière utilise les précédents résultats et en déduit les connexions effectives entre modules.

Précisons avant de commencer que nous avons travaillé avec une précision à l'échelle des événements, sans tenir compte des quelques occurrences de connexions qui se font avec des temps de lissage.

Aussi, les programmes que nous avons conçus se basent sur un typage des modules, ceux de synthèse et ceux de traitement. Afin de ne pas alourdir la suite de notre écrit nous utilisons les expressions « les *synth »* et « les *treat* » pour désigner réciproquement les modules de synthèse, qui produisent des signaux, et ceux de traitement, qui les modifient.

Enfin, pour plus de clarté, nous avons été amenés à différencier le routing *interne* de celui *externe* ; le premier fait référence aux connexions entre modules, le second aux connexions vers le système de diffusion – une stéréophonie classique au-devant de la scène et une quadriphonie qui entoure le public.

### 4.1. Les Pré-états des modules

Cette première étape permet de savoir si les modules sont actifs ou non. Mais il ne peut s'agir ici que d'un état premier (pré-état), puisque avant de pouvoir connaître l'état (final) il est nécessaire de connaître l'acheminement du signal à la diffusion[10].

Pour les *synth,* cette étape consiste à savoir s'ils produisent du signal ; pour les *treat,* s'ils sont aptes à en produire.

Pour les *synth* et à l'exception du Paf, nous avons procédé à une détection manuelle à l'aide des *qlist*, de la partition et de l'écoute ; pour les *treat*, à une détection automatique avec un programme informatique.

[10] Comme l'a montré Alain Bonardi [2] il est en effet possible que des embranchements n'aboutissent pas – comme une conséquence de la difficulté de représentation du routing.

#### *4.1.1. Détection manuelle*

Pour le Sampler nous pouvons aisément différencier deux utilisations. Une à partir d'écriture dans les *qlist* avec la variable *addsamp* et une autre à partir de lecture de fichiers (*e.g. ost6-start*) où les données sont lues et affectées aux modules de synthèse – à l'instar d'un lecteur de données MIDI. Pour la première utilisation nous avons lu les *qlist* et appliqué la règle suivante : si une des voies est active durant l'événement alors le module est actif, et inversement. Le Sampler est en effet particulier puisque le signal produit est nécessairement limité dans le temps. Pour la seconde utilisation nous avons eu recours à la partition et à l'écoute. La détection est aisée puisque cet usage du Sampler est très marqué, il correspond en effet aux *ostinatos*, éléments très saillants de la pièce.

Pour les modules Additive et Chapo nous avons aussi lu les *qlist*. Simplement, à la différence du Sampler ils sont inactifs quand *toutes* leurs voies possèdent une amplitude nulle.

#### *4.1.2. Détection automatique*

Pour le Paf et les *treat* nous avons réalisé un programme – *a_preStateOfMods.pde*[11] – qui déduit automatiquement les états à partir des *qlist*. Cette déduction est faite à chaque événement, qu'il y ait ou non de nouvelles écritures. La figure ci-dessous illustre le fonctionnement du programme : il reçoit en entrée tous les fichiers *qlist,* puis, après application de règles, il construit une table – nommée *preStatesOfMods.tsv* – qui représente sous forme booléenne les états des modules, événement par événement.

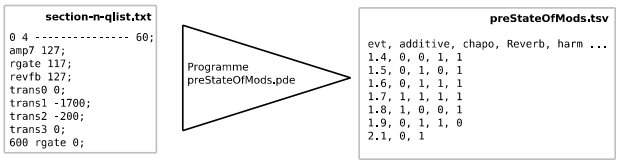



**Figure 1**. Illustration du programme de détection des pré-états des modules.

Tous les modules à traiter possèdent une structure de données commune (*e.g.* nom, type, état, variables). En utilisant la Programmation Orientée Objet (POO par suite) nous avons alors créé une classe, nommée Module, qui définit cette structure de données. Une seconde classe attachée à la précédente est aussi nécessaire. Nommée DictList elle a pour fonction de relier une variable à une liste de nombres ; par exemple la chaîne de caractère *fpos* à toutes les occurrences de cette variable.

[11] Nos programmes sont conçus avec le langage Processing. Ils sont disponibles avec les résultats à l'adresse suivante : github.com/Podatus/Jupiter (visité le 30 mars 2018).

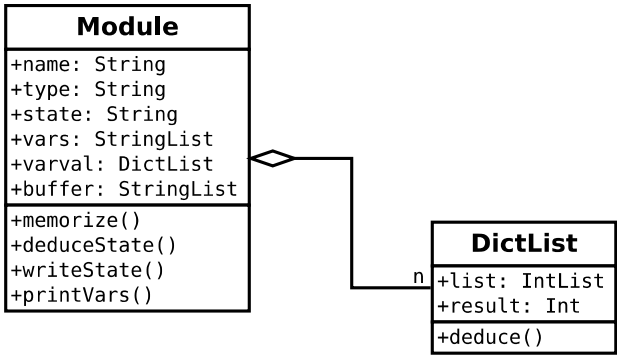



**Figure 2**. Schéma UML[12] de la structuration du premier programme.

Le programme s'initie en créant une instance de la classe Module par modules à analyser. Le *n* du précédent schéma précise que le nombre de variables attachés aux modules n'est pas fixe, comme en témoigne la table ci-dessous.

| Module | Type | Variables |
|---|---|---|
| paf | synth | amp1, amp2, amp3, amp4, amp5, amp6, amp7, amp8 |
| reverb | treat | rout |
| harm | treat | hamp1, hamp2, ham3, ham4 |
| freqshift | treat | fpos, fneg |

**Table 2**. Données des modules utilisées pour les instances du premier programme.

Les états des modules sont alors déduits à partir de trois règles, que nous pouvons ainsi présenter : si durant un événement la variable de sortie d'un Module

- a eu au moins une occurrence supérieure à 0, alors le module est actif ;
- n'a eu que des occurrences égales à 0, alors le module est inactif ;
- n'a eu aucune occurrence, alors le module reste dans l'état précédent.

Dans cette simulation les difficultés rencontrées viennent des variables de substitutions, *i.e.* des variables de plus haut niveau qui affectent un ensemble de variables, par exemple *amptutti6* affecte l'amplitude des 6 premières voies du Paf, *amp1, amp2 … amp6.*

### 4.2. Extraire le routing

La deuxième étape extrait les données de routing, interne comme externe. Si la première étape permettait de répondre à la question « quels modules produisent (au niveau du patch) du signal ? », la deuxième permet de répondre à la suivante « quelles sont les connexions présentes entre modules ?».

[12] Unified Modelling Language

Cette étape est réalisée automatiquement à l'aide d'un programme – *b_modsConnections.pde* – assez similaire au précédent. C'est en effet la même structuration en POO qui est utilisée, mais cette fois-ci avec une instanciation pour tous les modules, la flûte comprise, et bien évidemment avec toutes les variables de routing comme le présente la table ci-dessous.

| Module | Type | Variables |
|---|---|---|
| flute | synth | dtor, dtoh, dtof, dton, dto4 |
| sampler | treat | ttor, ttoh, ttof, tton, tto4, tto2 |
| osc | treat | otor, otoh, otof, oton, oto4, oto2, noise1, noise2, noise3, noise4, noise5 |
| reverb | treat | rtoh, rtof, rton, rto4, rto2 |
| harm | treat | htor, htof, hton, hto4, hto2 |
| freqshift | treat | ftor, ftoh, fton, fto4, fto2 |
| noise | treat | ntor, ntoh, ntof, nto4, nto2 |

**Table 3**. Données des modules utilisées pour créer les instances du deuxième programme.

Pour chaque événement le programme extrait les données du routing puis déduit les connexions entre modules. Le routing est ensuite représenté à l'aide d'une table – *modsConnections.tsv* –, avec une ligne pour chaque événement et une colonne pour chacun des modules. Enfin, le routing est indiqué à partir des envois. Par exemple, si la Reverb est connectée à l'Harmonizer à tel événement le programme inscrit *rtoh* dans la colonne Reverb à l'événement correspondant.

Dans cette étape les difficultés rencontrées viennent du groupement de module « osc » – pour oscillateur – qui regroupe Paf, Additive et Chapo. Une difficulté supplémentaire se présente à cause de la connexion du Paf vers le Noise, qui se fait à la fois à l'échelle du groupement de module – la variable *oton* – et à l'échelle des canaux du Paf – les variables *noise1, noise2 … noise5.* Enfin, il y a une variable de substitution à considérer qui affecte et le Sampler et le groupement de module « osc », *ston* affecte *tton* et *oton.*

### 4.3. Déduire les états des modules et le routing

La dernière étape permet de répondre à la question initiale, « quels sont les modules actifs à tel événement et quels sont leurs connexions ? », soit de rendre compte de la production. La réponse est obtenue intégralement à l'aide d'un programme – *c_stateAndRouting.pde*. Il croise les résultats des précédentes étapes et déduit, à l'aide de règles, les connexions et les états des modules. C'est une étape conséquente puisqu'il nous a été nécessaire de concevoir un algorithme qui traverse le réseau du routing.

### 4.3.1. *Fonctionnement*

Le programme utilise de nouveau la POO avec une classe Module, qui définit la structure de données et les méthodes communes aux instances.

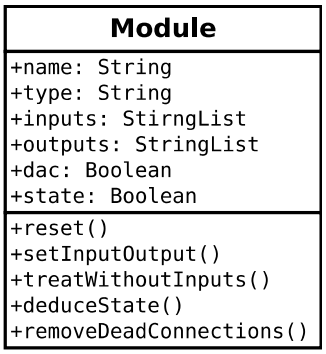



**Figure 3**. La classe Module du troisième programme.

Les attributs *inputs* et *outputs* permettent de stocker les différentes connexions des modules ; les attributs booléens *dac* et *state* de stocker la connexion à la diffusion et l'état du module.

Le programme crée une instance pour les 8 modules (Flûte, Sampler, Osc, Reverb, Harm, FreqShift, Noise et Spat) puis croise, événement par événement, les résultats des précédentes étapes. Les modules actifs sont d'abord sélectionnés – la table *preSatesOfMods.tsv* – puis les connexions entre modules sont affectées aux attributs *inputs* et *outputs* des modules correspondants. Par exemple, la lecture de *rtoh* dans la table *modsConnections.tsv* à un événement donné ajoute le terme *harm* à l'attribut *outputs* de l'instance *reverb* et le terme *reverb* à l'attribut *intput* de l'instance *harm*. De cette façon, il est possible d'avoir une représentation exhaustive du routing interne. Par exemple, dans la conception du programme il nous a été nécessaire d'avoir un retour sur l'état du routing ; nous avons opté pour la représentation suivante :

```
flute out(reverb)
noise in(osc,reverb) out(spat)
spat in(noise,osc,reverb)
osc out(noise,spat)
reverb in(flute) out(noise,spat)
```

### 4.3.2. *Règles*

Une fois les connexions établies deux règles ont été appliquées afin d'obtenir un routing cohérent :

- Si un module n'a aucune sortie, ni *dac* ni routing interne, alors il est inactif ;
- Si un module, hormis la Reverb[13], est un *treat* et n'a aucune entrée, alors il est inactif[14].

Ensuite, les états finaux des modules sont déduits – la méthode *deduceState()* – et les connexions mortes sont retirées – la méthode *removeDeadConnections()*.

### 4.3.3. *Chaînes de routing et algorithme*

La précédente représentation du routing est nettement trop lourde pour rendre compte du routing sur la demi-heure de la pièce. Afin de l'alléger nous avons pensé le routing à partir de chaînes, d'interconnexions entre modules. Une chaîne commence nécessairement par un *synth* et finit soit dans un *dac* soit dans le Spat. La transformation de la précédente représentation du routing à une nouvelle, avec ce principe de chaînes, a été réalisée par un algorithme. Ce dernier doit, tout en mémorisant leurs noms et leurs successions, traverser toutes les instances actives interconnectées jusqu'à arriver à une connexion au *dac* ou au Spat. La représentation finale que nous avons choisie est la suivante, avec en haut le numéro de la section et l'événement, puis les chaînes de routing, et enfin les modules reliés au *dac*.

```
10.12
flute>spat
sampler>reverb
sampler>harm>noise>spat
sampler>harm>reverb
dac : harm,reverb,sampler
```

Cette représentation est elle-même utilisée dans le programme pour comparer le routing d'un événement à l'autre afin de savoir s'il y a eu un changement ou non. Le fichier final – *routingChains.txt*[15] – de notre solution contient donc cette représentation pour chaque changement de routing. Avec un saut de ligne avant les événements, il contient environ 1700 lignes.

### 4.3.4. *Difficultés*

La principale difficulté a été de concevoir l'algorithme. Celui-ci n'est pas évident puisqu'il doit être récursif : le chemin à parcourir d'instance en instance n'est pas connu à l'avance mais se dessine au fur et à mesure de la traversée des instances. Dans le précédent exemple une fois la chaîne *sampler>reverb* trouvée, il faut remonter au Sampler pour investir l'autre connexion qui va à l'Harmonizer, et ainsi de suite. Lorsque l'algorithme se trouve à une instance donnée, toutes les connexions sont donc à placer dans un tampon, et de même pour le chemin qui a mené à cette instance.

[13] La Reverb possède en effet un mode infini avec lequel du signal est produit même s'il n'y a rien en entrée.

[14] La méthode *treatWithoutInputs()*

[15] cf. github.com/Podatus/Jupiter/blob/master/routing/c_statesAndRouting/routingChains.txt, (visité le 30 mars 2018).

## 5. UTILISATION DES RÉSULTATS POUR L'ANALYSE MUSICALE

Nous présentons ci-après l'usage que nous avons fait des précédents résultats lors de l'analyse musicale. Ces résultats nous ont servi à deux reprises. D'abord lors des premiers contacts avec le sonore de la pièce, de sorte à avoir un aperçu global, et ensuite, plus substantiellement, de sorte à expliquer ce que nous percevons.

### 5.1. Les états des modules

La première utilisation concerne exclusivement les états des Modules. Dans différentes analyses écrites de *Jupiter* nous avons constaté une pratique récurrente, celle d'éclairer les 30 minutes de la pièce avec l'enchaînement des modules actifs. Une représentation classique en deux dimensions est ainsi souvent donnée : la durée est présente sur l'horizontale avec les 13 sections, et les modules sont placés verticalement. L'éclairage consiste à donner les modules qui sont actifs pour les différentes sections (*e.g.* Andrew May [8]). Évidemment, avec notre précédent fichier il est possible d'aller plus loin : nous avons d'une part une discrétisation temporelle plus fine, celle des événements, et d'autre part nous avons une indication des modules actifs qui est basée sur une simulation du patch, qui dépasse donc l'écoute.

Nous avons donc prolongé la précédente représentation avec nos résultats. Le temps est présent en abscisse selon la succession des événements et non des minutes, et nous avons ajouté les segmentations de la partition : dessous le numéro des événements sont présents les lettres des sous-sections puis, dessous, le numéro des sections. Représenter plus de 500 événements demande cependant une taille conséquente ; nous plaçons donc ci-après un extrait de cette représentation, et nous la diffusons en pleine largeur sur le Web[16].

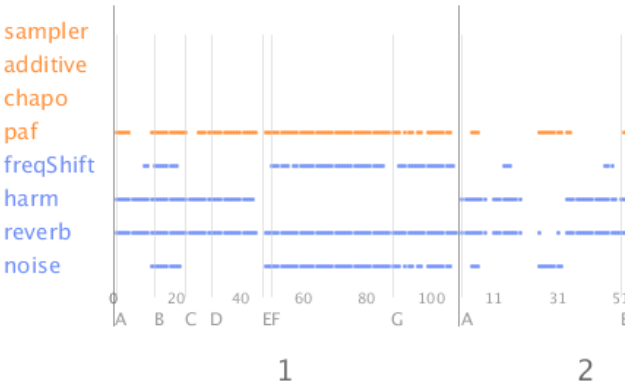



**Figure 4**. Extrait de la représentation des états des Modules.

[16] DOI : 10.6084/m9.figshare.5817459.v1

Avec la version intégrale nous remarquons de prime abord que les modules de traitement sont amplement utilisés. De plus, accompagnée d'une écoute cette représentation permet d'effectuer deux groupements de sections : la 6 et la 12, avec l'utilisation du Chapo, et un autre groupe plus vaste 1, 2, 4, 7 et 13 où l'on trouve les même modules.

Il faut cependant rester très prudent avec une telle représentation[17]. Il est en effet primordial de la relier à l'écoute et/ou à la partition, sans quoi il serait possible d'emprunter des chemins de compréhension déviants.

### 5.2. Le routing

Enfin notre analyse s'est fortement appuyée sur les chaînes de routing précédemment identifiées. Au fil de la lecture de ce fichier et de la pratique de notre analyse nous nous sommes aperçus que la précision obtenue était trop grande comparée à l'usage du routing qui a été fait dans la composition. C'est typique par exemple pour le Noise qui n'est pas un module « support de la composition », mais davantage un module de mixage : il vient brouiller un signal qui est déjà diffusé. Cela se retrouve aussi dans les nombreuses connexions des modules au *dac*. On s'aperçoit qu'il y a un *usage* récurrent des *treat* qui consiste moins à créer une nouvelle sonorité qu'a venir enrichir, brouiller, une déjà diffusée. Cet usage-là doit être dépassé par l'analyse musicale. Une analyse qui se baserait sur les connexions entre le Noise et la Reverb (en dehors du mode infini) et le FS resterait en effet lacunaire. Dans le même temps, et c'est fondamental, cet usage-là participe à l'empreinte sonore de *Jupiter*. Les ignorer serait une erreur, comme en atteste leur forte présence sur la durée de la pièce. Finalement, cet usage des *treat* s'éclaire bien avec une métaphore que le compositeur nous a transmise : « les modules de synthèse de l'époque sont comme des autoroutes, droites et monotones ; les modules de traitements permettent de faire face à cette uniformité de sorte que l'on ne s'ennuie pas dans l'écoute »[18].

Le fichier texte avec les chaînes de routing nous a ainsi été utile avec une lecture experte, de sorte à dépasser cette trop grande quantité, en relevant dans les changements ceux qui sont importants de ceux qui ne le sont pas. Avec ce fichier, nous avons pu éclairer des sonorités très surprenantes, comme l'événement 14 de la section 12, qui s'expliquent, en plus de la configuration des modules, par le routing, comme le présente la figure ci-dessous.

[17] Pour l'anecdote, on se souviendra que la première réaction de Philippe Manoury quand nous lui avions présenté cette représentation graphique fût « qu'est-ce que vous faite de cela ? »

[18] Métaphore retranscrite lors d'un entretien avec le compositeur à l'Ircam en novembre 2016.

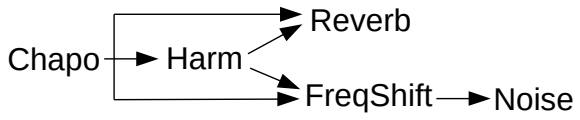



**Figure 5**. Exemple d'une configuration du routing. Événement 14, section 12.

## 6. DISCUSSION

À l'issue de ce travail se pose la question de l'adaptabilité de notre démarche à d'autres pièces. Précisons d'abord que les programmes développés ne sont pour l'heure fonctionnels que sur la pièce *Jupiter*, ils ont en effet été construits avec toutes les idiosyncrasies du patch[19], de sorte justement à les dépasser. Les pièces qui demanderaient le moins de remaniement des programmes sont bien évidemment celles du cycle *Sonus Ex Machina*, car composée par les mêmes personnes. Ensuite, en dehors d'une application rapide sur d'autres pièces, il nous semble qu'il y a dans ce travail une approche originale que nous pouvons éclairer ci-après mais qui mériterait d'être approfondie ultérieurement. Notre approche nous paraît singulière dans la mesure où nous utilisons les *qlist* dans le but d'éclairer la pièce dans sa durée, c'est-à-dire au-delà de la microstructure. Tous les *qlist* ont en effet été utilisés de sorte à éclairer – dans cet article – les embranchements des modules. Partir du postulat que toutes les informations relatives à l'électronique, sur la durée de la pièce, sont présentes dans le patch et les *qlist* est peut-être évident, en revanche, ce qui l'est moins, c'est d'aller vers cette densité et de l'exploiter dans une perspective d'analyse musicale : les *qlist* se situent à un bas niveau d'abstraction, 8000 lignes : utiliser cette quantité nécessite un pas conceptuel. Nous avons franchi celui-ci en deux temps : d'abord une forte compréhension de tous les modules et du fonctionnement du patch, ensuite une représentation de ces modules (POO dans notre cas) et du fonctionnement du patch dans des programmes informatiques.
Notre approche nous paraît donc singulière puisqu'elle va exploiter, à l'aide de représentations informatiques sur mesures, les données bas niveau présentes dans les *qlist*, dans le but d'en extraire des informations de plus haut niveau, éclairantes pour l'analyse musicale sur la durée de la pièce.

## 7. CONCLUSION

Dans cet article nous avons traité du problème du routing qui survient dans l'analyse musicale de *Jupiter*. Le problème tient dans le fait qu'il n'est pas possible pour l'analyste de connaître précisément les modules responsables de la production électronique à un passage donné. Nous avons expliqué la solution que nous avons réalisée dans le cadre de notre thèse [4]. Nous sommes arrivés à une représentation exhaustive et hors temps du routing, sans laquelle nous n'aurions pu développer notre analyse musicale. Les documents obtenus, confrontés à l'écoute, nous ont permis de relever une spécificité de la pièce : son empreinte sonore ne s'explique pas par un ou deux modules de traitement, mais par un réseau non linéaire de modules – dans le sens où des modules intermédiaires diffusent directement dans le *dac* et/ou le Spat –, qui évolue au fil des événements.

Enfin, la méthode que nous avons réalisée pour affronter le problème du routing nous paraît aussi originale pour l'analyse musicale. En effet avec une forte compréhension du patch nous avons pu simuler son fonctionnement, et cela en dehors de la computation audio. C'est une pratique originale puisque d'une part elle aboutit à un document qui est en dehors du temps de l'actualisation de la musique – ce qui autorise une saisie *in extenso –,* et d'autre part puisque nous avons utilisé l'outil informatique pour construire des représentations à partir d'un bas niveau d'abstraction (les *qlist*). Cette pratique pourrait aussi s'avérer féconde pour l'analyse d'autres pièces d'informatique musicale.

[19] Pensons par exemple aux variables de substiution (*e.g. ston* qui affecte *tton* et *oton)* ou plus fortement aux connexions du Paf vers le Noïse qui se font à l'échelle des voies (*noise1, noise2 ...)*. Tout cela doit se retrouver dans les programmes

## 8. REMERCIEMENT

Je remercie Philippe Manoury pour les échanges que nous avons eus.

## 9. RÉFÉRENCES

[1] Bonardi, A. "Analyser et représenter la production du son dans les œuvres mixtes. Exemple du patch Max/MSP de *En Echo* de Philippe Manoury" Présentation orale dans le cadre du séminaire *Analyse par segmentation des musiques électroacoustiques,* Saint-Étienne, 2013.

[2] Bonardi, A. "Analyser l'orchestre électronique interactif dans les œuvres de Manoury", *Analyser la musique mixte*, Delatour, Sampzon, 2017. hal-01575296

[3] Clarke, M. "Jonathan Harvey's *Mortuos Plango, Vivo Voco", Analytical methods of electroacoustic music,* dir. Mary Symony. Routledge, London, 2006.

[4] Di Scipio, A. "An Analysis of Jean-Claude Risset's Contours", Journal of New Music Research, vol.29, n°1, p 1-21, 2000.

[5] Dufeu, F. "Comment développer des outils généraux pour l'étude des instruments de musique

numériques ? Un prototype en JavaScript pour l'investigation de programmes Max", Revue Francophone d'Informatique Musicale, n°3, 2013. revues.mshparisnord.org/rfim/index.php?id=255 (visité le 30 mars 2018)

[6] Goody, J. *La raison graphique. La domestication de la pensée sauvage*, Édition de Minuit, Paris, 1979.

[7] Larrieu, M. "Analyse des musiques d'informatique : vers une intégration de l'artefact. Proposition théorique et application sur *Jupiter* (1987) de Philippe Manoury". Thèse, Université Paris-Est Marne-la-Vallée, 2018. tel-01757277.

[8] Lemouton, S. "Vingt ans de pratique de la Réalisation en Informatique Musicale". Mémoire de Master 2, Université Paris-Est Marne-la-Vallée, 2012.

[9] Manoury, P., Battier, M., Bonardi, A., Lemouton, S. *Les musiques électroniques de Philippe Manoury*, Ircam, Paris, 2003.

[10] Manoury, P. "Considérations (toujours actuelles) sur l'état de la musique en temps réel", l'Étincelle, n°3, Ircam, Paris, 2007.

[11] May, A. "Philippe Manoury's Jupiter", *Analytical methods of electroacoustic music*, dir. Mary Symony. Routledge, London, 2006.

[12] Puckette, M. "New Public-Domain Realizations of Standard Pieces for Instruments and Live Electronics", International Computer Music Conference, Havana, Cuba, 2001.

[13] Puckette, M. *The theory and technique of electronic music*, World Scientific Publishing Co Inc, Singapore, 2006.

[14] Risset, J.-C. "Problèmes d'analyse : quelques clés pour mes premières pièces numériques, *Little Boy* et *Mutations", Analyse en musique électroacoustique, Acte de l'académie Internationale de Musique Électroacoustique de Bourges.* Mnémosyme, Bourges, 2001.

[15] Will, U. "La baguette magique de l'ethnomusicologue : repenser la notation et l'analyse de la musique", *Cahiers de musique traditionnelles,* n° 12, 1999. journals.openedition.org/ethnomusicologie/671 (visité le 30 mars 2018).

[16] Zattra, L. "Généticques de la computer music", *Genèses musicales*, dir. Nicolas Donin, A. Grésillon, J.-L. Lebrave. Presse universitaire de Paris-Sorbonne, Paris, 2015.

# INFÉRENCE DE SEGMENTATION STRUCTURELLE PAR COMPRESSION VIA DES RELATIONS MULTI-ÉCHELLES DANS LES SÉQUENCES D'ACCORDS

*Corentin Guichaoua*
Université de Rennes 1
IRISA – Équipe-projet PANAMA

*Frédéric Bimbot*
CNRS
IRISA – Équipe-projet PANAMA

**RÉSUMÉ**

Une des approches possibles pour l'analyse automatique de structure musicale est d'aborder la question comme un problème de compression : une représentation efficace de la structure musicale d'un morceau serait ainsi une représentation qui permet de décrire précisément ce morceau en peu de termes.

La méthode présentée dans cet article exploite les réseaux de relations à courte et moyenne échelle entre les événements musicaux afin d'obtenir des représentations compactes de segments de séquences musicales symboliques. Ces représentations compactes sont ensuite utilisées pour exprimer le morceau analysé comme une suite de segments de faible complexité, d'après le postulat qu'une segmentation découpant un morceau en un nombre limité d'éléments simples est susceptible de s'aligner avec la segmentation annotée manuellement.

Cette méthode est testée sur des séquences d'accords annotées manuellement, correspondant à la base de données RWC-Pop, et est comparée à une autre méthode à base de compression. Les segments obtenus correspondent avec une F-mesure de l'ordre de 70% avec les segmentations de référence.

## 1. INTRODUCTION

La structure musicale est un concept crucial de l'analyse de la musique, comme en témoigne son utilisation pour des applications variées, telles que la séparation chant-voix [1], la transcription de parole [2] ou la composition automatique [3]. En dépit de son importance, sa définition reste encore très ambiguë [4].

Devant les multiples définitions possible de la structure musicale et la nécessité de trouver un cadre permettant de les regrouper, cet article aborde la question sous l'angle de la théorie de l'information. Plus précisément, nous nous intéressons aux notions de complexité de Kolmogorov et d'entropie de Shannon pour rendre compte de la redondance présente dans les segments musicaux et les compresser. Selon cette optique, la structure musicale réside dans les régularités utilisables pour exécuter cette compression [5].

L'utilisation de données symboliques, comme une partition ou une séquence d'accords au lieu d'un enregistrement, en interposant une couche d'abstraction, permet de faire abstraction des régularités propres au signal audio pour se concentrer sur les régularités musicales. On se limitera ici à l'étude de séquences de triades (échantillonnées à 1 accord par temps), mais les principes décrits peuvent également être appliqués à d'autres types de descriptions symboliques.

Les séquences musicales présentent des régularités plus poussées que ce qui apparaît en examinant simplement leur nature séquentielle. En particulier, on peut observer des dépendances entre les éléments qui sont situés à des positions métriques homologues. Afin de modéliser ces dépendances multi-échelles, nous introduisons un modèle s'appuyant sur des graphes hypercubiques, proche de celui présenté dans [6], développé de façon concomitante dans l'équipe PANAMA, mais axé sur la modélisation de différents type d'organisations au lieu d'une description détaillée. Nous proposons une méthode exploitant ces dépendances afin d'obtenir des représentations compressées de segments musicaux, que nous utilisons ensuite pour trouver une séquence de segments peu complexes.

Dans la section 2, nous introduisons les concepts de *patrons tensoriels* et d'*historique tensoriel*, qui formalisent différents réseaux de relations et les dépendances qui en résultent entre les éléments d'un segment. Chaque patron différent correspond ainsi à une hypothèse sur l'organisation du segment qui, selon son adéquation avec le segment, permet ou non de l'exprimer de façon compacte.

La section 3 décrit comment les réseaux de relations définis par les patrons tensoriels peuvent être utilisés pour tenter d'anticiper successivement chacun des éléments de la séquence.

En section 4, nous appliquons cette méthode de compression des segments pour la segmentation structurelle, en recherchant les séquences de segments résultant en une complexité minimale.

Nous étudions en section 5 l'application de cette méthode de segmentation sur des annotations de séquences d'accords de musique pop, avant de conclure en section 6.

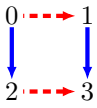
## 2. PATRONS TENSORIELS

Avant d'utiliser les relations multi-échelle entre les différents éléments d'une séquence musicale, il convient de

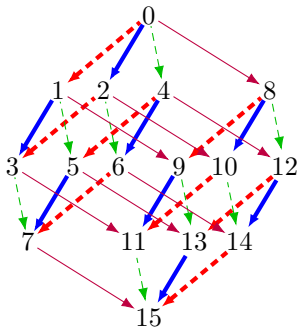
Corentin Guichaoua, Frédéric Bimbot, *Inférence de segmentation structurelle par compression via des relations multi-échelles dans les séquences d'accords*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 71–79, 2018. 

définir l'architecture de ces relations. Pour cela, nous introduisons les *patrons tensoriels*, qui décrivent l'organisation des éléments à plusieurs échelles simultanément. Plutôt qu'un modèle universel de l'organisation d'un segment musical, il s'agit d'un catalogue (non-exhaustif) d'organisations simples, parmi lesquelles celle permettant au mieux de décrire la séquence sera sélectionnée (cf. sections suivantes).

L'organisation non-linéaire la plus simple est une organisation de 4 éléments en carré, où chaque élément dépend de ses voisins apparaissant antérieurement dans la séquence, comme illustré en figure 1a. Ces relations sont considérées comme *homologues* lorsque les arêtes associées sont parallèles. Cette organisation peut être généralisée à des séquences de $2^d$ éléments, organisés selon un hypercube de dimension $d$ (par exemple en dimension 4 sur la figure 1b).




(a) Patron carré (2-cubique)



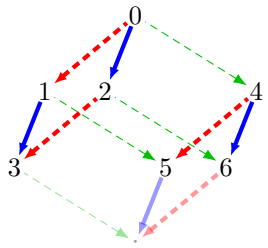

(b) Patron tesseractique (4-cubique)

**Figure 1**: Exemples de patrons hypercubiques. Les styles de traits indiquent les classes de relations homologues.

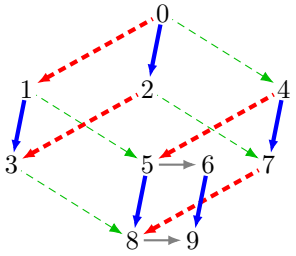
Cette extension, à elle seule, reste toutefois très insuffisante pour modéliser la variété des organisations rencontrées en pratique, même dans des genres considérés comme pauvres à cet égard comme la pop. En effet, on observe souvent des sections musicales dont la longueur en temps musicaux n'est pas une puissance de 2. Afin de pouvoir gérer une plus grande variété de séquences, nous étendons l'inventaire des patrons hypercubiques à l'aide de deux opérations de modification permettant d'augmenter ou de diminuer le nombre d'éléments admis par le patron. La première opération, dite d'insertion, agit sur une hyper-face [1] de l'hypercube définissant le patron, en la dupliquant et en connectant ses éléments avec les éléments correspondants de la nouvelle copie. La seconde opération, dite d'omission, agit également sur une hyper-face, la retirant du patron. Un exemple de chaque cas est présenté en figure 2.

Pour éviter une multiplication excessive du nombre de patrons, on se limite à la manipulation d'hyper-faces dont un des sommets est le dernier élément de l'hypercube, et à une seule insertion et/ou une omission. Cette restriction permet d'identifier les modifications de façon unique par

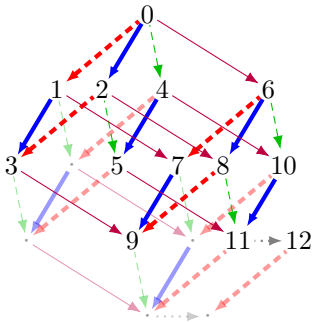
1 . Une hyper-face est un hypercube inclus dans un hypercube de dimension supérieur




(a) Patron avec une 0-omission, admettant une séquence de 7 éléments




(b) Patron avec une 1-insertion, admettant une séquence de 10 éléments



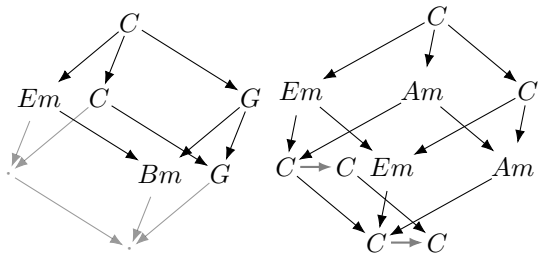

(c) Patron avec une 1-insertion et une 2-omission, admettant une séquence de 13 éléments

**Figure 2**: Exemples de patrons avec omission, avec insertion et avec une combinaison des deux.

la position du sommet opposé de l'hyper-face. Ainsi, en accord avec le principe d'utiliser des organisations simples, 3 nombres (au plus) suffisent à identifier les patrons : la dimension de l'hypercube, et les positions des modifications éventuelles.

Ces extensions supplémentaires, bien qu'elles ne permettent toujours pas de modéliser toutes les longueurs de séquences, offre un éventail bien plus diversifié d'organisations, qui suffisent à couvrir la majorité de cas rencontrés. La figure 3 montre des exemples de séquences avec des signatures impaires modélisées à l'aide de patrons déformés.

À partir de ces réseaux, on définit pour chaque élément son *historique tensoriel*, le sous-ensemble des relations et des éléments dont dépend cet élément, directement ou transitivement. Deux exemples d'historiques tensoriels sont présentés en figure 4. Ces historiques tensoriels sont pour chaque élément l'information qui est utilisée afin d'établir une anticipation, comme décrit dans la section suivante.




(a) Séquence de structure ternaire modélisée à l'aide d'une omission

(b) Séquence de structure pentadique modélisée à l'aide d'une insertion

**Figure 3**: Exemples de structures non carrées

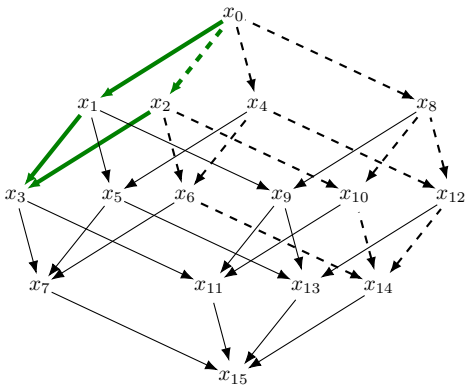



**Figure 4**: En vert épais l'historique tensoriel de $x_3$, en pointillé l'historique de $x_{14}$. Les éléments d'origine des arcs font partie des historiques tensoriels

## 3. ANTICIPATION SELON UN PATRON TENSORIEL

Une façon de mesurer l'adéquation d'un patron tensoriel à un segment est d'observer dans quelle mesure l'utilisation de ce patron permet d'anticiper chacun des éléments de la séquence correspondante en fonction des éléments passés. D'après les principes énoncés en introduction, un patron approprié permettra ainsi d'anticiper plus d'éléments qu'un patron inadapté. Cette mesure présente l'avantage de pouvoir être effectuée dans un cadre non-supervisé, sans entrée autre que la séquence à modéliser.

Ce procédé peut être exprimé selon un point de vue compressif : pour chaque élément de la séquence, 3 cas peuvent se présenter, qui déterminent la quantité d'information requise pour encoder l'élément :

1. L'élément observé correspond à l'élément anticipé : aucune information supplémentaire n'est nécessaire pour décrire cet élément.
2. L'élément observé ne correspond pas à l'élément anticipé : il est nécessaire de spécifier l'élément observé ou en quoi il diffère de l'élément anticipé (du même ordre dans le cas d'éléments simples).
3. Un élément anticipé n'a pas pu être produit (par exemple le premier élément de la séquence pour lequel aucune information n'est disponible) : il est nécessaire de spécifier l'élément observé.

En effectuant la simplification que le bit utilisé pour distinguer le premier du deuxième cas est négligeable devant la spécification d'un élément, la longueur de la description de la séquence selon le patron tensoriel est proportionnelle au nombre d'éléments n'ayant pas été correctement anticipés.

Avant de décrire le processus par lequel le patron tensoriel est utilisé pour produire des anticipations d'éléments, nous décrivons le modèle que nous utilisons pour la représentation des relations entre les accords [2].

### 3.1. Relations entre les éléments

Le modèle de représentation des accords et de leurs relations que nous avons choisi d'utiliser consiste en deux simplifications principales.

La première simplification est une restriction du vocabulaire des accords aux 24 triades majeures et mineures [3]. Cette simplification est assez courante dans les approches symboliques et ne sera pas détaillée.

La deuxième simplification porte sur les relations entre les accords deux à deux : ces relations sont modélisées par les rotations envoyant un accord sur un autre sur le cercle des tierces présenté en figure 5. En plus de ses propriétés musicales, par son équivalence avec $\mathbb{Z}/24\mathbb{Z}$, cet espace de représentation des relations a les propriétés clés suivantes (que nous dirons quasi-affines [4]) :

— la composition est définie et respecte la relation de Chasles pour toutes les paires de relations ;
— la composition est commutative et associative ;
— chaque accord a une unique image par une transformation donnée.

### 3.2. Cas du carré

Le cas le plus simple est celui de l'organisation en carré, qui correspond au modèle Système & Contraste introduit dans [7], illustré avec des formes en figure 6. L'arrangement des 4 éléments $x_0$, $x_1$, $x_2$ et $x_3$ de la séquence correspond à l'hypothèse d'une relation d'analogie entre ces éléments, où $x_3$ est à $x_2$ ce que $x_1$ est à $x_0$, soit, en utilisant une notation vectorielle $\overrightarrow{x_0x_1} = \overrightarrow{x_2x_3}$.

En pratique, cette analogie n'est pas toujours vérifiée, on note alors $\overrightarrow{\gamma_3}$ la différence entre l'élément observé $x_3$ et l'élément *anticipé* $\widehat{x_3}$ qui aurait vérifié l'analogie, que l'on nomme *contraste*.

Cette formulation permet de mettre en avant le rôle joué par chacun des éléments dans l'organisation de la sé-

2 . À l'exception de la section expérimentale, cette description du modèle des accords est la seule partie de cet article qui est spécifique aux accords. À condition de respecter les propriétés clés qui y sont énoncées, tout modèle de représentation des propriétés de la séquence est compatible avec les principes développés.

3 . Cet espace et les relations peuvent être étendus avec un non-accord pour les temps où la notion d'accord ne s'applique pas. Cette solution est un artifice qui permet de traiter par extension les passages où il n'existe pas d'harmonie identifiable.

4 . Ces propriétés s'apparentent à celles d'un espace affine, à ceci près que la multiplication par un scalaire n'est pas toujours inversible.

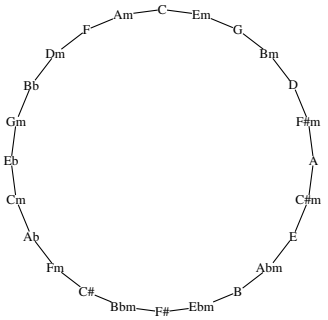



**Figure 5**: Le cercle des tierces utilisé pour représenter l'espace des accords. Chaque accord partage deux notes avec ses voisins et l'on passe d'un accord à son voisin horaire en décalant d'une tierce (majeure si l'accord de départ est majeur, mineure sinon) vers les aigüs.

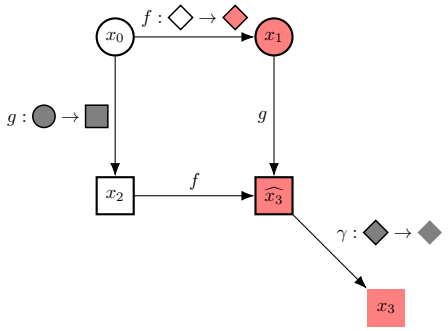



**Figure 6**: Visualisation des relations entre les éléments du modèle Système & Contraste sur un système d'éléments géométriques. Les relations portent sur le remplissage, la forme et le contour des éléments. L'élément $\widehat{x_3}$ est un élément virtuel qui ne fait pas partie de la séquence observée.

quence (analysée à l'aide du patron carré), et sa contribution à la quantité d'information nécessaire à l'encodage de la séquence. Le premier élément ($x_0$), appelé *primer*, permet d'ancrer la séquence et sert de point de référence. Il est présenté sans information préalable et doit toujours être encodé. Le deuxième et le troisième élément ($x_1$ et $x_2$), appelés sous-primers, établissent les transformations (ou relations) qui définissent le reste de la séquence. Un cas simple se dégage pour chacun de ces éléments, celui où l'élément est identique au primer, pour lequel une quantité d'information faible est requise. Enfin, le dernier élément $x_3$, en correspondant ou non avec l'élément anticipé (on dit qu'il est *concordant* ou *discordant* respectivement), détermine si la séquence est *contrastive* ou non. Dans le cas où il est concordant, très peu d'information est requise pour l'encoder.

On peut noter que dans le cas d'objets présentant plusieurs propriétés comme dans l'exemple des formes, l'encodage des transformations peut être plus économique en information, même si aucune des transformations n'est triviale. Ce n'est toutefois pas le cas avec notre représentation simplifiée des accords, ce qui limite l'intérêt du cas carré pris en isolation.

### 3.3. Cas régulier en dimension 3

Une première observation dans le cas des patrons hypercubiques de dimension supérieure est que tous les éléments à l'exception du dernier peuvent être ramenés à des cas de dimension inférieure : par exemple, dans un patron cubique (3-cubique), l'élément $x_6$ est le dernier élément du carré formé par les éléments $x_0$, $x_2$, $x_4$ et $x_6$. Ainsi, en procédant en dimension croissante, seule la détermination de l'élément anticipé pour la dernière position doit être adaptée.

Considérons d'abord le cas du cube : comme dans le cas carré, on peut considérer l'analogie définie par $\overrightarrow{x_0x_2} = \overrightarrow{x_1x_3} = \overrightarrow{x_4x_6} = \overrightarrow{x_5x_7}$, illustrée en figure 7. Cependant, contrairement au cas carré, plus d'une de ces relations a été observée dans l'historique tensoriel de $x_7$, et une contradiction peut être apparue si $x_3$ ou $x_6$ est discordant. S'ils sont concordants, on peut définir un élément anticipé pour $x_7$ d'après le parallélogramme formé par les éléments $x_0$, $x_2$, $x_5$ et $x_7$, de la même façon que pour une organisation carrée.

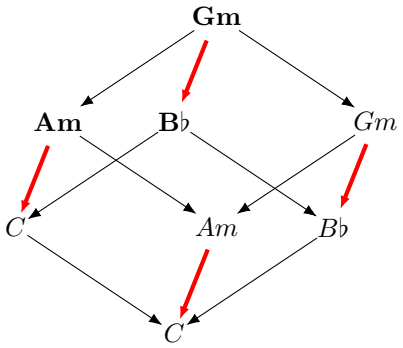



**Figure 7**: Mise en évidence d'une analogie sur un patron cubique, à partir de la séquence d'accords du pont (1'38 à 1'52) du morceau n°10 de RWC (*Getting Over*). La séquence est ici prise à 1 accord par mesure et peut être entièrement déduite des 3 seuls accords en gras, qui en constituent ainsi une représentation compressée.

Par symétrie, on peut également considérer les analogies utilisant les relations $\overrightarrow{x_0x_1}$ ou $\overrightarrow{x_0x_4}$. Ces analogies supplémentaires sont contredites respectivement si $x_3$ ou $x_5$ sont discordants et si $x_5$ ou $x_6$, et ne sont donc pas équivalentes, contrairement au cas carré où les deux directions étaient interchangeables.

Un autre type d'analogie que l'on peut effectuer est en considérant les faces parallèles formées d'une part par $x_0$, $x_1$, $x_2$ et $x_3$, et d'autre part par $x_4$, $x_5$, $x_6$ et $x_7$. Cette analogie est toutefois équivalente à l'analogie selon la relation $\overrightarrow{x_0x_4}$. Ici encore, on retrouve par symétrie deux autres analogies entre faces, équivalentes aux analogies selon les arêtes.

On peut remarquer que si plus d'un élément parmi $x_3$, $x_5$ et $x_6$ est contrastif, aucune des analogies n'est vérifiée sur les éléments observés : dans ce cas, on ne peut pas construire d'élément anticipé de cette façon, et il est nécessaire d'encoder $x_7$ non trivialement. Une conséquence

de la nature quasi-affine de l'espace des relations est que les analogies non contredites aboutissent au même élément anticipé. On a alors un unique point de référence auquel comparer l'élément observé $x_7$ pour l'encoder trivialement s'il est concordant.

### 3.4. Cas en dimension supérieure

En dimension 4 et au delà, le même processus peut être appliqué en considérant les analogies définies par les hypercubes contenant le primer. Comme dans le cas du cube, ces constructions sont complémentaires 2 à 2 : pour la définition de l'élément projeté pour $x_i$, l'analogie définie par l'hypercube entre $x_0$ et $x_h$ est équivalente à l'analogie définie par l'hypercube orthogonal entre $x_0$ et $x_{i-h}$. Ces deux analogies, si elles ne comportent pas de contradiction, peuvent être réduites au parallélogramme formé par $x_0$, $x_h$, $x_{i-h}$ et $x_i$. Un exemple de paire d'hypercubes complémentaires et du parallélogramme qui leur est associé est présenté en figure 8.

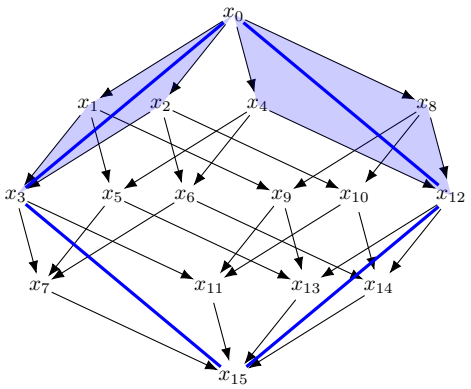



**Figure 8**: Paire de carrés complémentaires et l'organisation en parallélogramme correspondante

L'inventaire des conditions pour lesquelles l'analogie est contredite par les éléments observés peut sembler laborieux de prime abord, mais s'exprime relativement aisément en fonction de ces hypercubes complémentaires : l'analogie pour $x_i$ basée sur un hypercube entre $x_0$ et $x_h$ est contredite s'il existe au moins un élément discordant dans l'hypercube entre $x_0$ et $x_i$ qui n'est ni dans l'hypercube entre $x_0$ et $x_h$, ni dans son complémentaire (entre $x_0$ et $x_{i-h}$).

Là encore, les propriétés quasi-affines de l'espace de relations assurent que les analogies non-contredites aboutissent au même élément anticipé.

### 3.5. Influence des déformations

Jusqu'ici, nous avons décrit comment sont construit les éléments anticipés dans le cas des patrons basés sur des hypercubes non déformés. Afin d'utiliser tout l'éventail des patrons définis en section 2, nous étendons ces constructions aux patrons avec omission et/ou insertion.

L'extension aux patrons avec omission est presque triviale : l'omission ne fait que retirer des éléments à anticiper, et grâce à la contrainte sur les positions des hyperfaces supprimées, les éléments supprimés n'apparaissent que dans les historiques tensoriels d'autres éléments supprimés. La re-numérotation des éléments rompt toutefois la compatibilité des formules s'appuyant sur les index, mais cette difficulté est aisément contournée en appliquant ces calculs sur les index non modifiés.

L'extension aux patrons avec insertion présente également cette difficulté mineure, à laquelle s'ajoute le besoin de définition d'un élément projeté pour les éléments insérés. Une solution consiste à restreindre l'historique tensoriel des éléments insérés à l'hypercube formé par l'hyperface dupliquée et sa copie : ce cas se ramène alors localement à un cas hypercubique régulier.

## 4. APPLICATION À LA SEGMENTATION STRUCTURELLE

Dans la section précédente, nous avons décrit un processus permettant d'évaluer la complexité d'une séquence donnée selon un patron tensoriel donné. Cette complexité n'est pas nécessairement représentative de la complexité de la séquence, toutefois en répétant cette opération pour tous les patrons tensoriels compatibles et en sélectionnant celui permettant la plus forte compression, une approximation plus juste peut être obtenue, à condition que l'éventail des patrons soit suffisamment riche. Le nombre de patrons compatibles pour une longueur donnée ne dépassant pas l'ordre de quelques centaines (dans le pire cas) pour des longueurs correspondant à l'échelle des segments musicaux, une telle exploration exhaustive reste tractable. Afin de prendre en compte la régularité plus grande des patrons hypercubiques, une pénalité, réglée expérimentalement, est appliquée pour chaque type d'altération appliquée au patron.

Afin de passer de cette mesure de complexité d'un segment individuel à l'inférence d'une segmentation d'un morceau entier, nous calculons, pour chaque paire de position de départ et de fin formant un segment ne dépassant pas une taille plafond de 64 éléments [5] la complexité de ce segment. Il en résulte un graphe dirigé représentant l'ensemble des segmentations possibles du morceau, chaque arc étant portant la complexité du segment correspondant. Ce graphe peut être parcouru avec un algorithme de recherche du plus court chemin afin d'obtenir une segmentation minimisant la somme des complexités de chaque segment.

## 5. EXPÉRIENCES

Nous évaluons cette méthode sur les chansons de musique pop de la base de données RWC-Pop, constituées de 100 chansons composées et interprétées dans le style des musiques du hit-parade aux États-Unis dans les années 80

5 . Ce seuil est fixé a priori au double de l'échelle ciblée pour la segmentation, 32 éléments dans notre cas. Le choix de cette échelle cible implique que les patrons utilisés sont généralement de dimension 5, avec occasionnellement des patrons de dimension 3,4 ou 6.

et au Japon dans les années 90 [8]. Comme point de référence, nous utilisons une autre méthode basée sur le principe de compression, mais utilisant des algorithmes génériques de factorisation de répétitions, détaillée dans [9, Ch3-4].

### 5.1. Séquences d'accords

À notre connaissance, sur RWC-Pop, aucune base d'annotation n'existe associant à chaque temps un accord. Afin de produire de telles annotations, nous avons recoupé automatiquement les annotations d'accords par plage temporelle de [10] avec les annotations de temps de [11]. Cette base d'annotations, que nous dénotons AUTO, présente toutefois de nombreuses erreurs liées à la quantification des temps (en particulier pour les changements d'accords à contretemps), et nous avons effectué une première passe manuelle de correction des données, produisant ainsi une nouvelle base dénotée CORRECTED. Une seconde passe, focalisée sur la cohérence des séquences ambigües dans les segments répétés, a permis d'obtenir une nouvelle révision dénotée MANUAL. La figure 9 présente un extrait d'une séquence, un exemple détaillé est étudié en fin d'article et le reste des données est disponible en ligne [6] .

```
[...] Gm Gm Gm Gm Am Am Am Am Bb Bb
Bb Bb C C C C Gm Gm Gm Gm Am Am Am Am
Bb Bb Bb Bb C C C C F F F F C C C C Bb
Bb Bb Bb C C C C F F F F C C C C Bb Bb
Bb Bb C C C C F F F F F F F F F F F F
F F F F Dm Dm Dm Dm Dm Dm Dm Dm Bb Bb
Bb Bb Bb Bb Bb Bb Dm Dm Dm Dm Dm Dm Dm
Dm Bb Bb Bb Bb Bb Bb Bb Bb C C C C C C
C C Bb Bb Bb Bb Bb Bb Bb Bb C C C C C
C C C Bb Bb Bb Bb C C C C Gm Gm Gm Gm
Am Am Am Am Bb Bb Bb Bb C C C C Gm Gm
Gm Gm Am Am Am Am Bb Bb Bb Bb C C C C
F F F F C C C C Bb Bb Bb Bb C C C C F
F F F C C C C Bb Bb Bb Bb C C C C F F
F F F F F F F F F F F F F F F F F F
F F F F F F F N N N N
```

**Figure 9**: Extrait (de 1'38 à la fin) de la séquence d'accords du morceau n°10 de RWC-Pop. On peut noter que l'échantillonnage au temps près est ici redondant (un accord par mesure aurait suffi), mais le suréchantillonnage ne change pas le résultat de l'algorithme.

### 5.2. Métriques

Afin d'obtenir des métriques faisant écho dans un cadre discret aux mesures utilisées pour la segmentation à partir de l'audio dans la campagne d'évaluation MIREX, nous avons choisi de mesurer la performance des algorithmes d'après la F-mesure, calculée d'une part exactement et d'autre part avec une tolérance de $\pm 3$ temps (ce qui correspond approximativement aux intervalles de tolérance de 0.5 s et 3 s). Les segmentations de références sont issues de [12], adaptées manuellement à des valeurs de temps discrètes.

### 5.3. Sensibilité aux paramètres

En section 4 nous avons décrit une pénalisation des patrons modifiés pour refléter la complexité accrue des modèles utilisés pour expliquer la séquence. Concrètement, la complexité tensorielle d'un segment $X$ est donnée par la formule

$$\mathcal{C}(X) = \min_P \mathcal{C}(X|P) + \mathcal{C}(P) \tag{1}$$

où $\mathcal{C}(X|P)$ est le nombre d'éléments discordants de $X$ d'après le patron $P$ et

$$\mathcal{C}(P) = \begin{cases} 0 & \text{si } P \text{ est hypercubique} \\ p_+ & \text{si } P \text{ est avec insertions} \\ p_- & \text{si } P \text{ est avec omission} \\ p_+ + p_- & \text{si } P \text{ est mixte} \end{cases} \tag{2}$$

donne la complexité du patron, où $p_+$ et $p_-$ sont des paramètres dont nous cherchons à évaluer l'influence.

La table 1 expose les résultats obtenus pour différentes combinaisons de ces valeurs. On constate que l'influence de ces paramètres est modérée, à l'exception des cas où les pénalités sont nulles, et que les variations sont relativement lisses. Des expériences préliminaires de sous-échantillonnage confirment que les valeurs optimales sont peu fluctuantes.

**Table 1**: F-mesure moyenne (sans tolérance, en pourcentage) selon les valeurs des pénalités appliquées aux déformations du patron hypercubique.

| $p_+$ \ $p_-$ | 0 | 0.01 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| 0 | 43.7 | 48.6 | 47.9 | 48.4 | 48.0 | 47.6 |
| 0.01 | 48.4 | 49.8 | 52.7 | 54.1 | 53.1 | 52.1 |
| 1 | 47.8 | 52.3 | 55.5 | 55.6 | 55.6 | 55.2 |
| 2 | 50.2 | 54.1 | 56.4 | 59.5 | 58.7 | 56.9 |
| 3 | 50.2 | 54.6 | 56.7 | 59.3 | **59.6** | 58.6 |
| 4 | 49.7 | 54.6 | 56.9 | 58.4 | 57.5 | 57.9 |

### 5.4. Régularisation

Un examen manuel des segmentations révèle que les segments obtenus sont souvent pertinents, mais correspondent à une échelle de segmentation inférieure à celle considérée lors des annotations (de l'ordre de 32 temps par segment). Afin de calibrer l'algorithme pour cibler cette échelle (connue a priori), nous introduisons un terme de régularisation privilégiant les segments d'une longueur proche. Ce terme est de la forme suivante :

$$S_r(X) = \begin{cases} S(X) + r_+ \left(|X| - l_r\right) & \text{si } |X| \geq l_r \\ S(X) + r_- \left(l_r - |X|\right) & \text{si } |X| < l_r \end{cases} \tag{3}$$

6. https://gforge.inria.fr/projects/rwcpop-chordseq/

où $|X|$ est la taille de la séquence, $l_r$ est la taille cible des segments et $r_+$ et $r_-$ sont les coefficients de régularisation.

**Table 2**: F-mesure (sans tolérance, en pourcentage) selon les paramètres de régularisation. On utilise $p_+ = p_- = 2.25$, la paire de valeurs offrant les meilleurs résultats sans régularisation.

| $r_-$ \ $r_+$ | 0 | 0.0001 | 0.025 | 0.05 | 0.075 | 0.1 | 0.125 | 0.15 |
|---|---|---|---|---|---|---|---|---|
| 0 | 61.1 | 61.4 | 61.4 | 60.8 | 60.9 | 60.5 | 60.2 | 60.4 |
| 0.0001 | 62.8 | 62.8 | 62.5 | 61.9 | 61.7 | 61.2 | 61.3 | 61.1 |
| 0.025 | 62.2 | 62.2 | 62.4 | 62.4 | 62.6 | 62.7 | 61.7 | 61.7 |
| 0.05 | 65.5 | 65.5 | 65.3 | 65.5 | 66.2 | 66.1 | 65.9 | 64.8 |
| 0.075 | 66.2 | 66.2 | 66.7 | 67.0 | 66.9 | 66.7 | 66.6 | 66.3 |
| 0.1 | 67.8 | 67.9 | 67.0 | 66.9 | 66.7 | 66.7 | 66.8 | 66.5 |
| 0.125 | 67.8 | 67.9 | 67.1 | 66.5 | 66.4 | 66.1 | 66.4 | 66.3 |
| 0.15 | **68.0** | 67.8 | 67.5 | 67.5 | 66.5 | 66.1 | 65.9 | 66.0 |
| 0.175 | 67.4 | 67.4 | 67.6 | 67.2 | 66.8 | 66.5 | 65.9 | 65.7 |
| 0.2 | 65.9 | 65.9 | 66.1 | 66.5 | 66.6 | 65.8 | 65.3 | 65.0 |

La table 2 montre l'évolution de la F-mesure selon le réglage de ces paramètres. L'optimum est atteint pour une valeur de $r_+$ nulle, ce qui est cohérent avec le but de réduire la sur-segmentation. À l'exception d'un effet de seuil marqué entre les valeurs 0.025 et 0.05 pour $r_-$, l'influence de ces paramètres est encore modérée et lisse. À nouveau, des expériences préliminaires de sous-échantillonnage confirment les optimums obtenus.

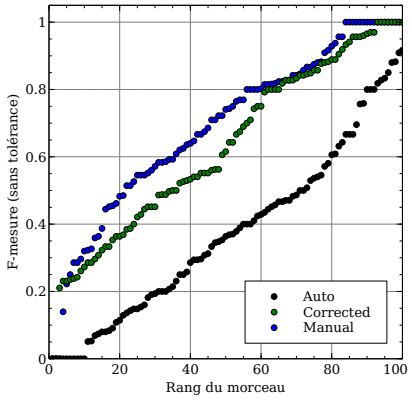
### 5.5. Choix de la base et variabilité

Les expériences ci-dessus étaient rapportées pour la F-mesure exacte, sur la base MANUAL, pour laquelle les données d'entrée sont de bonne qualité. Les bases AUTO et CORRECTED permettent de tester le comportement de l'algorithme lorsque les données sont à différents niveaux de dégradation (sans aller toutefois jusqu'à des annotations complètement automatiques). La figure 10 montre une vue transversale des performances par morceau de l'algorithme, selon la base et la version de la F-mesure utilisée.

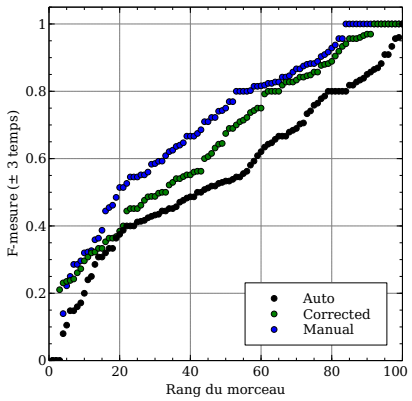
On observe que les différences entre la base MANUAL et la base CORRECTED sont assez modérées, mais qu'en revanche la base AUTO est nettement moins bien segmentée, en particulier dans le cas de la F-mesure exacte. Une autre propriété remarquable est que l'ajout d'une fenêtre de tolérance a une influence très faible sur les 2 meilleures bases, ce qui contraste avec ce que l'on peut généralement observer dans les campagnes MIREX sur la segmentation audio. Enfin, cette figure met en évidence la diversité de la qualité de la segmentation selon les morceaux : si presque la moitié de la base est correctement segmentée à 80% ou plus, 20% des morceaux ont moins de la moitié des frontières de segment correctement inférées.

### 5.6. Comparaison avec une autre approche

La table 3 rapporte les F-mesures obtenues pour les méthodes présentées dans cet article, comparées à une méthode de référence opérant sur le même type de données et utilisant des méthodes de compression de texte pour l'inférence de la structure [9, ch. 3-4]. Les méthodes à encodage tensoriel obtiennent des scores de segmentation nettement plus élevés.




(a) F-mesure sans tolérance




(b) F-mesure avec tolérance

**Figure 10**: F-mesure sur chacun des 100 morceaux de RWC-Pop selon l'annotation utilisée pour l'encodage tensoriel par analogie avec régularisation. Pour plus de lisibilité, les morceaux ont été ré-ordonnés par F-mesure croissante selon chaque annotation

### 5.7. Détail d'une segmentation

Le morceau numéro 24 de la base MANUAL est un morceau pour lequel la F-mesure entre la segmentation inférée est légèrement au dessus de la moyenne (F=0.71). La figure 11 montre la segmentation de référence et la segmentation obtenue par l'algorithme d'inférence.

On peut voir que les segmentations diffèrent à certains endroits :

— La frontière entre le deuxième segment et le troisième segment (0'19 à 0'41 dans l'audio) est placée 16 temps (un demi-segment) plus tard dans la segmentation inférée. En effet, sur la séquence d'accords, la première moitié du 3e segment forme un système non contrastif avec le deuxième segment (il s'agit d'une répétition à l'identique). L'indice principal qui permet de distinguer une frontière à cet endroit est le retour du chant dans l'instrumentation, mais cet indice n'est pas présent dans la séquence d'accords. On retrouve également la même

**8.25 (40)** Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C
Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C Fm Fm Fm
Fm Bb Bb N N
**4 (16)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm
**5 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm F# F# F# F# Ab Ab Ab Ab
**9.25 (40)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb
Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm F# F# F# F# Ab Ab Ab
Ab Eb Eb Eb Eb Eb Eb Eb
**9 (32)** Ab Ab Ab Ab F# F# F# F# Eb Eb Eb Eb Eb Eb Eb Eb
Ab Ab Ab Ab F# F# F# F# Ab Ab Ab Ab N N N
**8.25 (40)** Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C
Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C Fm Fm Fm
Fm Bb Bb N N
**4 (16)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm
**9.25 (40)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb
Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm F# F# F# F# Ab Ab Ab
Ab Eb Eb Eb Eb Eb Eb Eb
**9 (32)** Ab Ab Ab Ab F# F# F# F# Eb Eb Eb Eb Eb Eb Eb Eb
Ab Ab Ab Ab F# F# F# F# Ab Ab Ab Ab Ab N N N
**8.25 (40)** Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C
Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C Fm Fm Fm
Fm Bb Bb N N
**2 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm
**9 (32)** Ab Ab Ab Ab F# F# F# F# Eb Eb Eb Eb Eb Eb Eb Eb
Ab Ab Ab Ab F# F# F# F# Ab Ab Ab Ab Ab N N N
**12 (56)** Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C Ab
Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C Fm Fm Fm Fm
Bb Bb Bb Bb Gm Gm Gm Gm C C C C Fm Fm Fm Fm Bb Bb N N
**2 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm
**4 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb N N N N

(a) Segmentation de référence. Score : 103.25

**8.25 (40)** Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C
Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C Fm Fm Fm
Fm Bb Bb N N
**2 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm
**7 (16)** Eb Eb Eb Eb Eb Eb Ebm Ebm F# F# F# F# Ab Ab Ab
Ab
**5 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm F# F# F# F# Ab Ab Ab Ab
**9.25 (40)** Eb Eb Eb Eb Eb Eb Eb Eb Ab Ab Ab Ab F# F# F#
F# Eb Eb Eb Eb Eb Eb Eb Eb Ab Ab Ab Ab F# F# F# F# Ab
Ab Ab Ab Ab N N N
**8.25 (40)** Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C
Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C Fm Fm Fm
Fm Bb Bb N N
**2 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm
**7 (16)** Eb Eb Eb Eb Eb Eb Ebm Ebm F# F# F# F# Ab Ab Ab
Ab
**9.25 (40)** Eb Eb Eb Eb Eb Eb Eb Eb Ab Ab Ab Ab F# F# F#
F# Eb Eb Eb Eb Eb Eb Eb Eb Ab Ab Ab Ab F# F# F# F# Ab
Ab Ab Ab Ab N N N
**8.25 (40)** Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C
Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C Fm Fm Fm
Fm Bb Bb N N
**2 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm
**9 (32)** Ab Ab Ab Ab F# F# F# F# Eb Eb Eb Eb Eb Eb Eb Eb
Ab Ab Ab Ab F# F# F# F# Ab Ab Ab Ab Ab N N N
**4 (32)** Ab Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C Ab
Ab Ab Ab Bb Bb Bb Bb Gm Gm Gm Gm C C C C
**8 (24)** Fm Fm Fm Fm Bb Bb Bb Bb Gm Gm Gm Gm C C C C Fm
Fm Fm Fm Bb Bb N N
**2 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm
**4 (32)** Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb Eb Eb Ebm
Ebm Eb Eb Eb Eb Eb Eb Ebm Ebm Eb Eb Eb Eb N N N N

(b) Segmentation inférée. Score: 95.25

**Figure 11**: Segmentations du morceau n°24 de RWC-Pop. Chaque segment est précédé par son score de complexité et sa taille. $(r_+, r_-, p_+, p_-) = (0, 0.125, 2.25, 3)$.

**Table 3**: F-mesure moyenne (en pourcentage, sans tolérance) pour les méthodes à encodage tensoriel et la méthode de référence

| Encodage \ Base | AUTO | CORR | MANUAL |
|---|---|---|---|
| GDU [9] | 30 | 44 | 52 |
| Tensoriel | 33 | 56 | 61 |
| Tensoriel régularisé | 37 | 63 | 69 |

configuration entre le 6e et le 7e segment.

— La frontière entre le 4e et le 5e segment est placée 8 temps plus tôt dans la segmentation inférée. Ces 8 temps en $E\flat$ (0'56 à 1'00) sont répétés dans le segment suivant et permettent de constituer un système relativement régulier avec le segment suivant. Dans la version audio du morceau, la dynamique et la présence d'un fill de batterie permettent d'écarter cette hypothèse, mais ces indices ne sont encore une fois pas présents dans la séquence d'accords.

— Le troisième segment en partant de la fin (3'01 à 3'28) a été divisé en deux sous-segments. On peut remarquer d'une part que le segment de la référence obtient le même score que les deux segments réunis, et d'autre part que cette sous-division du segment est également plausible d'après l'audio.

## 6. DISCUSSION ET CONCLUSION

Dans cet article, nous avons présenté une méthode de segmentation structurelle de séquences symboliques basée sur la compression individuelle de chaque segment, selon un modèle exploitant les relations multi-échelle entre leurs éléments. Nous avons expérimenté cette méthode sur des séquences d'accords de musique pop, comparant favorablement les résultats à ceux obtenus à l'aide d'une autre méthode.

Ces résultats, de l'ordre de 70% de F-mesure au temps près, sont proches des taux de concordance qui peuvent être observés entre annotateurs humains lors de campagnes d'annotation participative [13], ce qui montre un fort potentiel des méthodes à compression tensorielle, et plus généralement du paradigme d'inférence par compression.

Pour réaliser ce potentiel, plusieurs axes prometteurs consistent à réduire l'information défaussée aux diverses étapes de la modélisation :

— Les régularités à long terme, complètement ignorées par la présente méthode, sont fréquemment utilisés par des algorithmes d'inférence de structure (musicale ou autre) ; la complémentarité de ces approches suggère que des avancées sont possibles en les hybridant.

— Les modèles de relation entre accords utilisés sont une vue simpliste de l'harmonie, qui gagnerait à être raffinée en relâchant une partie des hypothèses émises en section 3. En particulier, le cas où la relation entre deux accords peut être interprétée de plusieurs façons (transposition chromatique ou dia-

tonique par exemple) semble être une piste ardue mais fructueuse.

— La réduction d'un morceau à une séquence d'accords est une simplification extrêmement brutale. Avec les modèles de relation appropriés, la méthode peut être appliquée à des mélodies, des rythmes, des activités d'instruments, etc. Individuellement, chacune de ces dimensions musicales est une simplification aussi brutale de la musique, mais leur utilisation jointe serait certainement bénéfique.

On peut enfin noter que l'information contenue dans la représentation utilisée par cette méthode dépasse la simple segmentation et inclut à la fois des estimations de la structure interne des segments et desquels de leurs éléments apportent des informations nouvelles.

## 7. REFERENCES


[1] Antoine Liutkus, Zafar Rafii, Roland Badeau, Bryan Pardo, and Gaël Richard. Adaptive Filtering for Music/Voice Separation Exploiting the Repeating Musical Structure. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 53–56. IEEE, 2012.

[2] Matt McVicar and Daniel P.W. Ellis. Leveraging Repetition for Improved Automatic Lyric Transcription in Popular Music. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3117–3121, 2014.

[3] François Pachet, Alexandre Papadopoulos, and Pierre Roy. Sampling Variations of Sequences for Structured Music Generation. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, China, 2017.

[4] Jordan B.L. Smith. *Explaining Listener Differences in the Perception of Musical Structure*. PhD thesis, Queen Mary, University of London, UK, 2014.

[5] David Meredith. Analysis By Compression : Automatic Generation of Compact Geometric Encodings of Musical Objects. In *The Music Encoding Conference*, 2013.

[6] Corentin Louboutin and Frédéric Bimbot. Modeling the Multiscale Structure of Chord Sequences using Polytopic Graphs. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, China, oct 2017.

[7] Frédéric Bimbot, Emmanuel Deruty, Gabriel Sargent, and Emmanuel Vincent. Semiotic Structure Labeling of Music Pieces : Concepts, Methods and Annotation Conventions. In *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR)*, Porto, Portugal, 2012.

[8] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. RWC Music Database : Popular, Classical and Jazz Music Databases. In *Proceedings of the 3rd International Society for Music Information Retrieval Conference (ISMIR)*, volume 2, pages 287–288, 2002.

[9] Corentin Guichaoua. *Compression models and complexity criteria for the description and the inference of music structure*. Theses, Université Rennes 1, 2017.

[10] Taemin Cho. Manually Annotated Chord Data Set of US Pop Songs and Popular Music Collection of RWC Music Database, 2011.

[11] Masataka Goto. AIST Annotation for RWC Music Database. In *Proceedings of the 7th International Society for Music Information Retrieval Conference (ISMIR)*, pages 359–360, 2006.

[12] Frédéric Bimbot, Gabriel Sargent, Emmanuel Deruty, Corentin Guichaoua, and Emmanuel Vincent. Semiotic Description of Music Structure : An Introduction to the Quaero/Metiss Structural Annotations. In *Audio Engineering Society Conference : 53rd International Conference : Semantic Audio*, London, United Kingdom, jan 2014.

[13] Cheng-i Wang and J. Mysore Gautham. Re-Visiting the Music Segmentation Problem With Crowdsourcing. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*, pages 738–744, Suzhou, China, 2017.

# EXPLORATION DE DÉPENDANCES STRUCTURELLES MÉLODIQUES PAR RÉSEAUX DE NEURONES RÉCURRENTS

*Nathan Libermann*
Univ Rennes 1
nathan.libermann@inria.fr

*Frédéric Bimbot*
IRISA/CNRS
frederic.bimbot@irisa.fr

*Emmanuel Vincent*
INRIA
emmanuel.vincent@inria.fr

**RÉSUMÉ**

Dans le cadre de la génération automatique de mélodie structurée, nous explorons la question des dépendances entre les notes d'une mélodie en utilisant des outils d'apprentissage profond. Plus précisément, nous utilisons le modèle d'apprentissage séquentiel GRU, que nous déclinons dans différents scénarios d'apprentissage afin de mieux comprendre les architectures optimales dans ce contexte. Nous souhaitons par ce moyen explorer différentes hypothèses relatives a la non-invariance temporelle des dépendances entre les notes au sein d'un segment structurel (motif, phrase, section). Nous définissons trois types d'architectures récurrentes correspondant à différents schémas d'exploitation de l'historique musical dont nous étudions les capacités d'encodage et de généralisation. Ces expériences sont conduites sur la base de données Lakh MIDI Dataset et plus particulièrement sur un sous-ensemble de 8308 segments mélodiques monophoniques composés de 16 mesures. Les résultats indiquent une distribution non-uniforme des capacités de modélisation et de prédiction des réseaux récurrents testés, suggérant l'utilité d'un modèle non-ergodique pour la génération de segments mélodiques.

## 1. INTRODUCTION

La génération automatique de mélodie est une problématique régulièrement abordée en informatique musicale mais qui reste incomplètement résolue. Récemment revenues au premier plan, les méthodes par réseaux de neurones apparaissent potentiellement capables de modéliser des mécanismes de génération de mélodie par apprentissage à partir d'exemples.

Chen et al [4] furent parmis les premiers auteur à publier sur ce sujet. L'un des principaux problèmes qu'ils relèvent est le manque de structure globale dans les mélodies générées. D'autres auteurs comme Franklin [7], Eck et Schmidhuber [6] ont alors cherché à résoudre ce problème en utilisant des réseaux récurrent LSTM [9] sur un corpus de musique blues ou jazz. Ces travaux ont permis d'améliorer la qualité perçue de la musique générée mais les résultats continuent à présenter une insuffisance de structure. Boulanger-Lewandowski et al. [3] ont tenté de combiner le modèle LSTM avec des modèles génératifs, notamment le RBM [4]. Dans Huang et Wu [10], les auteurs s'intéressent à la représentation des notes dans un espace vectoriel (embedding). Jaques et al [11] proposent de restreindre un modèle LSTM préalablement appris grâce à l'apprentissage par renforcement de règles de musicologie prédéfinies. L'équipe Magenta [1] propose un modèle d'attention inspiré de Bahdanau et al [1]. A notre connaissance, il n'existe pas aujourd'hui de modèle capable d'apprendre à générer des mélodies présentant une structure pleinement satisfaisante à l'échelle de plusieurs mesures consécutives.

Le travail présenté dans cet article est une étude exploratoire qui s'inscrit dans ce cadre de la génération automatique de mélodie structurée, et qui fait appel à des outils d'apprentissage profond. Plus particulièrement nous considérons le modèle séquentiel GRU (Gated Recurrent Units) [5], que nous étudions dans différents scénarios d'apprentissage afin de mieux comprendre les potentialités de cette approche pour la modélisation de mélodies. Nous souhaitons notamment cerner l'importance d'une hypothèse de non-ergodicité (non-invariance dans le temps) de la structure musicale, en mettant en évidence les limites des architectures récurrentes à base de GRU et étudier les possibilités de les adapter à la génération de motifs mélodiques. On suppose en effet qu'il existe une planification dans la construction d'un segment mélodique qui ne se contente pas de se référer aux $k$ précédents éléments pour construire le suivant. Au contraire, nous faisont l'hypothèse qu'un segment mélodique forme un tout et qu'au fil du segment les divers éléments se conditionnent de façon non-adjacente pour former le schéma mélodique global.

Ainsi, dans l'esprit des travaux récents sur les modèles tensoriels/polytopiques de segments musicaux [8] [12], nous émettons l'hypothèse que dans le cadre de mélodies simples constituées de motifs présentant des relations d'analogie, les dépendances structurelles dans la musique ne suivent pas un procédé purement séquentiel, mais plutôt des dépendances multi-échelles. Selon cette approche, un élément musical dépend de façon privilégiée des autres éléments qui se situent dans des positions métriques homologues dans le segment plutôt que dans le voisinage immédiat. Autrement dit, les positions métriques des notes jouent un rôle dans la construction structurelle de la musique et les architectures neuronales doivent en tenir compte.

1. https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn

Nathan Libermann, Frédéric Bimbot, Emmanuel Vincent, *Exploration de dépendances structurelles mélodiques par réseaux de neurones récurrents*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 81–86, 2018. 

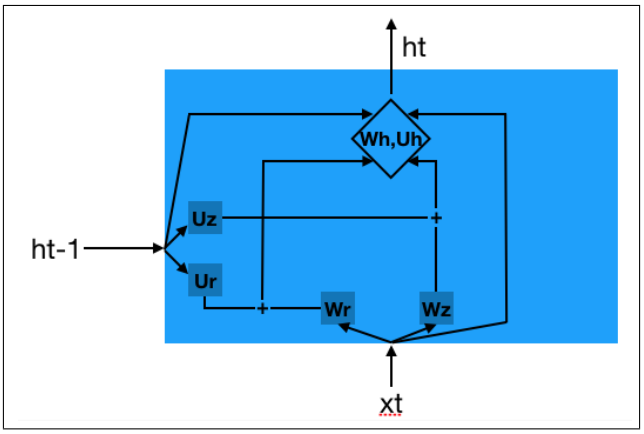



**Figure 1**. Gated Recurrent Units

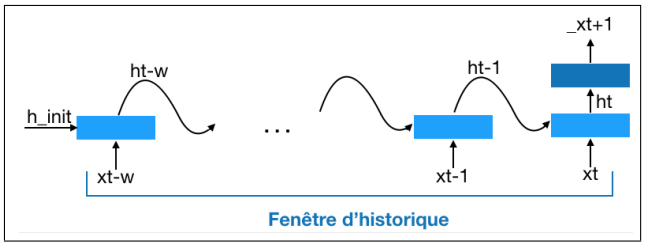



**Figure 2**. Architecture à historique glissant

Nous définissons donc trois modèles récurrents à base de GRU. Un modèle à historique glissant, qui correspond à une façon "standard" de construire et d'entrainer un modèle récurrent. Un modèle à historique croissant, qui correspond à une façon "dynamique" d'apprendre un modèle récurrent. Et enfin, un modèle à historique parallèle, avec des poids distincts selon les positions à prédire.

La comparaison des différents schémas d'apprentissage proposé selon leur capacité d'encodage de l'information musicale et de leur performance de prédiction permet d'étudier la pertinence et les limites de l'hypothèse de non-ergodicité dans les séquences mélodiques.

## 2. PROTOCOLE EXPÉRIMENTAL

### 2.1. Cellule de mémoire GRU et couche de prédiction

Pour définir les architectures étudiées, nous utilisons comme unité de base le réseau de neurones récurrent GRU (*Gated Recurrent Unit*, voir Figure 1) qui fonctionne comme suit. A chaque instant $t$ la cellule GRU reçoit en entrée, sous forme de vecteurs, l'observation courante $x_t$ et une variable interne $h_{t-1}$ qui tient lieu de mémoire des observations précédentes. A partir de ces deux entrées, la cellule GRU produit une remise à jour de $h$, laquelle est ensuite utilisée dans une cellule GRU semblable, prenant $h_t$ et l'observation $x_{t+1}$ en entrée,... et ainsi de suite. Dans nos expériences, $x_t$ est un vecteurs binaire de dimensions $n = 88$ correspondant à un ensemble discret de notes, chaque dimension représentant une note, ce vecteur n'a qu'un seul symbole 1 (one-hot).

La cellule GRU se décompose en 6 sous-ensembles de poids de *propagation d'historique* ($U_h$, $U_r$, $U_z$, $W_h$, $W_r$, $W_z$) qui se combinent avec les entrées $h_{t-1}$ et $x_t$ pour former $h_t$ selon les équations suivantes :

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \quad (1)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \quad (2)$$

$$h_t = (1-z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \quad (3)$$

où $\circ$ désigne le produit matriciel de Hadamard, $\sigma_g$ une fonction sigmoïde et $\sigma_g$ une tangente hyperbolique.

Nous avons aussi besoin pour définir nos architectures d'une couche de prédiction qui à partir d'une mémoire $h_t$ fournit une distribution de probabilités $\_x_{t+1}$ de la note suivante $x_{t+1}$. Pour ce faire, nous utilisons une couche entièrement connectée entre $h_t$ et $\_x_{t+1}$, constituée d'une matrice de poids de *prédiction* $W_p$ et d'un vecteur de biais $b_p$, ce qui fournit en sortie une distribution de probabilité a posteriori sur l'ensemble des notes :

$$\_x_{t+1} = W_p h_t + b_p \quad (4)$$

L'apprentissage est réalisé par rétropropagation du gradient à travers le temps [16].

### 2.2. Spécification des architectures

Dans ce travail nous considérons des segments mélodiques correspondant à des unités structurelles de type phrase ou section musicale. Dans cette optique, nous modélisons des séquences de $N = 64$ notes qui sont obtenues en échantillonnant des mélodies qui s'étendent sur 16 mesures.

Trois architectures à base de GRUs sont étudiées :

— L'architecture à *historique glissant* correspond à une façon standard de construire un réseau récurrent. On choisit une fenêtre d'historique de taille fixe $k$. Pour prédire la note $\_x_{t+1}$, on alimente la couche de prédiction $W_p$ avec l'historique $h_t$, qui, dans ce cas, est initialisé aléatoirement puis propagé depuis $h_{t-k}$ jusqu'à $h_t$ (voir Figure 2). La fenêtre d'historique $[t-k, t]$ étant fixe, on ne peut calculer les prédictions $\_x_{t+1}$ qu'à partir de l'instant $k+1$. Les poids des couches de propagation d'historique sont partagés (c'est-à-dire indépendants de $t$) et il en est de même des poids des couches de prédiction. Ainsi, cette architecture repose sur une hypothèse d'invariance dans le temps des éléments musicaux.

— L'architecture à *historique croissant* correspond à une façon plus dynamique de construire un modèle récurrent. Ici on procède de la même façon que pour l'architecture du modèle à historique glissant, mais on considère une fenêtre d'historique qui couvre l'intégralité de l'intervalle $[1, t]$ et qui par conséquent croît au fur et à mesure que l'on progresse dans le temps (voir Figure 3). Comme dans

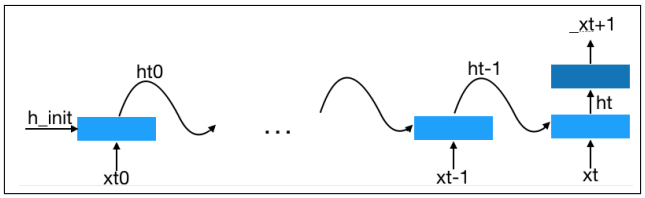



**Figure 3**. Architecture à historique croissant

l'architecture précédente, les poids de la couche de propagation d'historique sont communs pour les différentes positions de notes à prédire et il en est de même pour les poids de la couche de prédiction. Mais dans cette variante, il devient possible de prédire les notes dans toutes les positions temporelles, à partir d'un historique qui va croissant. On note toutefois que cette architecture repose aussi sur une hypothèse d'invariance dans le temps du fait du partage des poids.

— Enfin, l'architecture à *historique parallèle* repose sur le même principe que l'architecture à historique croissant, à ceci près que les poids des couches de propagation et de prédiction sont indépendants pour chaque position à prédire $\_x_{t+1}$. On peut ainsi voir cette configuration comme $N-1$ réseaux à historique croissant, indépendants les uns des autres et correspondant à chaque position à prédire $\_x_{t+1}$. L'indépendance de ces réseaux en fonction de la position est censée permettre à cette architecture de prendre en compte la non-invariance dans le temps et de tenir compte, si nécessaire, de dépendances complexes.

### 2.3. Données

Pour ce travail d'exploration, nous utilisons les données du Lakh MIDI Dataset [14], qui contient 176581 fichiers MIDI multipistes sans appréciation de genres. Etant donné que nous nous intéressons uniquement aux pistes mélodiques monophoniques, nous en avons sélectionné environ 70000 qui possédaient cette propriété dans le corpus originel. De ce sous-ensemble, nous avons extrait 8308 blocs structurels de 16 mesures sous forme de fichier MIDI. Ne disposant pas de segmentation automatique assez fiable, nous sélectionnons uniquement les 16 premières mesures de mélodies monophoniques de type 4/4.

Afin de permettre au modèle de pouvoir mieux extraire les relations relatives entre les notes, nous avons représenté ces mélodies en référence à la première note de la séquence (arbitrairement fixée à Do), de sorte à être indépendant de la tonalité initiale. Nous obtenons donc 8308 séquences mélodiques de 16 mesures.

Pour simplifier la représentation traitée, on considère dans un premier temps une discrétisation de l'information mélodique dans ces segments. Nous avons procédé à un découpage des mesures en 4 portions égales et relevé à chaque fois la note active sur ces positions. Si un silence

| Train | Test | Test |
|---|---|---|
| A | A | B |
| B | B | A |

**Table 1**. Protocoles d'apprentissage et de test pour mesurer les capacités de compression et de généralisation des architectures

apparaît sur l'un des temps considérés, nous prolongeons la valeur de la note précédente ce qui évite d'avoir à gérer des absences de notes.

Le résultat de ce processus de réduction conduit à des séquences "mélodiques" de 16 mesures correspondent toutes à des successions de 64 notes. Chaque note est représentée par un vecteur de dimension 88 (correspondant aux 88 notes d'un clavier de piano). Lorsqu'une note est active, la dimension associée à cette note dans le vecteur est mise à 1 et toutes les autres à 0.

### 2.4. Protocole

Le corpus composé de 8308 séquences est divisé en deux groupes A et B comprenant chacun 4154 séquences. Nous définissons quatre scénarios pour chaque architecture voir Table 1 :

— Apprentissage sur le groupe A, test sur le groupe A
— Apprentissage sur le groupe A, test sur le groupe B
— Apprentissage sur le groupe B, test sur le groupe A
— Apprentissage sur le groupe B, test sur le groupe B

Lors du test, nous relevons pour chaque position de note, l'erreur moyenne sur l'ensemble des exemples de test, ceci dans le cadre des 3 architectures décrites dans la section 2.2.

Les mesures effectuées en utilisant le même ensemble de test que celui utilisé pour l'apprentissage permettent de caractériser les capacités de *compression* de l'information mélodique par les différentes architectures, en fonction de la position de la note dans le segment musical. Celles qui croisent les ensembles d'apprentissage et de test rendent compte la capacité de *généralisation* des architectures à des données nouvelles.

### 2.5. Critère d'évaluation

Pour chaque instant $t$, nous calculons l'erreur entre le vecteur de sortie $\_x_t$ et la note réelle $x_t$ par la fonction d'erreur quadratique moyenne ($MSE$). On rappelle que $\_x$ est un vecteur de dimension $n = 88$ correspondant à une distribution de probabilités et $x$ un vecteur one-hot de même dimension, correspondant à la note effectivement active.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\_x_i - x_i)^2 \qquad (5)$$

C'est l'évolution de cette erreur que nous examinons pour chaque position de note et dans chaque scénario, afin d'observer la façon dont les différentes architectures encodent les dépendances structurelles. Nous analysons à

la fois les motifs engendrés par la variation de l'erreur moyenne entre les différentes positions ainsi que le critère de compression / prédiction.

### 2.6. Détails complémentaires de mise en oeuvre

Pour ces expériences nous utilisons la bibliothèque d'apprentissage profond Pytorch [2]. L'historique est un vecteur de dimension 100. Chaque exemple est présenté 40 fois au cours de l'apprentissage (40 epochs). L'optimiseur utilisé est Adagrad.

## 3. RÉSULTAT

Les figures 4 et 5 illustrent les résultats du protocole expérimental décrit dans la section précédente. La figure 4 correspond aux différentes architectures apprises à partir du corpus A et la figure 5 aux architectures apprises sur le corpus B.

Pour chacune de ces figures, les résultats de généralisation des architectures sont représentés sur les graphiques Ag, Bg, Cg et les résultats de compression sur les graphiques Ao, Bo et Co.

— graphique A : architecture à historique parallèle
— graphique B : architecture à historique croissant
— graphique C : architecture à historique glissant

En premier lieu, on constate une similarité de comportement entre courbes homologues sur la figure 4 et la figure 5. Ceci permet de penser que les tendances observées sont relativement peu influencées par les spécificités des deux demi-corpus.

On remarque ensuite que pour les architectures à historique glissant et croissant, les valeurs des capacités de généralisation (Bg et Cg) présentent un niveau d'erreur à peine plus élévé que les valeurs des capacités de compression (Bo et Co). A l'inverse, les capacités de compression et de généralisation de l'architecture parallèle se comportent très différemment l'une de l'autre. En effet, l'architecture parallèle possède beaucoup plus de paramètres libres ce qui crée une capacité bien plus forte du réseau à comprimer les données d'apprentissage mais entraîne un phénomène d'over-fitting (surapprentissage) qui altère la capacité de généralisation sur de nouvelle données. Ce phénomène est bien moins saillant sur les deux autres architectures.

Un examen plus précis des motifs observés sur les courbes d'erreur apporte également des observations intéressantes.

Les courbes pour les historiques glissant et croissant présentent une pseudo-periodicité marquée à l'échelle de 8 notes et des oscillations secondaires aux échelles 4 et 2, suggérant fortement une "synchronisation" des capacités de modélisation des architectures correspondantes sur des cycles de 2 mesures.

Les courbes Ao (que ce soit pour la figure 4 ou la figure 5) présentent pour leur part un comportement plus contrasté : tout d'abord une décroissance très nette sur le premier quart du segment puis des variations plus erratiques sur la partie centrale (2ème et 3ème quart) et enfin une remontée globale de l'erreur sur le dernier quart. On peut opérer un rapprochement entre ces observations expérimentales et différentes considérations musicologiques sur la structure des segments musicaux : notamment que l'on constate souvent la réalisation de formules musicales conventionnelles en fin de segment, donc finalement moins prédictibles en fonction du contexte, puisqu'elle sont plus ou moins prédéterminées d'avance indépendemment de celui-ci.

Par ailleurs, selon le modèle cognitif d'Implication-Réalisation de Narmour [13] (et son extension récente [2]), les fins de segments structurels constituent fréquemment des dénis d'implication par rapport aux progression musicales établies dans les portions antérieures, ce qui peut également constituer une hypothèse expliquant le comportement observé sur les courbes Ao. Toutefois ce comportement étant moins net sur les courbes Ag, des expériences complémentaires sont requises pour mieux asseoir ces hypothèses.

## 4. CONCLUSIONS

Dans ce travail d'exploration, nous avons considéré trois architectures de réseaux de neurones récurrents et nous avons analysé leur comportement en terme d'erreur de modélisation de schémas mélodiques simplifiés.

Ces expériences nous ont permis d'observer que la prise en compte de la non-invariance dans le temps de la structure musicale dans une architecture de réseau de neurones récurrent permet de rendre compte de façon plus fine de la structure globale des mélodies apprises par le réseau.

Dans le cadre de nos expériences actuelles, cette conclusion demeure partielle, du fait d'un volume de données d'apprentissage limité qui entraîne une capacité insuffisante de généralisation du réseau appris.

Toutefois, cette première exploration de l'hypothèse de non-invariance dans le temps de la structure musicale par des réseaux de neurones récurrents nous conforte dans l'idée de poursuivre nos recherches dans cette voie.

Plusieurs pistes sont prévues pour compléter ces travaux à court terme, permettant ainsi d'enrichir ces premières investigations par des expériences supplémentaires et les résultats correspondants.

## 5. REMERCIEMENT

Ce travail a reçu le support de l'Agence Nationale de la Recherche dans le cadre du projet DYCI2 *Dynamiques Créatives de l'Interaction Improvisée* (ANR-14-CE24-0002-01) et de la Région Bretagne.

2. http://www.pytorch.org

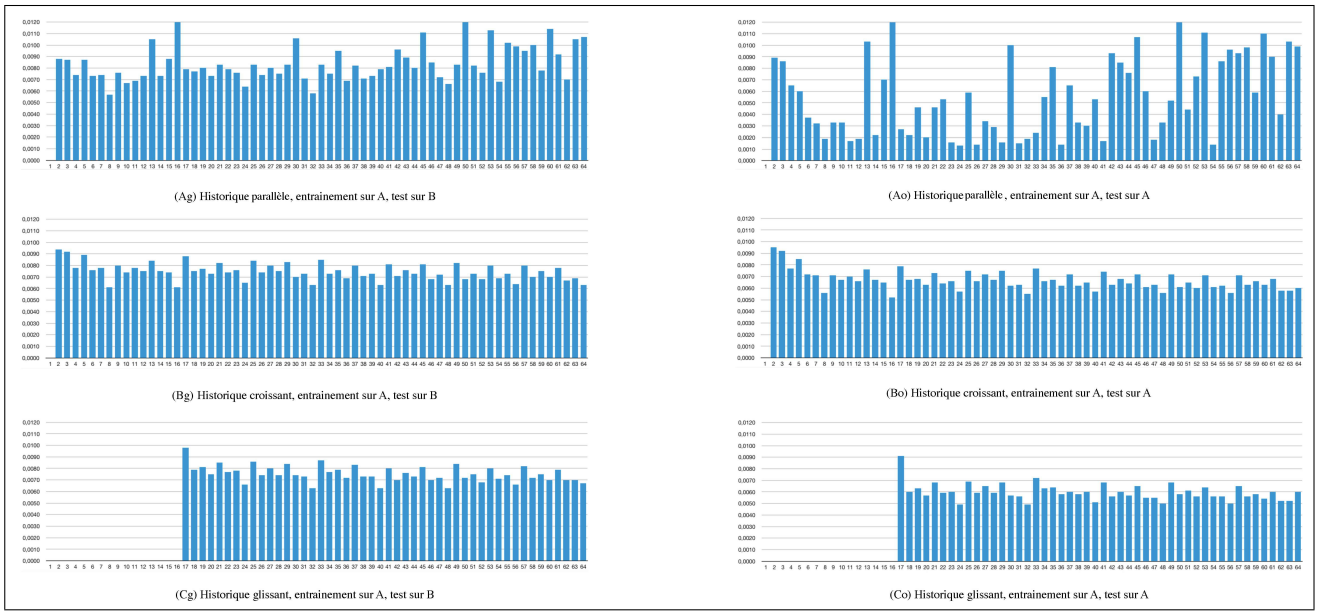



**Figure 4**. A. Erreur de prédiction moyenne pour chaque position.

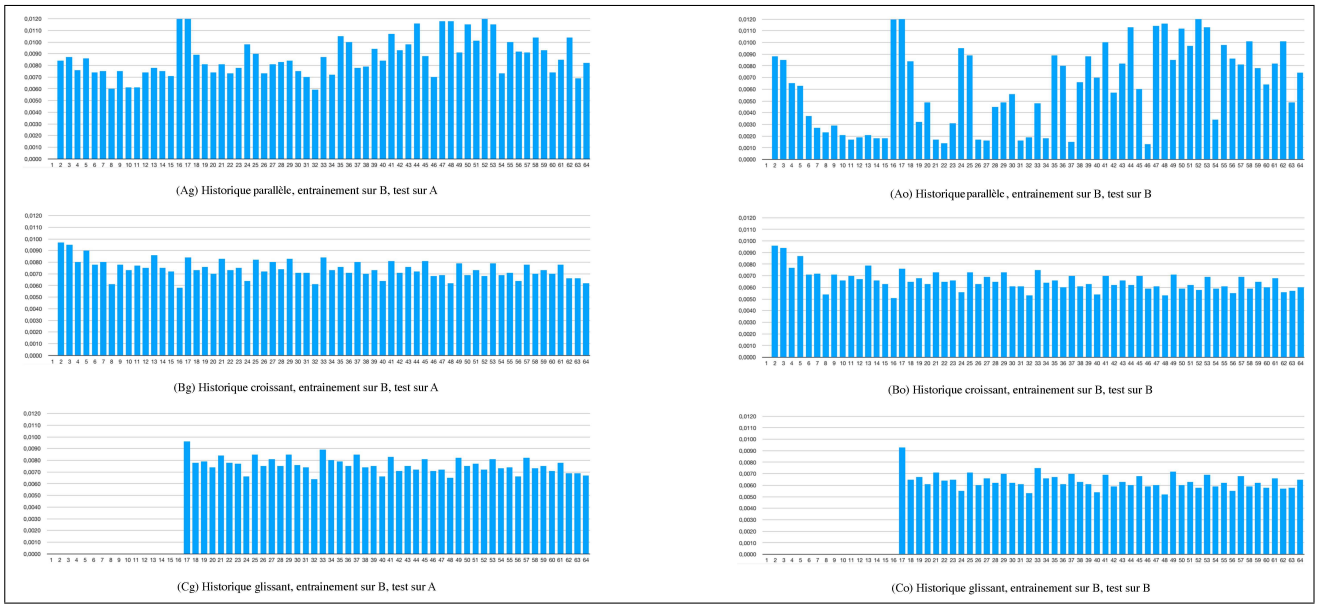



**Figure 5**. B. Erreur de prédiction moyenne pour chaque position.

# Références


[1] Bahdanau, D. Cho, K. Bengio, Y. "Neural machine translation by jointly learning to align and translate", *arXiv preprint arXiv :1409.0473*, 2016.

[2] Bimbot, F. Deruty, E. Sargent, G. Vincent, E. "System & contrast : a polymorphous model of the inner organization of structural segments within music pieces", *Music Perception, University of California Press*, California, USA, 2016.

[3] Boulanger-Lewandowski, N. Bengio, Y. Vincent, P. "Modeling temporal dependencies in high-dimensional sequences : application to polyphonic music generation and transcription", *arXiv preprint arXiv :1206.6392*, 2012.

[4] Chen, J., Miikkulainen, R. "Creating melodies with evolving recurrent neural networks", *Proceedings of the 2001 International Joint Conference on Neural Networks. IEEE*, Washington, USA, 2001.

[5] Cho, K. Van Merriënboer, B. Gulcehre, C. Bahdanau, D. Bougares, F. Schwenk, H. Bengio, Y. "Learning phrase representations using RNN encoder-decoder for statistical machine translation", *Conference on Empirical Methods in Natural Language Processing*, Qatar, 2014.

[6] Eck, D. Schmidhuber, J. "Finding temporal structure in music : blues improvisation with LSTM recurrent networks", *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, 2002.

[7] Franklin, J. "Jazz melody generation from recurrent network learning of several human melodies", *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference*, Florida, USA, 2005.

[8] Guichaoua, C. "Modèles de compression et critères de complexité pour la description et l'inférence de structure musicale", *Thèse supervisé par Bimbot, F.*, France, 2017.

[9] Hochreiter, S. Schmidhuber, J. "Long short-term memory", *Neural computation 9 (8), 1735-1780*, Massachusetts, USA, 1997.

[10] Huang, A. Wu, R. "Deep learning for music", *arXiv preprint arXiv :1606.04930*, 2016.

[11] Jaques, N. Gu, S. Turner, R. Eck D. "Generating music by fine-tuning recurrent neural networks with reinforcement learning", *Deep Reinforcement Learning Workshop, NIPS*, California, USA, 2017.

[12] Louboutin, C. Bimbot, F. "Description of chord progressions by minimal transport graphs using the system & sontrast model", *Proceedings of the 42nd International Computer Music Conference*, Netherlands, 2016.

[13] Narmour, E. "The analysis and cognition of basic melodic structures : the implication-realization model", *Univ. of Chicago Press*, Illinois, USA, 1990.

[14] Raffel, C. "Learning-based methods for comparing sequences, with applications to audio-to-MIDI alignment and matching", *PhD Thesis*, Colombia, USA, 2016.

[15] Smolensky, P. "Information processing in dynamical systems : foundations of harmony theory", *MIT Press Cambridge*, Massachusetts, USA, 1986.

[16] Werbos, P. "Backpropagation through time : what it does and how to do it", *Proceedings of the IEEE volume 78*, 1990.

# ÉVALUATION DE LA CORRECTION RYTHMIQUE DES PARTITIONS NUMÉRISÉES

*Francesco Foscarin*
CEDRIC, CNAM, Paris
francesco.foscarin@cnam.fr

*Raphaël Fournier-S'niehotta*
CEDRIC, CNAM, Paris
fournier@cnam.fr

*Florent Jacquemard*
Inria, Paris
florent.jacquemard@inria.fr

*Philippe Rigaux*
CEDRIC, CNAM, Paris
philippe.rigaux@cnam.fr

**Résumé**

Nous proposons une représentation des éléments de gravure musicale ayant trait au rythme – figures de notes, tuplets, ligatures, *etc.* – sous la forme d'arbres. Cette modélisation permet de lever les ambiguités et redondances contenues dans les différents encodages de partitions numériques, qui sont des sources d'incohérences majeures.

Le modèle est développé théoriquement, puis nous présentons son intégration dans des procédures de vérification d'encodages. Il est également montré que l'on peut l'utiliser pour des tâches de génération de partitions.

## 1. INTRODUCTION

Aujourd'hui, produire des partitions musicales numérisées de bonne qualité demande un effort important. En effet, il faut pour cela inspecter à l'œil le rendu graphique après numérisation pour y dénicher les erreurs, ce qui s'avère difficile en raison de la complexité sémiologique de la notation musicale.

Dans le cadre de l'édition collaborative, ce problème est particulièrement important, puisque chaque contributeur est susceptible d'utiliser son propre logiciel de gravure. Les encodages de documents musicaux comme MusicXML [6] ou MEI [13, 10] sont largement utilisés comme formats d'échange, ainsi que pour le stockage dans des bases de partitions numériques dont le contenu peut être inspecté, modifié et indexé. Ces encodages XML sont très permissifs et n'effectuent pas de vérification de la correction, de la cohérence ou de la complétude d'une partition. Leur grande flexibilité est bien adaptée à la prise en compte des évolutions et variations de la notation musicale, mais en contre-partie, laisse aux humains une partie délicate et couteuse de la production de partitions.

Nous proposons dans cet article une modélisation d'éléments de gravure, en vue d'assister les créateurs de partitions dans des tâches de vérification des données encodées. Nous nous limitons à la partie rythmique de la notation. Les problématiques auxquelles nous apportons une réponse sont détaillées dans la section 2. Nous présentons ensuite notre modèle dans la section 3, des projets en cours autour de leur application à l'analyse de partitions (section 4) ainsi que des perspectives d'applications dans le cadre de tâches impliquant la génération de partitions numériques (section 5). Nous développons en particulier les contributions suivantes :

– un formalisme de modélisation des éléments rythmiques de notation musicale, reposant sur deux structures sous-jacentes complémentaires : les arbres de *ligature* et les arbres de *tuple* (pour les groupements de notes tels que les triolets) ;

– un développement de ce formalisme pour *valider* des partitions existantes, c'est-à-dire en éliminer les ambiguïtés et redondances présentes dans l'encodage XML ;

– une proposition de génération de partitions dans un format XML, à partir de ces structures d'arbres ;

– l'intégration de cette modélisation dans une plateforme en ligne proposant d'ores et déjà de l'analyse automatisée de la qualité de partitions musicales.

## 2. ÉLÉMENTS DE NOTATION RYTHMIQUE

Les encodages MusicXML et MEI permettent de décrire dans un même document à la fois des informations relatives au contenu musical d'une pièce, indépendamment de la manière dont ce contenu est rendu sur une partition imprimée, et des informations relatives à l'apparence graphique de ce contenu.

Par exemple en MusicXML la *durée* de chaque événement est décrite dans un élément XML `<duration>`, qui contient une valeur exprimée en fraction de noire (*quarter-length*), ou plus précisément en nombre de divisions élémentaires de la noire. Le nombre maximal de divisions d'une noire est par ailleurs spécifié en début de voix (avec l'élément `<divisions>`), suivant l'approche du standard MIDI. Par exemple, pour un

Francesco Foscarin, Florent Jacquemard, Raphaël Fournier-S'Niehotta, Philippe Rigaux, *Évaluation de la correction rythmique des partitions numérisées*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 87–95, 2018. 

nombre de divisions de la noire de 120, dans une mesure à 4/4, une durée de 120 correspond à une ♩, 60 à une ♪, 30 à une 𝅘𝅥𝅯, 40 à une ♪ de triolet, *etc.*

Par ailleurs, le *type* de chaque note est également spécifié dans un élément `<type>`, qui prendra par exemple la valeur `quarter` pour ♩, `eighth` pour ♪, `16th` pour 𝅘𝅥𝅯, *etc.* Cette valeur peut être déduite de la durée spécifiée comme ci-dessus (en prenant en compte le nombre de divisions de la noire), mais elle est aussi écrite explicitement dans le document, pour décrire la figure de note qui sera imprimée dans la partition. Ainsi, pour l'exemple du paragraphe précédent, le même type de note ♪ correspond aux deux durées 60 et 40.

La correspondance entre durée et type de note dépend aussi du contexte de la note, et plus précisément du groupe dans lequel elle figure. Par exemple, les trois premières noires de la première mesure de la figure 1 ont une durée de $\frac{2}{3}$ de noire, car elles forment un triolet de noires.

Dans la suite de cet article, nous utiliserons le terme *tuple*, issu du jargon informatique, pour désigner les groupes de notes correspondant à une valeur rythmique quelconque (noire, blanche croche), tels que les triolets notés avec le chiffre 3, ou plus généralement d'autres divisions composées d'un ensemble impair de notes [5].

Tous les tuples de notes définissent des durées musicales suivant une logique de division d'un intervalle temporel en parts égales. La notation d'un tuple par $n : m$ (`<time-modification>`, en MusicXML) signifiant une division en $n$ parts dans le même temps que $m$, précise à la fois la longueur de l'intervalle divisé ($m$ valeurs) et le nombre $n$ de divisions du tuple. Cependant, du point de vue de la gravure, suivant le contexte et la signature rythmique en vigueur, les entiers $m$ et $n$ devront figurer explicitement ou non dans une partition. Ainsi, en figure 1, tous les 3 signifient 3 : 2, le 2 étant omis.

Les tuples de notes peuvent être imbriqués suivant un système hiérarchique de divisions. Les limites d'un tuple sont explicitées par des crochets ou des *ligatures* (*beams*). Les ligatures ne sont pas formellement nécessaires pour la définition des durées de notes en notation musicale (les figures de notes et indications de tuples étant a priori suffisantes), leur usage est essentiel pour faciliter la lecture d'une partition, en particulier la compréhension de la métrique. Par exemple, dans la figure 1, les tuples divisant des durées d'un temps à la fin des mesures 1 et 3 sont ligaturés. En MusicXML, les début et fin de tuples et de ligatures (ainsi que les continuations de ligatures ou les ligatures partielles) doivent être indiqués explicitement à chaque note, par des éléments `<tuplet>` et `<beam>`, dont la valeur (pour le second) ou une valeur d'attribut (pour le premier) indique s'il s'agit d'un début, d'une fin, d'une continuation, *etc.* Cependant, les imbrications de tuples ou ligatures (tels que ceux de la figure 1) ne sont pas représentés par des imbrications d'éléments XML. Ainsi, un document XML valide pourra contenir des éléments marquant les début et fin de tuple/ligature qui ne sont pas bien appariés (en général, le chargement d'un tel document dans un outil comme Finale va échouer). Cela peut induire des incohérences et complique l'implémentation de conventions.

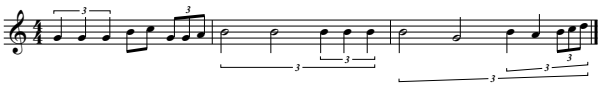


**Figure 1** : Tuples imbriqués avec ou sans ligatures. Dans cet article, le terme *ligature* (*beams*) se réfère aux barres utilisées pour regrouper des croches, à ne pas confondre avec les liaisons.

En outre, la relation entre les durées et les types de notes peut aussi être modifiée par l'ajout de *liaisons* (`<tie>`) et de *points*.

Les interdépendances décrites ci-dessus induisent des redondances pour les données présentes dans les encodages de partitions, en particulier en ce qui concerne les aspects rythmiques. Il existe alors des risques d'incohérences dans les encodages, par exemple si les durées (de l'élément `<duration>`) ne correspondent pas aux figures de notes spécifiées (élément `<type>`). D'autres risques d'incohérences structurelles sont liés aux éventuels problèmes d'appariement mentionnés ci-dessus, par exemple avec la spécification d'un tuple ouvert et non fermé.

Une manière d'éviter de telles incohérences pourrait passer par l'expression de *contraintes d'intégrité* (dans un formalisme adéquat) et leur vérification au chargement d'un document. Les schémas XML utilisés actuellement pour les partitions numériques n'incluent pas de telles contraintes. Un document valide d'un point de vue syntaxique (*i.e.* valide par rapport au schéma XML considéré) pourra donc contenir des erreurs, le rendant inutilisable.

Modifier les encodages existants pour ajouter des contraintes d'intégrité ne semble pas une solution réaliste. Premièrement, en raison de la multiplicité des formats considérés comme standards. Deuxièmement, parce qu'en plus des règles strictes d'intégrité (par exemple pour les durées), un certain nombre de règles d'usage, comme la notation des tuples et des ligatures (*cf.* figure 8), relèvent plus de conventions, et les coder de façon définitive dans un schéma serait trop restrictif. De plus, les règles pourront varier suivant le style musical considéré – on peut citer l'exemple des valeurs ajoutées en musique contemporaine.

Pour vérifier qu'une règle ou une convention est satisfaite dans une partition donnée, ou en tenir compte dans des tâches générant une partition, nous choisissons de l'exprimer sous forme de règles simples, portées par un formalisme non ambigu et indépendant des encodages, présenté dans la section suivante.

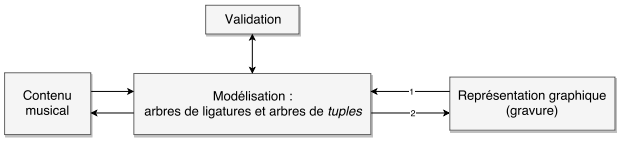



**Figure 2** : Le modèle proposé et son intérêt pour la production de partitions. La flèche 1 correspond aux processus décrits dans la section 4 de cet article, la flèche 2 à ceux de la section 5.

## 3. MODÈLE ABSTRAIT DE GRAVURE

Afin de répondre aux problèmes présentés ci-dessus, nous proposons un modèle abstrait d'éléments de notation rythmique, orienté vers les informations graphiques relatives à la gravure. La figure 2 illustre l'intérêt de notre modélisation pour la production de partitions.

Suivant des travaux antérieurs [9, 1], nous choisissons des représentations de la notation rythmique dans des structures d'arbres. En effet, un tel codage est naturel pour le rythme dans la mesure où il reflète le principe de division hiérarchique des durées en musique [11]. Une autre motivation pour le choix de cette structure de données standard est de permettre d'exprimer simplement des règles de notation à satisfaire et d'en automatiser efficacement la vérification.

Nous avons pris soin d'éviter la redondance d'information dans les modèles proposés. Ainsi, les informations non représentées explicitement, telles que les durées de notes, peuvent être calculées à partir des informations présentes dans le modèle.

Nous avons implémenté des procédures de traduction depuis des encodages XML de partitions vers des arbres et dans la direction inverse, de manière à exploiter ce modèle d'une part dans des tâches de vérification de contraintes d'intégrité appliquées à des partitions XML existantes, et d'autre part pour la génération de partitions XML correctes.

Nous considérons deux types d'arbres dans les traductions, correspondant à deux aspects : les arbres de ligature d'une part, les arbres de tuple d'autre part. Nous les présentons ci-dessous.

### 3.1. Arbres de ligature

À chaque mesure de chaque voix d'une partition nous associons un arbre représentant à la fois les figures de notes et les ligatures entre notes.

**Définition.** Dans un arbre, s'il existe une arête $\langle v, v' \rangle$, on dit que le sommet $v'$ est le *successeur* de $v$. Un sommet qui n'est le successeur d'aucun autre est appelé *racine*. Il n'existe qu'une racine par arbre considéré. Un sommet sans successeur est appelé *feuille*. Les sommets qui ne sont pas des feuilles sont appelés *internes*.

Chaque feuille d'un *arbre de ligature* est étiquetée par un symbole représentant les informations suivantes :

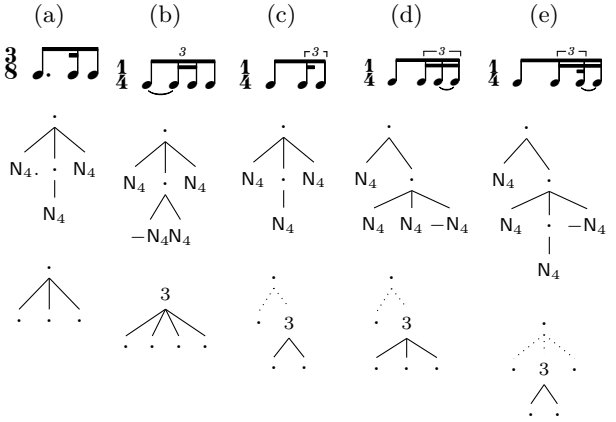



**Figure 3** : Différentes gravures d'un même rythme (en haut), les arbres de ligature (milieu) et les arbres de tuple associés (en bas).

- une *figure de tête de note*, parmi 𝅜, 𝅝, 𝅗𝅥, et ♩. Ces figures sont notées respectivement $\mathsf{N}_{\frac{1}{2}}$, $\mathsf{N}_1$, $\mathsf{N}_2$, $\mathsf{N}_4$, l'indice indiquant une durée en fraction de ronde (suivant l'usage dans certains encodages dont MusicXML). On pourrait ajouter des symboles plus longs que 𝅜 avec $\mathsf{N}_{\frac{1}{4}}$, *etc.* En revanche, les durées plus courtes que ♩ (représentées en notation musicale par des crochets – ou ligatures, et non des têtes de notes particulières) sont codées dans la structure de l'arbre comme nous le verrons plus bas.
- optionnellement, un ou plusieurs *points*, chaque point augmentant la durée de la moitié de sa valeur (pour le premier point) ou de la valeur du point précédent (pour les suivants).
- optionnellement, une *liaison* (*tie*), notée par '-' au début du symbole, indiquant que la durée du symbole s'ajoute à celle du symbole précédent dans l'arbre (suivant un parcours des feuilles en profondeur d'abord).

On utilise également des symboles de silences construits autour de $\mathsf{R}_{\frac{1}{2}}$, $\mathsf{R}_1$, $\mathsf{R}_2$, $\mathsf{R}_4$ suivant le même principe. On trouve différents exemples de symboles de feuilles dans les arbres de ligature de la figure 3.

Les nœuds internes des arbres de ligature sont sans étiquette. Nous les représentons avec un · dans la figure 3.

Les arêtes d'un arbre de ligature peuvent être de deux types. Les *arêtes pleines* se propagent vers les feuilles : si $\langle v, v' \rangle$ est une arête pleine, toute arête sous $v'$ est pleine. Chaque arête pleine représente un crochet dans le cas de notes isolées, ou une ligature dans le cas de notes regroupées. Plus précisément, pour une feuille étiquetée par $\mathsf{N}_4$ (figure de note ♩), le nombre de crochets (ou ligatures) est le nombre d'arêtes pleines sur le chemin entre cette feuille et la racine. Donc une feuille $\mathsf{N}_4$ avec deux arêtes pleines jusqu'à la racine représente une durée de 𝅘𝅥𝅯. C'est

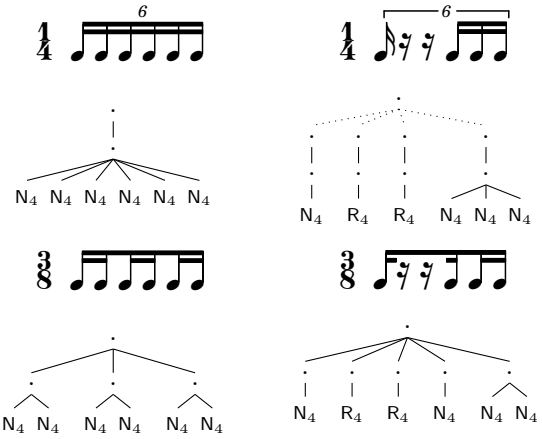



**Figure 4** : Arbres de ligature.

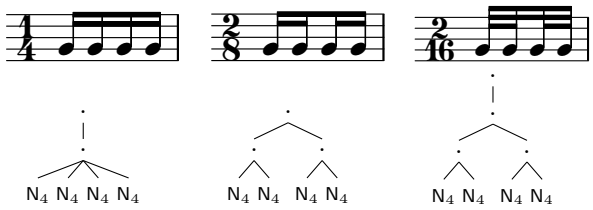



**Figure 5** : Arbres de ligature (examples de [7]).

le cas par exemple de la première note, isolée, dans l'exemple en haut à droite de la figure 4 et des trois dernières notes de cette mesure (qui sont groupées par une ligature).

Le principe est le même pour les silences, comme on peut le voir dans la même mesure de la figure 4, et la mesure en bas à droite de la même figure.

Les arêtes du second type sont représentées en pointillés, et n'ont pas la signification des premières. Elle sont simplement utilisées pour ajouter des événements qui ne sont pas liés par des ligatures (c'est le cas en particulier des silences), dans un arbre où les événements sont lus dans un parcours en profondeur d'abord.

Les arêtes des arbres de ligature apportent des informations quand à la gravure des figures de rythmes, ainsi que décrit formellement dans le paragraphe ci-dessous. En revanche, les durées des notes représentées ne peuvent être calculées à partir des arbres de ligature seuls. L'utilisation conjointe des arbres de ligature et des arbres de tuple décrits au paragraphe 3.2 est nécessaire pour déduire les informations sémantiques de durées.

**Propriétés.** Appelons *profondeur pleine* d'un sommet d'un arbre de ligature le nombre d'arêtes pleines sur le chemin entre ce sommet et la racine.

Nous avons vu que dans le cas d'une feuille, la profondeur pleine indique le nombre de crochets ou de ligatures pour l'élément correspondant.

Le plus petit ancêtre commun de deux sommets $n_1$ et $n_2$ d'un arbre de ligature est noté $lca(n_1, n_2)$, et

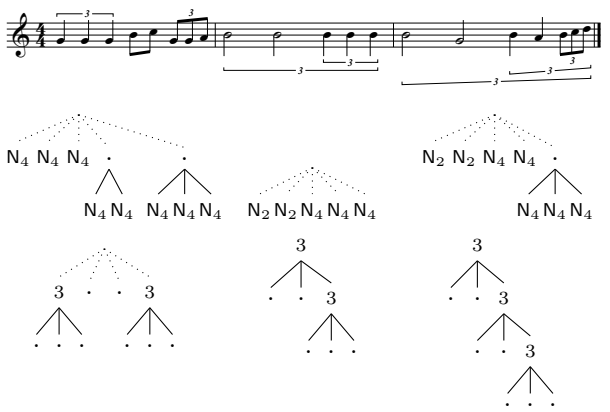



**Figure 6** : Arbres de ligature et arbres de tuples pour les mesures de la figure 1.

$plca(n_1, n_2)$ est le plus petit ancêtre commun de $n_1$ et $n_2$ en ne passant que par des arêtes pleines. Plus formellement, $plca(n_1, n_2) = lca(n_1, n_2)$ si le chemin entre $n_1$ et $lca(n_1, n_2)$ ainsi que le chemin entre $n_2$ et $lca(n_1, n_2)$ ne contiennent que des arêtes pleines, et $plca(n_1, n_2)$ n'est pas défini sinon.

La propriété suivante des arbres de ligature est utilisée dans les applications présentées ci-dessous :
Le *nombre de ligatures* entre deux feuilles $\ell_1$, $\ell_2$ est la profondeur pleine de $plca(\ell_1, \ell_2)$ plus 1, si $plca(\ell_1, \ell_2)$ est défini, et 0 sinon.

On peut par exemple vérifier cette propriété sur la figure 5. Le nombre de ligatures entre chaque paire de notes successives de la première mesure est 2 (car la profondeur pleine de leur *plca* est 1). Dans le cas de la deuxième (resp. troisième) mesure de la figure 5, le nombre de ligatures entre la première et deuxième note, ainsi qu'entre la troisième et quatrième note est 2 (resp. 3), mais le nombre de ligatures entre la deuxième et troisième note n'est que de 1 (resp. 2). Nous verrons en section 4.2 que cela correspond à des conventions de notation des ligatures.

### 3.2. Arbres de tuple

Les *arbres de tuple* représentent les éléments de notation des tuples, plus précisément les emplacements de chiffres et crochets définissant explicitement dans une partition la nature des tuples. Les figures 3 et 6 présentent des exemples d'arbres de tuple.

**Définition.** Chaque nœud interne d'un arbre de tuple est étiquetés ou bien par une paire d'entiers $n : m$, $n, m \geq 2$, signifiant $n$ éléments à la place de $m$, ou bien par un seul entier $n$ si $m$ peut être omis, ou bien pas étiqueté (représentés avec un $\cdot$ dans les figures). Les feuilles de ces arbres sont sans étiquette.

**Propiétés.** La sémantique d'un arbre de tuple est la suivante. Un arbre de tuple et son arbre de ligature associé ont toujours le même nombre de feuilles, correspondants au nombre de figures de notes représentées. Lorsqu'un nœud interne est étiqueté par $n$

ou $n : m$, la suite des éléments correspondant aux feuilles sous le nœud interne se voit affectée (dans la notation rythmique) la même étiquette ($n$ ou $n : m$). L'étiquette est notée ou bien au-dessus de la ligature définie par l'arbre de ligature, ou bien au dessus d'un crochet ajouté dans le cas où l'arbre de ligature ne définit pas de ligature entre les éléments concernés.

Par exemple, dans le cas de la première mesure figure 6, la suite des trois premières notes est étiquetée par 3 (avec un crochet), la suite des trois dernières notes est étiquetée aussi par 3 (sans crochet car ces trois notes sont ligaturées comme défini par l'arbre de ligature), et les deux notes du milieu n'ont pas d'étiquette de tuple.

### 3.3. Calcul des durées et complémentarité

Notons que, pas plus qu'un arbre de ligature, un arbre de tuple seul ne permet pas de déduire les informations sémantiques sur les durées de notes. L'utilisation conjointe d'un arbre de ligature et d'un arbre de tuple est nécessaire pour cela (cf. paragraph 4.2). En se sens, les deux types d'arbres sont complémentaires.

L'introduction de ces deux types d'arbres complémentaires s'est avéré nécessaire car il existe des cas où il n'est pas possible de coder simplement dans un même arbre les ligatures et les informations sur les notations de tuples. Considérons par exemple la mesure (c) de la figure 3. Les deux dernières feuilles correspondent à des éléments appartenant au même triplet (triolet de doubles). Mais dans l'arbre de ligature, les deux dernières feuilles ne sont pas à des positions frères. Par conséquent il n'est pas possible d'ajouter l'étiquette 3 de triolet à un nœud interne de l'arbre de ligature : le nœud parent de l'avant-dernière feuille ne concernerait que cette feuille, et la racine concernerait l'ensemble de la figure (un 3 à la racine correspondrait à un triolet de croches). Le rôle de l'arbre de tuple est ici donc d'offrir un nœud interne parent de ces deux dernière feuilles pour introduire la notation 3.

Les durées de notes peuvent être calculées à partir des étiquettes des arbres de ligature et tuple, et de leur structure : nous avons vu que ces deux types d'arbres fournissent les informations nécessaires pour la gravure de rythmes (figures de notes, points, liaisons, ligatures et tuplets). Les durées peuvent donc être déterminées à partir des rythmes gravés suivant les règles du solfège.

Plus précisément, l'arbre de ligature permet de déduire pour chaque feuille une durée *temporaire*, suivant l'étiquette de cette feuille et sa profondeur pleine dans l'arbre (qui indique rappelons-nous, le nombre le nombre de crochets ou de ligatures). Par exemple, pour le cas (b) de la figure 3, la durée temporaire de la première note est $\frac{1}{8}$ (de ronde), déduit de l'étiquette $\mathsf{N}_4$ (correspondant à $\frac{1}{4}$ de ronde) et la profondeur pleine 1 (correspondant à une ligature, donc une division par 2 de $\frac{1}{4}$). C'est aussi le cas de la dernière note. La durée temporaires des deuxième et troisième feuilles est $\frac{1}{16}$. Puis cette durée temporaire est ajustée suivant les informations contenues dans l'arbre de tuple : si une feuille est sous un nœud étiquetté $n : m$ (avec $m = 2$ dans le cas où le second entier est manquant), on applique le coefficient $\frac{m}{n}$. Dans l'exemple ci-dessus, on obtient une durée de $\frac{1}{12} = \frac{1}{8} \times \frac{2}{3}$ pour la note correspondant à la première et la dernière feuille, et $\frac{1}{24}$ pour les deuxième et troisième notes.

Dans le cas (c) de la même figure, la durée de la première note est $\frac{1}{2}$ (ici le coefficient d'ajustement est 1), la durée de la seconde note est $\frac{1}{24} = \frac{1}{16} \times \frac{2}{3}$, et la durée de la dernière note est $\frac{1}{12} = \frac{1}{8} \times \frac{2}{3}$.

### 3.4. Autres représentations

L'environnement d'assistance à la composition OpenMusic [3] utilise une structure appelée *arbres de rythme* (OMRT) pour la représentation des données musicales temporelles symboliques. Les OMRT suivent le principe de la définition de durées symboliques par divisions récursives d'intervalles temporels, et sont donc proches conceptuellement de la notation rythmique traditionnelle. Cela les rend très utiles dans un contexte d'aide à la création et de manipulation algorithmique de structure rythmiques complexes. Leur usage dans un contexte de gravure des rythmes a aussi été étudié dans [1].

Les arbres de ligature et de tuple présentés ici diffèrent des OMRT dans la mesure où ils mettent l'accent sur la représentation graphique de la notation, et non sa définition logique (i.e. sa sémantique).

En section 5, nous suggérons des applications où les arbres de ligature et tuple sont utilisés comme un intermédiaire entre les OMRT (ou des arbres de rythmes similaires) et la notation finale dans un contexte d'aide à la gravure.

## 4. APPLICATIONS À L'ÉVALUATION DE PARTITIONS NUMÉRISÉES

Nous présentons dans cette section la validation de partitions numérisées à l'aide de notre modélisation sous forme d'arbres, ce qui correspond à la flèche 1 de la figure 2.

### 4.1. Importer des fichiers XML

Nous avons écrit des procédures d'extraction d'arbres de ligature et arbres de tuple à partir de fichiers MusicXML et MEI.

La construction des arbres se fait séparément (en deux passes) à partir des informations dans le fichier XML sur les structures de ligatures et de tuples, *e.g.* dans le cas de MusicXML à partir des éléments respectivement de la forme `<beam>` et `<tuple>`.

Nous passons pour cela par la boite à outils MUSIC21 [4], qui assure la lecture et l'analyse (*parsing*)

du fichier XML et la conversion sous forme de séquences d'événements. Ces événements contiennent en particulier des informations sur les début et fin de chaque ligature, tuple, *etc.* Des fonctions Python ont été écrites pour parcourir ces séquences d'événements de gauche-à-droite et construire les arbres de ligature et de tuple à l'aide d'une pile.

### 4.2. Évaluation de la qualité des données

Dans le cadre du projet GioQoSo portant sur l'étude de l'évaluation de la qualité des données dans les partitions numériques, nous avons proposé un outil qui réunit dans une interface unifiée différentes procédures d'évaluation qualitative de partitions musicales [1] , voir l'illustration sur la figure 7.

Ces évaluations s'appliquent à toute partition dans un format XML (MusicXML ou MEI) accessible par une URL. Nous donnons ci-dessous quelques exemples de propriétés sur les données relatives à la notation rythmique, vérifiées grâce à la construction d'arbres de la section 3.

**Vérification des durées.** Le calcul présentée au paragraph 3.3 est utile pour la vérification de l'intégrité des données contenues dans une partition numérique, plus précisément pour vérifier que chaque durée symbolique enregistrée (en *quarter-length*, XML `<duration>`, cf. paragraphsec :elements) correspond bien à la durée définie par les types de notes, les nombres de crochets ou ligatures, voire les tuples dans lesquelles ces notes sont incluses (cf. paragraph 3.3).

Une incohérence entre durée enregistrée et sa représentation graphique peut être considérée comme une erreur critique sur une partition, particulièrement si cette valeur de durée est utilisée dans un traitement de la partition.

Une autre procédure de GioQoSo vérifie que chaque mesure a la durée totale attendue, en faisant la somme des valeurs de durées des éléments contenus dans la mesure. La cohérence entre valeurs et représentations des durées est donc importante dans ce cadre.

**Notation de tuple.** En fonction du contexte (signature rythmique), l'étiquetage des tuples (par un entier ou une paire d'entier) doit être écrit explicitement ou pas, suivant des règles précises. Il peut être vérifié directement sur les arbres de tuple, par des tests sur les étiquettes, (cf. description au paragraphe 3.2), que les notations de tuples écrites devaient bien être explicites, et que ces notations sont correctes.

**Conventions sur les ligatures.** Certaines règles de bonne pratique pour l'écriture des ligatures, moins critiques que les problèmes d'intégrité présentés plus haut, sont néanmoins d'une grande importance en ce qui concerne la lisibilité des partitions. Les groupements de notes définis par les ligatures permettent en effet de donner au lecteur des informations visuelles immédiates sur les appuis métriques. Ces groupements doivent donc être cohérents avec la métrique.

Par exemple, il est convenu (voir [7] page 155) que certaines positions de la mesure (temps forts) ne doivent pas être croisées par une ligature car elles correspondent à des temps forts de la mesure (comme le troisième temps d'une mesure à 4/4).

Une telle propriété peut être vérifiée à l'aide de l'estimation des durées (paragraphe 3.3) et de la notion de nombre de ligatures entre notes, que nous avons présentée au paragraphe 3.1. Dans ce cas, le nombre de ligatures entre les deux feuilles situées de part et d'autre d'un temps fort à ne pas croiser devra être 0.

Il est également admis que les grands groupes de notes courtes (sous la double-croche, donc notés avec de multiples ligatures) sont plus faciles à lire lorsqu'ils sont divisés en sous-groupes dont la durée dépend de la métrique. D'après [7], le principe à suivre est que le nombre de ligatures séparant deux sous-groupes doit être égal à la durée des groupes qu'elles séparent. Cette recommandation est illustrée en figure 8 et peut aussi être vérifiée grâce à la propriété des arbres de ligature mentionnée au paragraphe 3.1.

Un problème détecté concernant les règles ci-dessus peut être signalé comme recommandation, il n'est pas du même niveau de criticité que les éventuelles incohérences sur les durées vues plus haut. Les fonctionnalités présentées plus haut sont été intégrées dans la bibliothèque Neuma, dans le cadre du projet GioQoSo.

### 4.3. Comparaison de partitions

La question d'une opération de recherche de *différences* dans des partitions numériques a été étudiée dans le cadre de l'édition collaborative en ligne de partitions [2]. Constatant d'une part que la comparaison manuelle est particulièrement fastidieuse, et que lancer la commande Unix diff sur les fichiers (texte) XML contenant les partitions ne donne pas de résultat exploitable, [2] propose d'appliquer sur des représentations hiérarchiques de partitions des algorithmes de calcul de distance d'édition sur les arbres [15, 12].

La représentation de la notation rythmique proposée ici pourrait être utilisée pour détecter des différences visuelles [2] entre partitions à un fin niveau de granularité, par simple comparaison (isomorphisme) d'arbres. Une représentation fidèle dans les arbres d'éléments graphiques de notation rythmique est avantageuse dans ce cadre, car elle assure une correspondance entre le résultat de la comparaison d'arbres et d'une comparaison visuelle de partitions. En d'autres termes, cette représentation des données rythmiques pourrait aider à l'automatisation de la tâche

1 . `http://neuma.huma-num.fr/quality`

2 . Notons que des notations sémantiquement équivalentes en terme de durées peuvent être différentes visuellement, cf. figure 3.

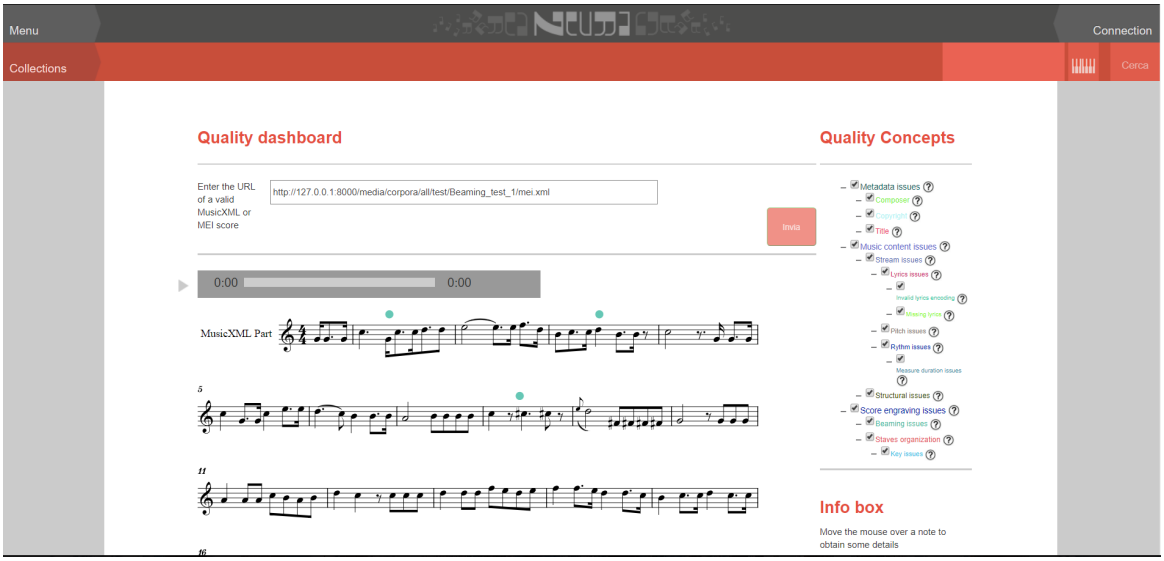



**Figure 7** : Une capture d'écran de l'interface de GioQoSo, intégré à la plateforme Neuma. Des problèmes de notation rythmique sont indiqués par des points verts.

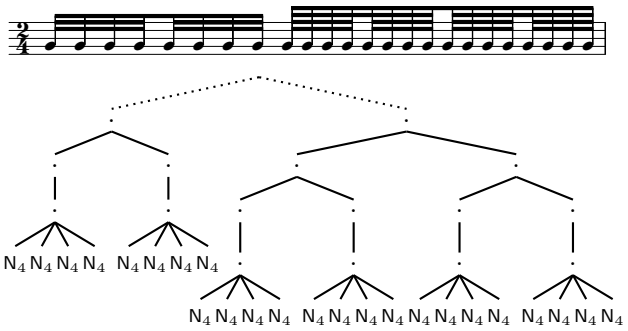



**Figure 8** : Divisions dans les ligatures et arbres de ligatures correspondant. D'après [7] : *"le nombre de ligatures séparant deux sous-groupes doit être proportionnel à la durée des groupes qu'elles séparent"*.

laborieuse de comparaison de différentes versions de partitions.

En prévision d'un tel usage, et afin de proposer un retour précis à l'utilisateur sur les différences observées, les identifiants des éléments XML sont stockés dans les sommets des arbres de ligature et tuples (ce qui est possible avec MEI), dans la procédure d'extraction décrite dans la section 4.1.

## 5. APPLICATIONS EN MODÈLE GÉNÉRATIF (PERSPECTIVES)

Il est possible par un parcours d'arbres de ligature et de tuple d'extraire les éléments XML (`<beam>` et `<tuple>`) produisant les notations correspondantes. Nous discutons informellement dans cette section de plusieurs usages possibles (actuellement en cours de développement) des modèles présentés ci-dessus en modèle génératif, dans le cadre d'applications produisant des partitions numériques, ce qui correspond à la flèche 2 de la figure 2.

L'avantage de l'utilisation de représentations intermédiaires sous forme d'arbres comme ci-dessus est de pouvoir s'appuyer sur des procédures bien établies, par exemple de *parsing* ou de *transformations* d'arbres, qui s'intègrent aisément dans un processus complet de production de partitions.

**Procédure générique.** Plus précisément, considérons le scénario suivant dans lequel les arbres de ligature et de tuples apparaissent comme représentation intermédiaire.

On suppose qu'à un certain stade de production d'une partition, nous disposons d'une suite d'événements musicaux dont les durées sont exprimées en fraction de noire (*quarter-length*). Cette séquence de durées peut être prise en entrée par une procédure de *parsing* suivant une *grammaire formelle* décrivant différents choix possibles de subdivisions successives des durées. Une telle procédure est présentée dans [8]. Les arbres de syntaxes produits sont appelés *arbres de divisions* car ils décrivent les divisions choisies. Ils sont très similaires aux arbres de rythmes OpenMusic [1].

La figure 9 présente trois arbres de divisions possibles pour la suite de durées initiale $\frac{1}{2}\ \frac{1}{6}\ \frac{1}{3}$. Dans le premier des 3 arbres, on a une division initiale du temps en 3 parties égales. Le premier fils, feuille étiquetée par $\bullet$, correspond à une note de durée $\frac{1}{3}$. Le deuxième fils est à nouveau divisé en deux parties égales de durée $\frac{1}{6}$. Le premier de ses deux petits-fils est étiqueté par $-$, ce qui signifie qu'il s'agit d'une continuation de l'événement précédent (ce qui peut être représenté par une liaison ou aussi ici un point). Le premier événement représenté aura donc une durée de $\frac{1}{3} + \frac{1}{6} = \frac{1}{2}$. Le second petit-fils est étiqueté par $\bullet$ et correspond à une note de durée $\frac{1}{6}$ (deuxième événement en entrée). Enfin, le troisième fils est éti-

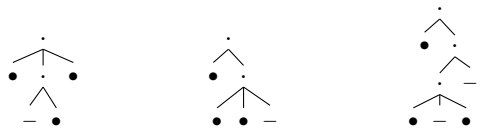


**Figure 9** : Trois arbres de divisions possibles pour la suite de durées $\frac{1}{2}\ \frac{1}{6}\ \frac{1}{3}$.

queté par • et correspond donc à une note de durée $\frac{1}{3}$ (troisième événement en entrée).

Le premier arbre de division de la figure 9 correspond aux arbres de ligature et de tuple des cas (a) et (b) de la figure 3, à partir desquels sera généré le code XML donnant les notations (correspondantes). Le deuxième arbre de division de la figure 9 correspond aux cas (c) et (d) de la figure 3, et le troisième arbre au cas (e).

**Gravure.** Des langages textuels comme Lilypond ou Guido permettent d'écrire de manière symbolique des suites d'événements musicaux avec des durées. La procédure décrite sommairement ci-dessus pourrait donc être utile dans le cadre d'une conversion de code dans ces formats textuels vers des partitions XML.

Des informations sur les tuples peuvent être contenues dans le code Lilypond ou Guido, qui devront être prise en compte pour la création des arbres de division. Une approche pour traiter ces contraintes est de les utiliser pour modifier la grammaire formelle utilisée dans le parsing.

**Transcription.** La procédure générique ci-dessus, qui utilise les arbres de ligature et tuples, pourra aussi être utilisée comme sous-tâche en aval d'une procédure de transcription par étapes.

Supposons que les premières étapes de transcription permettent d'obtenir des suites d'événements aux durées arbitraires (*e.g.* provenant d'un enregistrement MIDI de performance, ou de descripteurs extraits d'un enregistrement audio). Une étape de quantification rythmique retournera une suite d'événements aux durées quantifiées (en *quarter-length* comme ci-dessus) ou bien directement des structures semblables aux arbres de divisions ci-dessus dans le cas d'une procédure comme [14]. Nous pouvons alors procéder comme ci-dessus pour la production d'une partitions XML à partir de ces données.

## 6. CONCLUSION

Nous avons proposé et développé un modèle abstrait capable de représenter l'information relative au rythme d'une partition musicale. Il repose sur une structure de données unique, les arbres (au sens de la théorie des graphes), utilisée pour produire des *arbres de ligature* et des *arbres de tuple*, à partir d'encodages XML de partitions. Ce modèle a été mis à profit pour valider ces encodages, au cœur de la plateforme Neuma. Nous avons également discuté les possibilités (en cours de développement) de génération de notation à partir d'arbres existants, pour des tâches liées au rendu pour langages musicaux textuels ou la transcription.

Cette première avancée ouvre des perspectives intéressantes de travail et de collaboration avec la communauté musicologique. Tout d'abord, il serait sans doute utile de proposer une visualisation des arbres créés, dans Neuma mais aussi peut-être par l'intermédiaire d'un outil externe. Cela favoriserait l'adoption de ce modèle, pour les analystes comme pour les développeurs d'encodages.

Des extensions théoriques de ce travail de modélisation sont envisageables, pour valider par exemple d'autres aspects d'une partition que le seul rythme (accords, texte...). Le modèle pourrait également être étendu pour représenter l'ensemble d'une partition, avec des conteneurs pour voix, portées et systèmes.

Enfin, nous poursuivons l'objectif d'élaborer un outil de comparaison de partitions, dans lequel s'inscrit l'effort de modélisation présenté ici. Cela nécessitera d'une part l'élaboration d'algorithmes de comparaisons d'arbres appropriés et d'autre part le développement d'un retour visuel sur les partitions.

## 7. RÉFÉRENCES


[1] C. Agon, K. Haddad, and G. Assayag. Representation and rendering of rhythm structures. In *Proceedings Second International Conference on WEB Delivering of Music (CW'02)*, pages 109–113, 2002. IEEE Computer Society.

[2] C. Antila, J. Treviño, and G. Weaver. A hierarchic diff algorithm for collaborative music document editing. In *Third International Conference on Technologies for Music Notation and Representation (TENOR)*, 2017.

[3] J. Bresson, C. Agon, and G. Assayag. Openmusic : visual programming environment for music composition, analysis and research. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 743–746. ACM, 2011.

[4] M. S. Cuthbert and C. Ariza. Music21 : A Toolkit for Computer-Aided Musicology and Symbolic Music Data. pages 637–642, 2010.

[5] A. L. Danhauser. *Théorie de la musique*. H. Lemoine, 1872.

[6] M. Good. *MusicXML for Notation and Analysis*, pages 113–124. W. B. Hewlett and E. Selfridge-Field, MIT Press, 2001.

[7] E. Gould. *Behind Bars*. Faber Music, 2011.

[8] F. Jacquemard, A. Ycart, and M. Sakai. Generating equivalent rhythmic notations based on rhythm tree languages. In *TENOR* 2017.

[9] M. Laurson. Patchwork : A visual programming language and some musical applications. Technical report, Sibelius Academy, Helsinki, 1996.

[10] Music Encoding Initiative. `http://music-encoding.org`, 2015.

[11] P. Nauert. A theory of complexity to constrain the approximation of arbitrary sequences of timepoints. *Perspectives of New Music*, 32(2) :226–263, 1994.

[12] M. Pawlik and N. Augsten. Tree edit distance : Robust and memory-efficient. *Information Systems*, 56 :157–173, 2016.

[13] P. Rolland. The Music Encoding Initiative (MEI). In *Proc. Intl. Conf. on Musical Applications Using XML*, pages 55–59, 2002.

[14] A. Ycart, F. Jacquemard, J. Bresson, and S. Staworko. A Supervised Approach for Rhythm Transcription Based on Tree Series Enumeration. In *Proceedings of the 42nd International Computer Music Conference (ICMC)*, 2016.

[15] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6) :1245–1262, 1989.

# THE STRUCTURAL FUNCTION OF MUSICAL TEXTURE: TOWARDS A COMPUTER-ASSISTED ANALYSIS OF ORCHESTRATION

*Didier Guigue*
Mus[3]–Musicologia, Sonologia
& Computação – University of Campinas
didierguigue@gmail.com

*Charles de Paiva Santana*
Núcleo Interdisciplinar de Comunicação
Sonora – University of Campinas
charles@nics.unicamp.br

**ABSTRACT**

Musical form and orchestration are closely related dimensions of the musical discourse. Despite the proposition of several practical tools for assisted-orchestration in the past few years, scarcely any computational methods for analyzing the role of orchestration in shaping musical form and its perception have been put forward. As an answer to that, in this presentation, we discuss a couple of *OpenMusic* functions we designed for estimating the textural complexity of a piece's successive formal segments and sections. The calculations behind the functions originate in an appropriation of the mathematical theory of integer partitions and Wallace Berry's ideas on musical texture. In the current stage, our estimation of textural complexity is carried out upon the composer's prescriptions found on the musical score. Nevertheless, the work we present here is part of an overall model that ultimately aims to analyze form and orchestration combining symbolical and signal data, that is, joining the musical score and its performance recordings.

## 1. INTRODUCTION

From the end of the 19th century onwards, western classical composers began to adopt increasingly sophisticated, systematic strategies of orchestration [1]. As part of that trend, they were apt to actively correlate the orchestration technique with the crucial effort of structuring musical segments into a coherent whole, that is, the musical form. Eventually, different approaches for analyzing the orchestration technique of various composers and periods would emerge. For instance, Walter Piston, in his 1955 classic book on orchestration, stresses, although briefly, the relevance of such an undertaking:

> The objective in analysis of orchestration is to discover how the orchestra is used as a medium to present musical thought. [...] It is a means of studying how instruments are combined to achieve a balance of sonority, unity and variety of tone color, clarity, brilliance, expressiveness, and other musical values. Ultimately, the analytical process shows the differences in orchestral style between various composers and periods. [11, p. 397].

Afterward, the literature witnessed several developments concerning the analysis of orchestration; from the empirical, meticulous classifications of Charles Koechlin [10], through the pioneering sound-color theory of the 1978 book *Sound Design* [5], to the liberating and encompassing work of *Instrumentation in der Musik des 20.Jahrhunderts: Akustik-Instrument* [8].

Nowadays, in the era of computer-assisted composition and computational musicology, several practical tools for assisted-orchestration have been proposed. Even so, scarcely any computational methods for analyzing the role of orchestration in shaping and perceiving musical form haven been put forward. As an answer to that, we have been developing a methodological model for investigating such structural functions of orchestration using quantitative and qualitative methods supported by symbolical (musical score) and signal (audio files) data.

In this presentation, we are going to introduce the overall model followed by the implementation of one of its main stages, namely *the qualitative evaluation of musical texture using symbolical data*. More specifically, we would like to share the following contributions:

- A formalism allowing the correspondence between the segmentation of a musical score and audio files of its performance (section 2).
- An adaptation of the theory of integer partitions, together with an interpretation of Wallace berry's ideas on texture, for evaluating texture complexity as a function of orchestration (section 3).
- A demonstration of the above concepts, involving symbolical data, implemented and running on the Openmusic[4] environment (section 4).

The Openmusic functions are available as part of the SOAL library. The presentation is concluded with a discussion on future developments and perspectives of the overall model.

[1] By orchestration, we mean the technique used to combine a large number of sonic resources.

Didier Guigue, Charles de Paiva Santana, *The structural function of musical texture : Towards a computer-assisted analysis of orchestration*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 97–103, 2018. 

## 2. THE OVERALL MODEL: FROM THE SYMBOLICAL TO THE MATERIALITY OF THE PERFORMANCE

In our model, the orchestration is regarded as a two-dimensional phenomenon. The first dimension is the normative level of the written code, that is, the composer's prescriptions as they appear on the musical score. The second one is the practical, acoustic level of its performance. In other words, it concerns the actual result of those abstract prescriptions which, in turn, can be examined through audio files (via signal processing).

By acknowledging both dimensions, we intend to keep away potential biases and to cover orchestral styles less reliant on the normative level.

The segmentation of the musical piece into instrumentally distinctive, successive segments, in both dimensions, is pivotal to the overall model. That means that musical score and audio file(s) are sectioned into corresponding subsections. The segmentation, however, is not carried out concurrently in both mediums. It is achieved by firstly examining the musical score and cataloging every individual component of the orchestral sonic palette. It means recognizing not only the instruments required to play the composition but also every mode of execution specified on the score, including extended techniques, as *pizzicato*, *flutter-tonguing*, *col legno*, *sul ponticello*, etc. We designate this catalog as being the *Index of Sonic Resources* (SRI), and it takes the form of a textual list.

By tracking every change in the combination of components of the index, we can segment the musical score into successive individual blocks. We call each of those blocks a *Local Sonic Setup* or LSS for short.

It is only after the determination of the musical score's LSS's that the respective audio file(s) of its performance(s) are fittingly sectioned. We identify each of these audio file segments as a *Local Audio Unit* or LAU for short [2] .

LSSs and LAUs can be examined using quantitative or qualitative methods. LAU's analysis can be addressed by several audio feature's extraction algorithms.

An LSS can be examined, quantitatively, by counting the number of sonic resources it employs. For this purpose, we conceived a measurement that we call as the *Weighted Number of sonic Resources* or WNR for short [3] Finally, an LSS can be examined, qualitatively, by a function we call as *Relative Voicing Complexity* (RVC) which we shall describe in this presentation. The analysis of LSS's and LAU's can be synthesized to obtain a measure of the orchestration's relative complexity. Figure 1 shows a flowchart outlining the overall model.

[2] We have been using AudioSculpt's *Positive Spectral Differencing* to achieve very satisfactory segmentations, which are then fine-tuned to match LSSs.

[3] WNR workings should be detailed in a future paper.

## 3. THE QUALITATIVE EVALUATION OF MUSICAL TEXTURE

### 3.1. Partitional Analysis

We believe that the organization of sound resources into compound autonomous flows, that is, the orchestral texture, directly influences the dynamics of musical form. The texture is a dimension that characterizes a composer's style and reflects her particular way of negotiating with each instrument uniqueness and their role in stratified sound masses.

In *Structural Functions in Music* [3], Wallace Berry proposes a representation of musical texture based on the interdependency of the sound resources employed in a given musical structure. According to him, the musical texture consists of "real-components" – a part, or voice, that comprises one or more coordinated instruments – which are individualized in the polyphonic totality. Additionally, the sum of the sound resources used in a musical structure would also determine the musical texture.

This reasoning, as well as the numerical representation that it proposes, intersects with some fields of discrete mathematics, especially what is called as the theory of integer partitions[1, 2]. Its origins date back to Euler's work in 1748, and it was adapted to music theory by the Brazilian musicologist Gentil-Nunes [7, 6]. For instance, consider the number 5. It has 18 partitions, that is, 18 ways in which it can be represented as the sum of other integers. It means that if a composer wants to employ up to five sonic resources on a given musical passage, he or she can choose to lay them out into one of 18 combinations of groups and individual parts. Those combinations include all the partitions of the sub-groups as well, as the combinations of four, three and two simultaneous sound resources plus one resource alone. The partitions for a group of up to five sonic resources and its numerical representation are shown in table 1.

| | | | | | | |
|---|---|---|---|---|---|---|
| (5) | (4 1) | (3 2) | (3 1 1) | (2 2 1) | (2 1 1 1) | (1 1 1 1 1) |
| (4) | (3 1) | (2 2) | (2 1 1) | (1 1 1 1) | | |
| (3) | (2 1) | (1 1 1) | | | | |
| (2) | (1 1) | | | | | |
| (1) | | | | | | |

**Table 1**: All possible partitions of a *Local Sonic Setup* containing 5 (the sum) sound resources.

In our model, the partitions of a given group of sonic resources are represented by a list of integers enclosed in parenthesis. In this representation, the most extended list represents the most dispersed, viz. the most complex texture. For instance, consider the list $(1\ 1\ 1\ 1\ 1)$. It is five elements in length and describes a texture of five sonic resources playing independent parts. This configuration is more complex than, let's say, the configuration represented by the list $(2\ 1\ 1)$ whose length is 3 and refer to a texture composed by two resources playing coordinated parts and other two playing independent parts. Each element of such lists represents what is called a real component. The sim-

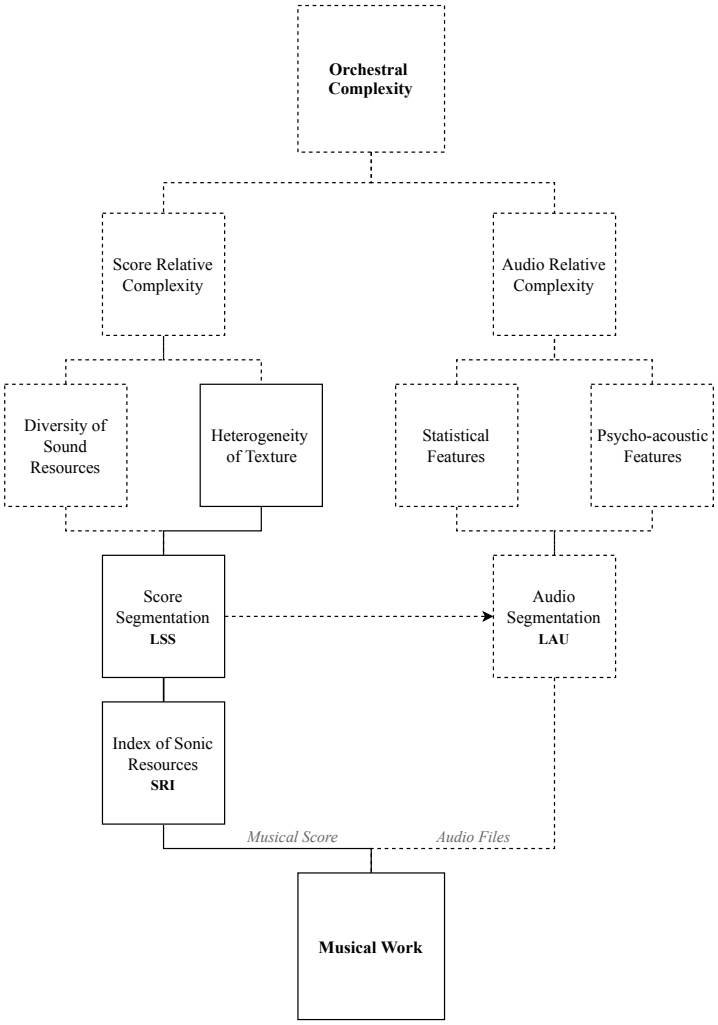



**Figure 1**: The overall model of our proposal for the computer-assisted analysis of orchestration. In this communication, we introduce the concept of an *Index of Sound Resources* which takes into consideration not only instruments but also their different modes of execution allowing the segmentation of the musical score into *Local Sonic Setups* (LSS). Each of these sonic configurations is analyzed by a qualitative method in terms of heterogeneity of the musical texture.

plest case for a texture of any number of sonic resources is the singleton list $(1)$ that represents a texture comprised of one single solo instrument. Thus, we say that the complexity of texture is a function of its rate of dispersion and the magnitude of its real components.

To calculate the rates of interdependence and independence, denominated bellow as the agglomeration and dispersion rates, respectively, of a given *Local Sonic Setup*, firstly we need to count every combination, or rather every possible relation of any two elements of the LSS. We can do it by referring to the general formula for finding the number of combinations of $p$ objects from a set of $n$ objects, known as $n$ choose $p$,

$$C(n,p) = \frac{n!}{p!(n-p)!} = \frac{n(n-1)(n-2)\dots(n-p+1)}{p!}. \tag{1}$$

For instance, in a setup composed of 4 sonic resources playing agglomerated parts, let's say a woodwind quartet, represented by the singleton list (4), there is a total of 6 unique pairs as $C(4,2) = 6$, that is,

(Fl Ob) (Fl Cl) (Fl Fg) (Ob Cl) (Ob Fg) (Cl Fg).

On the other hand, in an LSS of four soloists playing independent parts, that is $(1\ 1\ 1\ 1)$, when we consider any of its real-components or instruments, we find a total of zero unique pairs as $C(1,2) = 0$.

We will refer to the total number of unique pairs of any resource or real component of a given setup as $T_2$ or simply $T$. By reworking the equation given at 1, we will define it as a function in the following way:

$$\begin{aligned} T_2 &: \mathbb{N}^* \to \mathbb{N} \\ n &\mapsto \frac{n(n-1)}{2}. \end{aligned} \tag{2}$$

The successive total unique pairs $T_2(n)$ when n is mapped to the first eight positive integers, that is $1, 2 \dots 8$ is equal to $0, 1, 3, 6, 10, 15, 21, 28$.

It follows that, in order to calculate the rate of interdependence, or agglomeration, of a given LSS, we need

to sum the $T_2$ value of each of its components. For instance, the rate of agglomeration of the setup represented by the list $(2\ 1\ 1)$ is given by $(T_2(2) + T_2(1) + T_2(1))$ which results in 1. It is formally defined by the following summation function

$$\begin{aligned} \mathcal{A}\colon \mathbb{N}^r &\to \mathbb{N} \\ (a_0 \dots\ a_{r-1}) &\mapsto \sum_{i=0}^{r-1} T_2(a_i), \end{aligned} \tag{3}$$

where the list $(a_0 \dots\ a_{r-1})$ represents an LSS, $a_i$ each of its elements, that is, its real-components, and $r$ the length of the LSS.

We denote the dispersion rate of a given LSS as the difference between $T_2$ value of its sum by its agglomeration rate:

$$\begin{aligned} \mathcal{D}\colon \mathbb{N}^r &\to \mathbb{N} \\ (a_0 \dots\ a_{r-1}) &\mapsto T_2(\rho) - \mathcal{A}(a_0\ \dots\ a_{r-1}), \end{aligned} \tag{4}$$

where $\rho$ is the sum, the number of sonic resources of the LSS, that is, $\sum_{i=0}^{r-1} a_i$.

Both functions can be used to form a pair of indices from which graphs can be built and used to visualize the textural dynamics through time. In addition, the rates of dispersion and agglomeration can be combined and then visualized in one single dimension, symbolizing the global tendency of textural qualities, denoted by

$$\mathcal{I}(a_0 \dots a_{r-1}) = (\mathcal{D} - \mathcal{A})(a_0 \dots a_{r-1}). \tag{5}$$

In figure 2, the top chart shows the dispersion and agglomeration rates for a section (bars 56–73) of Anton Webern's *Variations for Orchestra, Op. 30*, while the bottom chart displays the overall trend in texture complexity (eq. 5).

### 3.2. Relative Voicing Complexity

To correspond the texture of an LSS with the piece's global sonic design, we propose to calculate "scaled" values for $\mathcal{I}$, where the lower boundary, 0, stands for the less complex texture and the upper limit, 1, the most complex. For now, the solution we are working with is to divide the $\mathcal{I}$ value by the $T_2$ value of the greatest LSS sum [4], i.e. the number of sound resources of the most dense LSS of the work, symbolized below by $\rho_{\max}$. We denote the ensuing quotient as the *Relative Voicing Complexity* or RVC for short,

$$\mathrm{RVC}(a_0 \dots a_{r-1}) = \frac{\mathcal{I}(a_0 \dots a_{r-1})}{T_2(\rho_{\max})}. \tag{6}$$

The RVC value is scaled through the application of $\frac{(\cdot)-\min\lambda}{\max\lambda}$, where $\lambda$ stands for $(a_0 \dots a_{r-1})$.

[4] Alternatively, the number of total resources used in the musical work, that is, the number of elements of the score's SRI, can be used.

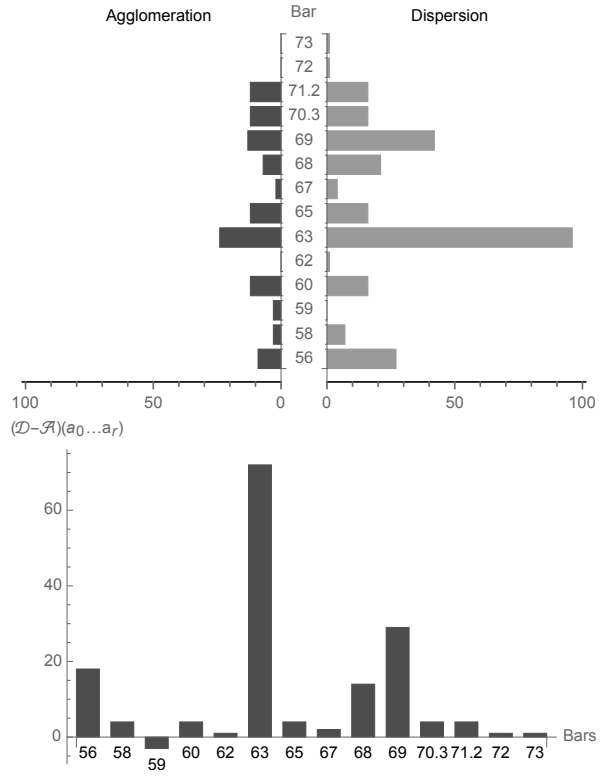



**Figure 2**: *Top*: agglomeration and dispersion rates for the bars 56 to 73 of Anton Webern's *Variations for Orchestra, Op. 30*. *Bottom*: Difference between dispersion and agglomeration rates for the same excerpt.

### 3.3. Criteria for the evaluation of dispersion rates

When we identify the local sonic setups, the most common criterion to designate a sound resource as part of a given real-component is the rhythmic coordination between resources. Such criterion follows Wallace Berry's correlation between rhythmic heterogeneity and textural complexity. But we preferred to leave to the choice of the musicologist the parameter or parameters he considers most appropriate, (generally according to the context and the work). We also give the possibility of placing them in hierarchical order by assigning them a more or less significant weight which would correspond to the impact that the parameter has on the dispersion.

This way, in our preliminary analysis of *Variations Op. 30*, besides rhythmic heterogeneity, we considered as a second heterogeneity parameter the tone-color heterogeneity, which is the textural organization into different flows of tonally homogeneous groups, like when we say *strings against brass*. In this particular analysis, tone-color heterogeneity was considered higher in the hierarchy and therefore weighted the calculation of an LSS's dispersion rate towards greater complexity.

There are other parameters that we have experimentally included in our analysis. They include heterophony, heterogeneity of articulation, the heterogeneity of intensity.

Each criterion is mapped to a different positive integer, following the order 1 = *strongest* to $n$ = *weakest*. The

RVC value can then be weighted by $(\cdot)\,[(\mathsf{c}_{\max} - \mathsf{c})s + 1]$, where $\mathsf{c}$ stands for the criterion of dispersion and $\mathsf{c}_{\max}$ for the number of criteria. The weighting is further controlled by a percentage, $s$.

In case of non-dispersion (when all the resources of a setup are aggregated) (no criteria apply), an arbitrary integer, greater than the one corresponding to the total number of criteria adopted, is inserted so that the algorithm understands that the setup has no dispersion.

### 3.4. Structural organization of orchestration's and texture's analytical data

Following the examination of the musical score, the information regarding the orchestration and textural complexity is organized in the form of a multidimensional table. In this format, the temporal succession of LSS's is represented horizontally and labeled, seen on the *header row*, by their starting points in the form *bar.beat*.

The table's *header column* refers to the piece's index of sonic resources. Each cell of the table indicates whether the specified sonic resource is employed in the respective LSS. The integer number which fills a cell designates how many simultaneous sounds were allocated for the specified active sound resource.

Other information about the successive LSS's is conveyed in additional rows at the bottom of the table. These include the total number of resources used in the LSS, the weighted number of sound resources, the rates of agglomeration, dispersion, their sum, and so forth. Further supplementary rows may be used to convey additional evaluations given by other functions of the SOAL library. The multidimensional table may be processed with the assistance of spreadsheet software.

## 4. OPENMUSIC IMPLEMENTATION

SOAL, the *Sonic Object Analysis Library* [9], is an OpenMusic external library that we continuously develop at the *Musicologia, Sonologia & Computação* group, the Mus3 [5] . It is conceived to be useful for a range of analytical purposes and supports a top-down approach.

SOAL's central concept is the *Compound Sonic Unit* or CSU. We define it as the combination and interaction of musical primary and secondary components: the former refers to collections of pitches and the latter to aspects as intensities, ranges, registers, densities and so forth. We also recognize as secondary components statistical measurements such as distributions, deviations, and entropy, among others.

SOAL is modular; new ad-hoc functions and components can be easily incorporated into the library. SOAL allows the inference of musical structures by comparing the relative sonic qualities of a sequence of CSU's and ultimately symbolizing them by a *vector of relative complexity*.

[5] The library can be downloaded at http://git.nics.unicamp.br/mus3-OM/soal4/tags

| | **56** | **58** | **59** | **60** | **62** | **63** | **65** |
|---|---|---|---|---|---|---|---|
| Fl | 1 | | | | | 1 | 1 |
| Fl *flatterzung* | | | | | | | |
| Ob | 1 | | | 1 | | 1 | 1 |
| Cl[B♭] | 1 | | | 1 | | 1 | 1 |
| Bcl | | | | 1 | | 1 | 1 |
| Hn[F] | | | | | | | |
| Hn[F] *sord.* | 1 | 1 | | 1 | | 1 | 1 |
| Tpt[C] | | | | | | | |
| Tpt[C] *sord.* | 1 | 1 | | 1 | | 1 | 1 |
| Tbn | | | | | | | |
| Tbn *sord.* | 1 | 1 | | 1 | | 1 | 1 |
| Tba | | | | | | | |
| Tba *sord.* | | | | 1 | | 1 | 1 |
| Cel | | | | | 4 | | |
| Hp | | 4 | | | 4 | | |
| Hp *harm.* | | | | | | | |
| Timp | | | | | | | |
| Timp trill | | | | | | | |
| Vn 1 *solo arco* | | 1 | | | | | |
| Vn 1 *solo pizz.* | | | | | | | |
| Vn 1 *solo pizz. sord.* | | | | | | | |
| Vn 1 *div. pizz.* | | | | | | 1 | |
| Vn 1 *tutti arco* | | | | | | | |
| Vn 1 *tutti sord.* | | | | | | | |
| Vn 1 *tutti harm.* | | | | | | | |
| Vn 1 *tutti pizz.* | | | | | | | |
| Vn 1 *tutti pizz. sord.* | | | | | | | |
| [etc.] | | | | | | | |
| Db *pizz.* | | | | | | | |
| Total resources | 9 | 5 | 3 | 8 | 2 | 16 | 8 |
| WNR | .67 | .49 | .34 | .64 | .21 | .85 | .64 |
| Agglomeration | 9 | 3 | 3 | 12 | 0 | 24 | 12 |
| Dispersion | 27 | 7 | 0 | 16 | 1 | 96 | 16 |
| Difference | 18 | 4 | -3 | 4 | 1 | 72 | 4 |
| Setup Complexity | .9 | .52 | .33 | .69 | .21 | 1 | .69 |

**Table 2**: Structured data from orchestration and textural analysis of Anton Webern's *Variations for Orchestra, Op. 30. Header column* refers to the SRI of the work, while the *header row* indicates the bar numbers.

The analysis of the *Relative Voicing Complexity* is, thus, one more option in the collection of functions for top-down music analysis. Indeed, an LSS is a CSU described by its sonic or instrumental configuration.

As with every other OpenMusic library, SOAL's functions are organized by purpose in different folders. SOAL's folder partitional analysis includes the function soal-texture-complexity which proposes an assisted-analysis of musical texture according to the premises exposed earlier in this presentation.

Figure 3 shows an OpenMusic patch as typically programmed for use with `soal-texture-complexity`. The list connected to the leftmost input of the function refers to the sequence of LSS's in the form ((`partition`) (`criterion`)). The list of bar numbers is inserted in the 5th input (from left to right). Other arguments include the list of total sound resources per LSS and the total number of dispersion criteria (cf. section 3.3).

The figure also highlights some of the function's outputs

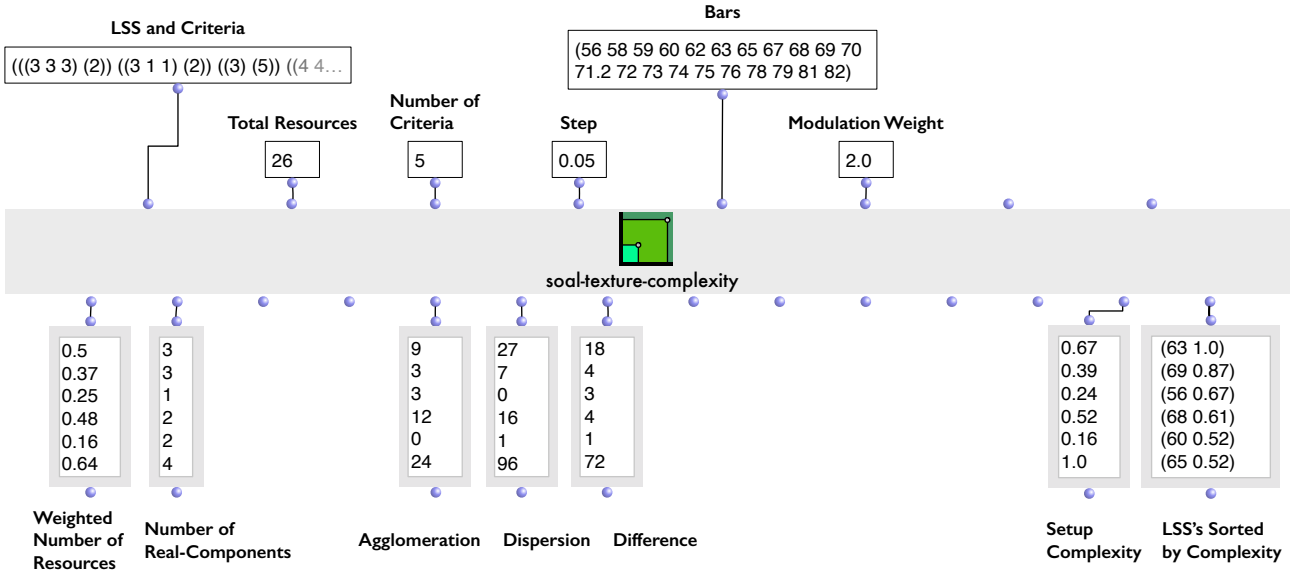



**Figure 3**: OpenMusic patch with the `soal-texture-complexity` function prototype in use to analyze bars 56–82 of Anton Webern's *Variations for Orchestra, Op. 30*.

such as:

1. the *weighted number of resources*, WNR, $\frac{\log \rho}{\log \rho_{\max}}$;
2. the number of real components, $r$;
3. the list of agglomeration rates, eq. 3;
4. the list of dispersion rates, eq. 4;
5. the list of differences between dispersion and agglomeration rates, eq.5;
6. the *relative setup complexity* [6], $\text{WNR}(\text{RVC}_w + 1)$;
7. The LSSs list, labeled by bar number, sorted from the most complex to the less complex.

Figure 4 shows an example of the kind of results our approach can produce. It depicts the analysis made on an excerpt from Beethoven's *5th symphony*, more precisely, the last bars of the 3rd movement together with the *attacca* which opens the finale. Even though not surprising, the analysis indicates that the finale's opening tutti - the place where all the expanded orchestra's resources are ultimately activated - correspond to the less complex texture of the excerpt, as practically all instruments play in homophony. The homophony explains why the last bar of the chart is so flat (cf. bar 242), even though the music reaches, at that moment, its most thrilling point, one could say.

## 5. CONCLUSIONS

Our experimental model for computer-assisted analysis proposes a formal strategy for evaluating the role of orchestration in musical structure and perception. It works by looking at the symbolic level of the score's prescriptions and the "perceptional level" of the performed music. At the symbolic level, the musical texture is considered as a qualitative aspect of orchestration, whereas the number of sonic resources involved in each local setup is the qualitative aspect. The analysis we put forward makes use of an application of the theory of partitions. We described how to collect and format (section 3.4) data from the score, its mathematical background (section 3.1), how the implemented function works (section 4), and the kind of results it may return (figures 2, 4 and table 2).

Nevertheless, there is still a great deal of research to do for the completion of the project as exposed in figure 1. For instance, the audio branch of our model (right side of figure 1) is at a very incipient stage: choosing, importing, developing and implementing appropriate descriptors is not a trivial task and demand further extensive research and

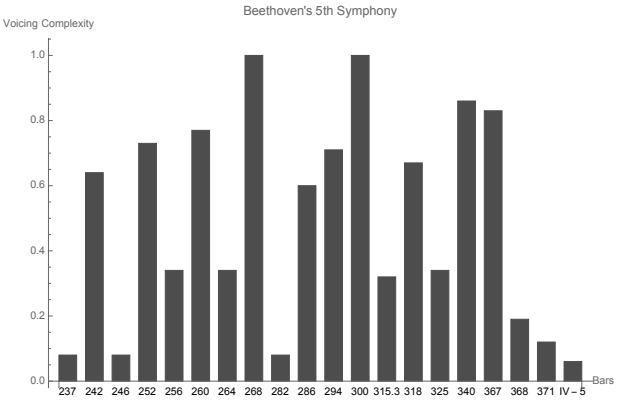



**Figure 4**: A sample of a scaled and weighted RVC analysis: Beethoven, *5th Symphony*, third movement, bars 237-371 plus first bars of *Finale*.

[6] The RVC weight onto WNR can be adjusted by the user

experiments. Furthermore, the impact of an LSS' duration upon the overall musical perception should be properly addressed in the next stages of our work.

Some aspects of orchestration are omitted in our model, such as the impact of polyphonic resources (e.g., a piano) playing agglomerated sounds (e.g., chords) in an overall dispersed texture. We are inclined to think such behavior works against the global sensation of dispersion, but we have no data, at the moment, that could help us to quantify to which extent this occurs.

In a more technical note, soal-texture-complexity should be able to incorporate a (semi)-automated tool for analyzing and structuring data as presented in table 2. Pattern recognition of texture or instrumental configurations in a single work, in a composer's whole oeuvre, or for some historical period, could bring a relevant complementary analysis. Such devices would significantly reduce the fastidious manual labor involved in the process. In fine, soal-texture-complexity, and the library as a whole should be implemented as stand-alone software in future works.

In any case, based on our first experimental results, we are quite confident that this approach consists of a helpful contribution in the field of computer-assisted music analysis.

## 6. REFERENCES


[1] Andrews, G. *The theory of partitions.* Cambridge University Press, 1984.

[2] Andrews, G. & Eriksson, K. *Integer Partitions.* Cambridge University Press, 2004.

[3] Berry, W. *Structural functions in music*. Dover, New York, 1987.

[4] Bresson, J, Agon, C., Assayag, G. "OpenMusic: visual programming environment for music composition, analysis and research." In *Proceedings of the 19th ACM international conference on Multimedia*, pp. 743-746. Scottsdale, USA, 2011.

[5] Cogan, R. & Escot, P. *Sonic Design: The Nature Of Sound And Music.* Prentice-Hal, New Jersey, 1976.

[6] Gentil-Nunes, P. *Análise particional: uma mediação entre composição musical e a teoria das partições*. PhD thesis. Universidade Federal do Estado do Rio de Janeiro, 2009.

[7] Gentil-Nunes, P. & Carvalho, A. "Densidade e linearidade na configuração de texturas musicais". *Proceedings of IV Colóquio de Pesquisa do Programa de Pós-Graduação da Escola de Música da UFRJ*. Rio de Janeiro: UFRJ, 2003.

[8] Gieseler, W., Lombardi, L., Weyer, R. D. *Instrumentation in der Musik des 20. Jahrhunderts.* Moeck Verlag, Celle, 1985.

[9] Guigue, D. *Sonic Object Analysis Library – OpenMusic Tools For Analyzing Musical Objects Structure*. 2016. http://www.ccta.ufpb.br/Mus3.

[10] Koechlin, C. *Traité de l'Orchestration*. Max Eschig, Paris, 1954.

[11] Piston, W. *Orchestration*. Norton, New York, 1969 [1955].

# ENSEIGNER LE PATCHING DE MANIÈRE COLLECTIVE AVEC LE LOGICIEL COLLABORATIF *KIWI*

*Philippe Galleron*
CICM - EA1572 Musidanse
Université Paris 8
philippe.galleron@gmail.com

*Eric Maestri*
CICM - EA1572 Musidanse
Université Paris 8
eric.maestri@gmail.com

*Jean Millot*
CICM - EA1572 Musidanse
Université Paris 8
jean.millot7@gmail.com

*Alain Bonardi*
CICM - EA1572 Musidanse
Université Paris 8
alain.bonardi@gmail.com

*Eliott Paris*
CICM - EA1572 Musidanse
Université Paris 8
eliott.paris@gmail.com

**RÉSUMÉ**

Les pratiques musicales collaboratives fondées sur le traitement temps réel du son à des fins de composition musicale constituent un nouvel enjeu en matière de pédagogie musicale. Notre équipe du CICM [1] s'est engagée dans une recherche-action dans le cadre du projet ANR [2] Musicoll [3] pour essayer de mieux comprendre les enjeux des pratiques collaboratives dans le *patching* [4]. Cette réflexion a été menée autour du développement d'un logiciel de *patching* collaboratif appelé *Kiwi*. Ce logiciel est le produit de la réflexion sur la pratique du *patching* et est refaçonné par les retours d'expérience dans le cadre de la pratique artistique et pédagogique. Cet article resitue brièvement les phases de développement du projet et apporte un éclairage sur les acteurs de l'équipe de recherche et leur façon de collaborer. Il traite ensuite plus spécifiquement de l'orientation pédagogique retenue dans la conception du cours de *patching* avec le logiciel collaboratif *Kiwi*. Nous soulignons les aspects pédagogiques innovants et la possible reconfiguration de l'enseignement du *patching* avec les possibilités d'interactions de ce nouveau logiciel. Notre réflexion porte sur l'utilisation de cet environnement numérique dans le contexte d'un cours universitaire de Licence 2 (L2) donné en présentiel dans la mineure CAO [5] au sein du département Musique de l'Université Paris 8. Nous abordons les stratégies d'enseignement, les scénarios de pré-implémentation conçus par les auteurs du cours et leur mise en œuvre pendant le second semestre de l'année universitaire 2017-2018. Nous traiterons du *design* pédagogique pour refondre le cours de *patching*, l'impact attendu sur l'apprentissage et les scénarios pédagogiques mis en œuvre. Enfin nous aborderons l'analyse d'un cas réel constitué par une séance de test effectuée en décembre 2017 avec des étudiants de L3. Cette séance a permis d'éprouver la stabilité du logiciel en conditions réelles durant un cours et nous a permis de « patcher » de manière collaborative avec la classe.

## 1. INTRODUCTION

### 1.1. La raison de l'article

Cet article est un compte-rendu d'étape du processus d'implémentation dans un cours universitaire du nouveau logiciel de *patching* collaboratif nommé *Kiwi*. Il est rédigé selon une manière collaborative par les membres de l'équipe du CICM à l'image des méthodes de travail en cours au laboratoire.

Il s'est écoulé un demi-semestre entre la première version de cet article en décembre 2017 qui était une pré-implémentation du logiciel en classe et avril 2018 où nous en sommes à la 6e séance d'enseignement du *patching* avec *Kiwi*.

L'équipe se définissant volontiers elle-même comme une communauté de pratique (au sens de Wenger [12]) de la composition musicale constituée d'enseignants-chercheurs et de leurs étudiants. L'ensemble des membres de l'équipe sont des « patcheurs » utilisant les logiciels *Max* et/ou *Pure Data* et des compositeurs maîtrisant l'informatique musicale en temps réel. Quatre des auteurs ont enseigné ou enseignent l'introduction au *patching* sur ces logiciels.

Le professeur du cours, le responsable pédagogique, le *designer* pédagogique et les développeurs de *Kiwi* font un suivi hebdomadaire de l'enseignement du cours de *patching* sur *Kiwi*. Les séances débutent par un retour

[1] Centre de Recherche en Informatique et Création Musicale
[2] L'Agence Nationale de la Recherche
[3] Musicoll : MUSIque COLLaborative temps réel et nomade, projet ANR (2016-2018) associant le CICM et l'entreprise OHMFORCE.
[4] Programmation graphique musicale
[5] Composition Assistée par Ordinateur.

Philippe Galleron, Eric Maestri, Jean Millot, Alain Bonardi, Eliott Paris, *Enseigner le patching de manière collective avec le logiciel collaboratif Kiwi*
*(**Prix AFIM 2018**)*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 105–114, 2018. 

d'expérience mis en commun. Ces retours alimentent les évolutions du logiciel qui à leur tour influent sur la pédagogie (Figure 1).

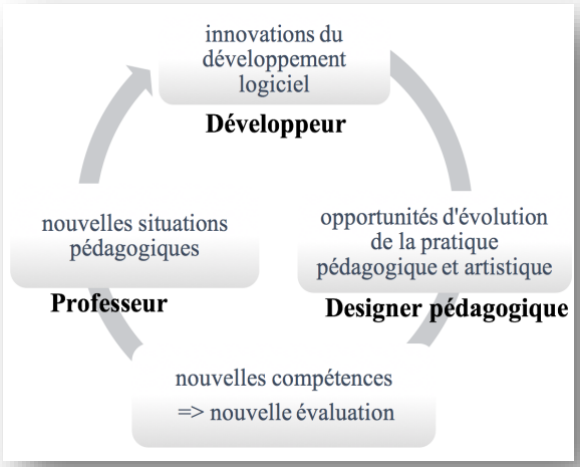



**Figure 1** : Cycle du développement technico-pédagogique du projet Musicoll.

### 1.2. Le *patching* collaboratif avec Kiwi

*Kiwi* est un logiciel de *patching* numérique dans la lignée de *Max* et *Pure Data*. Les spécifications techniques de ce logiciel du point du vue du développement ayant été publiées aux JIM 2017 [5], nous renvoyons le lecteur à ce texte.

Les programmes *Max* et *Pure Data* sont largement utilisés de nos jours et constituent une référence dans la pratique de la musique mixte ainsi que par ailleurs pour la pédagogie de l'informatique musicale. Cependant, ils ne sont pas conçus dans une optique de partage créatif et pédagogique. A l'opposé, le logiciel *Kiwi* que nous avons conçu au CICM dans le cadre du projet ANR Musicoll (2015-2018) développe cette pratique en permettant l'interaction entre les programmeurs sur le même *patch*.

L'élaboration du logiciel *Kiwi* est une réponse à un double constat, le premier porte sur les limites de la modalité unique d'enseignement du *patching* et le second est relatif à la pratique du *patching* qui est jusqu'à présent plutôt individuelle et solitaire derrière son propre ordinateur. C'est précisément sur cette tension entre un enseignement musical collectif et une pratique solitaire que le développement de ce logiciel vient agir.

L'enseignement des logiciels de *patching* est habituellement caractérisé par une approche frontale dans laquelle la relation entre l'enseignant et la classe est de type vertical : l'enseignant transmet son savoir et sa propre manière d'utiliser les logiciels, d'une manière hiérarchique descendante. D'ailleurs, l'expérience acquise dans l'enseignement en interne à Paris 8 des environnements de *patching* confirme cette perspective. En effet, la phase d'analyse [6] du cours existant a mis en évidence que lors des cours de *patching*, les professeurs, chacun avec leurs propres conceptions et leurs critères, restent fidèles à ce modèle de transmission du savoir [3]. Cette stratégie d'enseignement exploite les caractères des environnements existants, qui favorisent ce type de transmission à cause de leur structure, laquelle ne permet pas des modalités collaboratives. En effet, si la pratique de l'enseignement du *patching* à Paris 8 se fait en classe de manière collective, la pratique du *patching* ne l'est pas à proprement parler.

Par analogie, si nous devions le comparer à une pratique collective instrumentale, le geste de programmation n'est pas directement apparent, le son produit non plus car l'interaction visuelle et sonore n'est pas immédiate. Il faut donc un processus relativement long entre la programmation sur sa machine et l'obtention des premiers résultats sonores que l'on peut partager avec les autres. D'autre part, partager l'architecture d'un *patch* demande soit de transmettre le fichier d'une machine à l'autre, ce qui demande un temps d'analyse à celui qui le découvre *a posteriori*, soit de partager un visuel sur un écran, ce qui permet d'explorer dynamiquement les ressources du *patch* mais n'en donne qu'un aperçu. De fait, l'interaction tant visuelle que sonore est fortement différée ; on comprend aisément alors que ce type de pratique musicale ne se prête que peu au collectif.

Or, ce manque d'interactions dans les échanges se ressent également dans l'enseignement du *patching* et a des conséquences sur l'apprentissage. En effet, dans les sciences de l'éducation, les approches socio-constructivistes inspirées des travaux de Lev Vygostky [11] montrent l'importance des interactions dans le processus d'apprentissage qu'elles soient symétriques (entre étudiants) ou asymétriques c'est-à-dire de participant avancé à moins avancé (par exemple entre professeur et étudiants). C'est pourquoi le développement d'un logiciel collaboratif dans ce domaine apporte des changements dans la pratique car elle peut se réaliser collectivement. Les délais de partage sont réduits et offrent des opportunités nouvelles en termes d'enseignement, notamment une approche de la transmission plus horizontale qui nous conduit assez naturellement à une approche collaborative. Notre approche peut faire penser au domaine du *Computer supported collaborative learning* (CSCL) qui « est une branche des sciences de l'éducation qui s'intéresse à l'étude de la façon dont les gens peuvent apprendre ensemble avec l'aide des ordinateurs » [7] [8].

Par ailleurs notre équipe concilie des méthodes et points de vues pédagogiques complexes comme on le retrouve dans les CSCL qui considèrent des conceptions opposées de l'enseignement [8] comme [notre traduction] :

[6] Selon les phases du modèle d'*instructional development ADDIE* (analysis, design, development, implementation, evaluation).

[7] « branch of the learning sciences concerned with studying how people can learn together with the help of computers ».

– la métaphore de l'acquisition « dans laquelle l'apprentissage consiste en l'acquisition de connaissances stockées dans l'esprit » [8]
– la métaphore de la participation « dans laquelle l'apprentissage consiste en une participation croissante aux communautés de pratique » [9]
– la métaphore de la création des connaissances qui « sont créées […] grâce à la collaboration » [10].

Enfin notre méthodologie de recherche comme dans les CSCL combine des approches « de conception expérimentale, descriptive et itérative » [11] [8].

Toutefois nous devons bien signifier que *Kiwi* n'est pas à ce stade pensé comme un dispositif d'apprentissage en ligne, ni pour faciliter l'apprentissage de la musique à l'aide d'un ordinateur. Il est enseigné et pratiqué en présentiel. Il participe à l'apprentissage de l'audionumérique tout en apprenant le *patching* à des fins de création musicale. Pour cela il est conçu pour développer un enseignement et une pratique musicale collaborative.

Dans cet article, nous discuterons comment au CICM nous élaborons des stratégies d'enseignement de la programmation musicale suite à l'introduction du logiciel musical collaboratif *Kiwi* dans l'offre de formation en informatique et création musicale au sein du Département de musique de l'Université Paris 8. Nous traiterons particulièrement du *design* pédagogique et de l'impact de ce nouveau moyen technologique sur la refonte du cours « Introduction au *patching* avec *Max* et *Pure Data* ». Pour aborder ces sujets, le texte suivant s'articule en quatre parties. Premièrement, il présente de manière succincte le projet de réalisation du logiciel de *patching* collaboratif *Kiwi* mené dans le cadre du projet Musicoll (§ 2) ; deuxièmement, il discute le *design* pédagogique dans sa phase de pré-implémentation (§ 3-4) ; troisièmement, il propose un scénario-type ainsi que le déroulement d'une séance de cours (§ 5-6) ; enfin, les auteurs font un retour d'expérience d'une séance de test logiciel dans laquelle ils ont pu réaliser des *patches* en collaboratif pour la première fois avec les étudiants et évaluer la robustesse de *Kiwi* en situation réelle (§ 7).

## 2. LE PROJET MUSICOLL : *KIWI*

L'idée du développement d'un projet de recherche sur la programmation collaborative découle de l'observation des environnements de traitement du signal en temps réel existants, du constat de leurs limites en termes d'interactions collaboratives entre utilisateurs-programmeurs et de la pratique de *Max* et *Pure Data* dans le cadre de l'enseignement musical à l'Université Paris 8. A partir de l'analyse du cadre de ces pratiques, les chercheurs de l'équipe-projet Musicoll ont conçu un nouveau logiciel de *patching*[12] appelé *Kiwi*. Ces chercheurs souhaitent :

– développer un environnement numérique audio collaboratif pour le traitement du signal en temps réel ;
– étudier son utilisation par les musiciens-compositeurs ;
– renouveler l'enseignement des logiciels en temps réel en refondant le cours de *Max* et *Pure Data* dans le cadre de la mineure « Composition Assistée par Ordinateur » en Licence 2 au Département Musique de l'Université Paris 8.

*Kiwi* propose un environnement similaire à celui des programmes existants. La figure 1 exemplifie cet environnement dans laquelle, à la manière de *Max* et *Pure Data*, dans laquelle des modules, qu'on appelle « objets », sont connectés [13]. [adc~] donne accès au signal d'entrée audio, [delaysimple~] correspond à un retard avec réinjection du même signal, tandis que [dac~] est la sortie audio ; [*~] permet de contrôler l'amplitude du signal et les objets [*] mettent à l'échelle les valeurs des curseurs pour régler les paramètres de retard et réinjection correspondants (Figure 1). A la différence, dans *Kiwi* un tel *patch* peut être modifié par tous les utilisateurs y ayant accès.

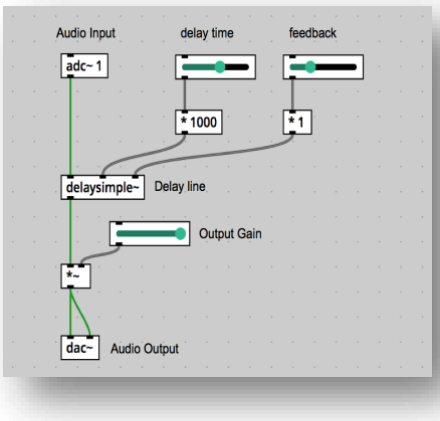



**Figure 2**. *Patch Kiwi* appliquant un retard avec réinjection et un gain sur un signal audio.

[8] « in which learning consists of individuals acquiring knowledge stored in their minds ».
[9] « in which learning consists of increasing participation in communities of practice ».
[10] « in which new knowledge objects or social practices are created […] through collaboration ».
[11] « Research methodology in CSCL is largely trichotomized between experimental, descriptive and iterative design approaches. Although sometimes combined within a single research project, the methodologies are even then typically kept separate in companion studies or separate analyses of a single study. Different researchers sometimes wear different hats on the same project, representing different research interests and methodologies ».
[12] Le terme *patching* décrit l'action de programmation de traitements sonores dans des environnements graphiques comme *Kiwi* ; de manière *très simplifiée il s'agit de connecter des boîtes représentant des* traitements avec des câbles virtuels.
[13] Par convention les objets sont écrits entre crochets ; le « ~ » signifie un objet qui traite un signal audionumérique par opposition à ceux qui traitent les données de contrôle.

## 3. CONCEPTUALISER L'ENSEIGNEMENT DU *PATCHING* COLLECTIF AU MOYEN D'UN NOUVEAU *DESIGN* PEDAGOGIQUE

L'un des principaux objectifs du projet Musicoll est de repenser et renouveler l'enseignement de l'interaction musicale en temps réel au service de la composition. Les enseignants et leurs étudiants sont associés dans la conception de nouvelles stratégies pour apprendre à « patcher » de manière collaborative.

L'esprit qui guide la conception pédagogique de ce cours se fonde sur l'acquis d'expérience suivant : la qualité et la rapidité de l'apprentissage semble être favorisée par le partage des travaux et des outils logiciels par la communauté des étudiants. En effet, nous pensons que les interactions dans la classe qui peuvent être symétriques – comme par exemple la co-résolution entre pairs – ou asymétriques – comme par exemple professeur-étudiants et expert-novices – « interviennent intrinsèquement dans la mise en œuvre des activités cognitives résolutoires et dans [la] genèse des processus intra-individuels de développement des compétences » [7]. La conception pédagogique que nous proposons est en cohérence avec ce propos. Il s'agit d'une approche de type socio-constructiviste considérant dans notre cadre d'enseignement que les interactions sociales autour de cette pratique musicale génèrent des conflits cognitifs qui sont une source d'apprentissage. Pour tirer parti de ce nouveau logiciel aux possibilités d'interactions sociales étendues, nous avons pour cela choisi de re-*designer* le cours visant tout d'abord à redéfinir ses objectifs, les outils et les scénarios pédagogiques pour l'enseignement de la programmation musicale temps réel.

## 4. PORTAGE DES OBJECTIFS PÉDAGOGIQUES DANS UN NOUVEL ENVIRONNEMENT

### 4.1. Anticipation des changements occasionnés

Un des objectifs que nous nous sommes fixés dans cette refonte est de renforcer au sein du cours le *savoir-apprendre* et le *savoir-travailler* de manière collaborative afin de montrer aux étudiants une manière alternative d'apprendre, de travailler et de créer. Une telle approche de l'apprentissage numérique collaboratif offre une contribution concrète à la transformation des pratiques pédagogiques [9]. Ces compétences sont considérées comme essentielles pour l'avenir du travail et de l'enseignement, comme le souligne l'organisation *Partnership for 21st Century Learning (P21)* [6]. Par ailleurs, l'actualité de cette approche est en phase avec des projets de création récents. Grâce aux développements des technologies numériques de partage des données, les pratiques musicales et artistiques évoluent fortement dans le sens du collaboratif en questionnant le statut traditionnel du compositeur, comme nous avons pu le constater directement [1] [14]. Grâce à des ajustements collaboratifs dans leur façon de « patcher », les étudiants devraient apprendre à définir des règles de partage et aussi de *workflow* commun tout en construisant leur environnement de programmation audionumérique et leur espace de travail pour l'intégrer de manière cohérente à leur propre pratique musicale.

A travers le *design* pédagogique de ce nouveau cours, nous essayons d'anticiper l'impact que pourrait avoir *Kiwi* sur le professeur et sa stratégie d'enseignement du *patching*. D'autre part, avant la véritable mise en place qui a eu lieu à partir du deuxième semestre de l'année universitaire 2017-18, nous essayons d'entrevoir les effets sur les étudiants de cette nouvelle forme d'interaction visuelle informelle entre les membres du groupe des étudiants qui n'existait pas dans le cours jusqu'à présent. En effet, l'enseignement du *patching* dans l'ancien cours était généralement composé de deux étapes. Tout d'abord, chaque étudiant préparait son *patch* en local sur son ordinateur puis le partageait sous la forme d'un fichier pouvant être dupliqué et ouvert sur d'autres machines. Chaque duplicata du *patch* étant une version indépendante dont les modifications ultérieures étaient propres à chaque étudiant. Cette collaboration asynchrone impliquait une phase solitaire de réalisation, puis des phases de partage et de retour d'expérience multiples.

Le logiciel *Kiwi* introduit au sein du cours permet quant à lui une collaboration synchrone et collective en programmation musicale. Cette pratique synchrone est fondée sur un document unique partagé et hébergé par un serveur, localement ou sur internet, accessible et éditable par chaque utilisateur du réseau sans aucune autorisation des autres utilisateurs. Les étudiants sont par conséquent invités dans cette nouvelle situation à assumer pleinement une position de connaissance partagée. Ils sont de fait dans une petite communauté musicale au sein de laquelle ils communiquent des stratégies de partage d'objets numériques et renforcent ainsi l'apprentissage réciproque dans et à travers leur utilisation. Par conséquent, la migration d'un mode de programmation musicale essentiellement solitaire à un autre collectif, dès la création du *patch*, implique des changements qui ont des répercussions sur la pédagogie de la création musicale. La question est alors d'enseigner et de se « former à et par la collaboration numérique » [2].

Dans le tableau (Tableau 1), nous soulignons les transpositions qui vont s'opérer entre notre expérience pédagogique de l'ancien cours et la projection que nous en avions. Puisqu'il s'agit d'une refonte de l'ancien cours de *Max* et *Pure Data*, les professeurs chargés de ce cours s'appuient sur leur savoir-faire existant en matière d'enseignement du *patching* afin de le faire évoluer. Nous souhaitons donc que les étudiants relient le *patching* aux questions de composition musicale et en outre développent de manière accrue leurs compétences

[14] L'un des auteurs du présent article, Eric Maestri, a pu expérimenter un projet de composition collective avec le collectif de compositeurs italiens */nu/thing*, dont une grande partie s'est déroulée à distance grâce à ces outils.

en termes de collaboration : le partage implique des normes de politesse dans la construction du document, des contacts fréquents, de la motivation personnelle et de l'auto-apprentissage.

| **Programmer avec *Max* ou *Pd*** | **Programmer avec *Kiwi*** |
|---|---|
| Espace de travail numérique individuel | Espace de travail numérique partagé |
| Partage asynchrone (par exemple par mail ou sur un *cloud*) | Partage synchrone sur le même document de travail |
| Mode collaboratif en présentiel | Mode collaboratif en présentiel dans la classe et/ou en ligne |
| Un espace de travail visible uniquement à proximité du poste de travail | Un espace de travail visible à distance |
| Travail sur le même *patch* limité à 3-4 étudiants (derrière le même ordinateur) | Travail sur le même *patch* (théoriquement) illimité |

**Tableau 1** : Modification de l'environnement d'apprentissage des participants du cours de *patching*.

### 4.2. Évaluation du dispositif

#### *4.2.1. Évaluation des étudiants*

Pour évaluer les étudiants, nous procédons à trois évaluations individuelles et sommatives par questionnaires sur le *Moodle* du cours. La première précède le cours et nous permet d'évaluer le niveau de connaissances des étudiants en audionumérique. La seconde à mi-semestre et la troisième à la fin du semestre portent sur le fonctionnement des objets de *Kiwi* et leurs connexions.

Par ailleurs nous mettons en place un travail artistique qui nous permet d'évaluer chaque groupe. Il s'agit d'un format de type « concert » dans lequel les groupes doivent présenter une petite pièce musicale mettant en œuvre un ou plusieurs procédés appris dans *Kiwi* (traitements, synthèse et transmission de données de contrôle). Les étudiants devront exprimer à l'oral ce qui est en œuvre dans leur patch. Ils devront à l'issue de cette démonstration fournir à la communauté leur patch accompagné d'un document explicitant sa prise en main et leur intention artistique et technique.

#### *4.2.2. Évaluation de l'impact de l'environnement collaboratif sur le cours*

Concernant l'évaluation scientifique des effets de l'introduction du collaboratif dans le cours de *patching* nous procédons en deux étapes :

- des enregistrements vidéo de la classe en activité et du professeur faisant un retour sur chaque séance.
- des entretiens filmés appelés d' « autoconfrontation » [15] et de « remise en situation par les traces matérielles »[16].

Nous nous inspirons de la méthodologie de recherche empirique dite « du cours d'action » [10]. C'est une manière de favoriser l'expression des acteurs sur leur sujet et ainsi enrichir l'analyse de leur activité. Nous pourrons par exemple déceler chez les participants des modifications de leur savoir-être ou préciser leur implication dans l'activité de groupe. Pour réunir des traces de leur activité les étudiants ayant donné leur accord sont filmés ; des captures vidéo de leur écran durant le cours sont collectées.

#### *4.2.3. L'influence du design et de ses limites*

Le *design* pédagogique tel que nous le pratiquons dans le projet Musicoll procède par allers-retours entre projection et réalisation. Les propositions d'innovations pédagogiques lors de la refonte d'un cours tant sur les méthodes que sur les contenus doivent être validées par l'ensemble des acteurs.

A la fin du semestre nous allons d'une part comparer ce qui était prescrit avec ce qui a été réalisé, d'autres part mesurer ce qui a été transmis. Pour cela nous procéderons

[15] Concrètement le participant est filmé en commentant les vidéos de lui-même en pleine activité.

[16] Le participant est amené à s'exprimer à partir des documents de travail qu'il a générés (p. ex. *patchs*, pièces).

à l'analyse par la *Method for Analysing and Structuring Knowledge* (MASK) [17] du cours préparé par le professeur, ce qui nous permettra de dégager les *savoirs*, *savoir-faire* et *savoir-être* en jeu dans la mise en activité des étudiants.

Puis nous comparerons cette somme de connaissances supposées avec les résultats des évaluations des étudiants et l'analyse des entretiens d'autoconfrontation.

Enfin nous comparons ces résultats avec ceux obtenus l'année précédente dans le cours « Introduction à la programmation avec *Max* et *Pure Data* ».

Ces données d'analyse nous permettrons à terme de dresser une évaluation qualitative sur les compétences développées par les étudiants en terme *patching* et de collaboration.

## 5. RÉSULTAT DE L'ANALYSE DU CONTEXTE D'ENSEIGNEMENT : *UN PREMIER SCÉNARIO-TYPE*

Les cours de *patching* du second semestre 2018 sont conçus comme des séances d'apprentissage en présentiel, même si, à court terme, nous envisageons un modèle d'apprentissage mixte, combinant l'apprentissage en présentiel et à distance sur la plateforme pédagogique *Moodle* de l'Université Paris 8. Le scénario exposé ici est essentiellement pour un cours en présentiel, mais nous incitons les étudiants à ce que ces situations d'apprentissage avec *Kiwi* prennent appui sur leur propre pratique de création musicale en dehors du cours. Ce travail de fond sera valorisé en classe et constituera l'un des supports de cours utilisé par le professeur.

C'est pourquoi nous suggérons donc aux étudiants de pratiquer *Kiwi* afin de :

- programmer régulièrement des traitements à distance sur un *patch* partagé connecté au serveur internet de l'application (ce travail doit se faire au minimum en binôme),
- de faire appel en cas de blocage sur le *patch* à l'aide d'un tiers (l'enseignant ou un étudiant plus avancé) pour déboguer leur *patch*,
- de documentariser [18] [4] leur *patch*, c'est-à-dire d'y ajouter des métadonnées, par exemple écrire un chapeau sur l'objet de leur *patch* ou constituer un petit document d'aide à sa prise en main.

## 6. DÉROULEMENT D'UNE SESSION-TYPE DE TRAVAIL COLLABORATIF SUR UN *PATCH* ORIGINAL PRODUIT PAR UN ÉTUDIANT

La session commence avec la correction des devoirs des étudiants. A partir d'un travail réalisé par un groupe d'étudiants (par exemple une pièce pour instrument électronique avec traitements en temps réel dans *Kiwi*), le professeur propose une activité collaborative autour d'un problème technique ou musical, identifié comme pertinent dans le *patch* présenté, par exemple l'utilisation d'un *delay* ou d'une enveloppe d'amplitude. Ensuite, le professeur invite le groupe d'étudiants à produire un *patch* en collaboration. Les bénéfices attendus sont les suivants :

- une autonomie accrue des étudiants dans la réalisation de leur *patch*,
- l'intégration des principes de la musique par ordinateur et la capacité de construire un discours autour de leur propre problématique musicale,
- être capable d'exprimer son approche musicale et compositionnelle avec un vocabulaire pertinent et spécifique au *patching* collaboratif,
- l'émergence, par cette situation collective, de l'émulation favorable également à l'apprentissage individuel.

Nous projetons un déroulement de la séance de la manière suivante :

1) Une introduction de la problématique du cours par l'enseignant depuis son bureau sur le tableau, en situation frontale vis-à-vis des étudiants. L'enseignant explique les bénéfices qu'il attend du cours et exprime explicitement qu'il est attentif à la formulation des idées sous-jacentes au *patch*.

2) Les étudiants procèdent à une présentation en groupe de leurs travaux de *patching*. Chaque groupe partage ses progrès, ses difficultés et ses projections pour améliorer son *patch*. Les autres étudiants peuvent les rejoindre sur leur document *Kiwi* pour suivre visuellement leurs explications. Ce moment est suivi d'échanges verbaux et d'interactions sur le logiciel entre les étudiants.

3) A partir de ce *patch* original présenté par leurs pairs, chaque groupe connecté au réseau *Kiwi* crée une copie du document partagé original (par exemple le groupe 1 crée le document 1 et le groupe 2, le document 2 *etc*.) et les étudiants s'approprient ainsi le *patch* du groupe à partir de leur propre ordinateur. L'enseignant lui se déplace de groupe en groupe et participe aux échanges verbaux et soutient la réflexion des élèves.

4) Chaque groupe propose une solution possible pour développer le *patch* et il s'ensuit un moment de discussion sur cette solution. Cela conduit à une co-assignation des tâches par groupe et au sein de chaque groupe.

5) Chaque groupe joue ses *patches* pour les expliquer. Pour cela le groupe présente son *patch* de manière orale et visuelle (*via Kiwi*) avec les autres groupes pour discuter de leur processus de *patching* et réaliser une auto-évaluation de celui-ci. Les solutions valides sont alors fusionnées dans le *patch* d'origine et/ou constitueront, pour la classe, un répertoire de pratique pouvant être mobilisé immédiatement au besoin à partir du serveur.

[17] La méthode MASK est une méthode de capitalisation, d'analyse et de structuration des connaissances.

[18] Documentariser selon Manuel Zacklad consiste à doter les documents « d'attributs spécifiques permettant de faciliter (i) leur gestion parmi d'autres supports, (ii) leur manipulation physique, condition d'une navigation sémantique à l'intérieur du contenu sémiotique et enfin, (iii) l'orientation des récepteurs » [13].

## 7. RAPPORT DU TEST DE TERRAIN EFFECTUÉ ET COMMENTAIRES

Ces perspectives de conception du cours et ces scénarios-types ont pu être partiellement mis à l'épreuve lors d'une séance de test en classe préparée en décembre 2017. Cette séance, prévue selon un protocole qui permettait de tester le logiciel avec un nombre élevé de connexions (jusqu'à 21 étudiants), a permis aux auteurs de se projeter dans une situation de cours réelle et de pouvoir apprécier la portée de leur conception pédagogique ainsi que de remarquer des aspects nouveaux qui seront intégrés lors de sa mise en œuvre effective. Dans une atmosphère de découverte, tant pour les étudiants que pour les professeurs, nous avons pu envisager des pistes d'interactions entre les membres de la classe et les enseignants. Les pédagogues et les développeurs ont alors conçu cette séance en trois étapes pour tenter de mettre à l'épreuve trois types de connexions de données sur un serveur local, distant et mobile. À partir de cette base ils ont pu valider la tenue du logiciel, des connexions de réseau, du serveur et interagir avec la classe en utilisant *Kiwi* pour la réalisation de trois *patches*. L'objectif principal de cette séance était clairement orienté vers un test technique car la séance était intitulée « crash test *Kiwi* ». D'autre part tester la prise en main de *Kiwi* par les étudiants n'a pas été proprement évoqué lors de cette séance et les critères n'ont pas été définis en équipe. Ce test technique consistait donc à évaluer :

1. la montée en charge de *Kiwi,*
2. le comportement de *Kiwi* sur les différentes configurations matérielles (systèmes d'exploitation des étudiants et type de machine),
3. les accès au réseau disponibles en séance (réseaux universitaires et connexions mobiles),
4. le travail sur *Kiwi* à partir d'un serveur distant puis à partir d'un serveur local déployé sur l'ordinateur d'un des encadrants du crash test.

Cependant tacitement chaque membre de l´équipe avait sa grille d'évaluation personnelle selon qu'il était le professeur du cours, le pédagogue et *designer* de ce nouveau cours ou l'ingénieur de recherche et développeur de *Kiwi*.

### 7.1. Point de vue du *designer* pédagogique

L'expérience de pédagogue nous a convaincus que les conditions de travail d'une classe sont déterminantes d'un point de vue pédagogique et influencent par conséquent l'apprentissage. A travers ce test technique le *designer* pédagogique s'est ainsi intéressé à l'analyse des supports du cours et de l'environnement de travail que nous mesurons selon les ressources matérielles à disposition et leur distribution entre les acteurs. Au-delà de ces considérations propres aux ressources matérielles nous étions particulièrement attentifs au confort des étudiants au sein de ce dispositif de cours et à travers l'utilisation du logiciel *Kiwi*.

#### *7.1.1. Conception et réalisation de la séance*

Nous avons cherché à déterminer si la salle de classe et les étudiants disposaient bien de l'équipement nécessaire pour développer l'axe collaboratif renforcé que nous nous sommes fixé dans la refonte du cours. Dans le cadre de cette séance de test il fallait donc évaluer :

1. la mise en réseau ou non des ordinateurs et la présence de disques durs partagés et de serveurs,
2. les systèmes d'exploitation des machines des étudiants (et leur mises à jour et autorisations),
3. les supports de diffusion de l'information (tableau, enceintes, projecteurs) pour diffuser le *patch Kiwi* et les clefs USB pour installer *Kiwi* sur les machines,
4. le mobilier de classe et sa modularité pour créer des îlots ou des pôles de travail (chaises et tables),
5. la connectivité des salles en termes de prises électriques, de connexions internet par *Ethernet*, de niveau et de stabilité du réseau *wi-fi*, de qualité de réception du réseau téléphonique et connexions internet mobiles, *etc*.,
6. le nombre d'ordinateurs et de connexions internet.

En séance, nous avons déroulé un protocole de test et nous avons réalisé à chaque étape un sondage à main levée auprès des étudiants après chaque mise en situation concrète de travail sur *Kiwi* et recueilli leurs commentaires oraux.

#### *7.2.2. Évaluation de l'environnement technique d'apprentissage de la classe*

Une fois réalisé ce premier inventaire sur les ressources existantes et utiles au travail collaboratif, nous avons interrogé les étudiants propriétaires de ce matériel sur leur disposition à le partager. En effet partager ce matériel (leur connexion internet mobile, par exemple) est lié assez naturellement avec la question de la maintenance. L'aspect collaboratif numérique implique en effet un matériel toujours en mesure de fonctionner pour assurer le travail qui en dépend. Il n'est pas envisageable d'avoir un problème de connexion à internet persistant ou récurrent que cela vienne du réseau ou des machines. Il faut que ce problème intervienne le moins possible et qu'il soit très rapidement résolu : un internet haut-débit *wi-fi* stable est donc nécessaire pour un usage intensif des ressources en ligne et il faut en l'occurrence avoir un serveur robuste. Si l'un ou l'autre faisait défaut il nous faudrait des solutions de secours notamment en s'appuyant sur le matériel des étudiants (connexion internet mobile, par exemple, ou constitution d'un serveur local sur leur machine). C'est pourquoi, par ailleurs, nous avons testé le serveur commun de *Kiwi* sur un internet distant et sur un réseau local.

Pour évaluer le confort des étudiants nous avons observé :

– leur comportement quand les premiers sons sont sortis de leur machine,
– les premiers commentaires écrits et oraux qu'ils ont partagés sur leur *patch* commun,
– la fluidité de leur geste sur l'interface graphique,
– leurs difficultés éventuelles d'installation et de connexion.

*7.2.3. Bilan*

L'analyse des supports et de l'environnement de travail en situation a mis en relief la question de la maintenance des connexions internet. Nous suggérons d'attribuer des responsabilités et d'en distribuer la compétence entre le service informatique de l'Université, les professeurs de la spécialité, les étudiants eux-mêmes à titre individuel ou de constituer des groupes d'étudiants compétents pour maintenir le serveur en cas de *crash* et supplanter une éventuelle défaillance de la connexion internet.

Par ailleurs, dans la mesure où nous souhaitons que les étudiants puissent travailler en groupe en dehors du lieu où se déroule le cours, notre test portant sur les connexions mobiles des étudiants et des professeurs en situation de cours a été éclairant. Ce test a montré qu'un téléphone mobile peut supporter environ 4 étudiants connectés sur le serveur. En revanche, le taux de *crash* est supérieur et la qualité des interactions visuelles est moins bonne qu'en utilisant les réseaux universitaires Eduroam ou Eduspot.

Les questions d'équipement en machine et en connexion internet étant encore fortement dépendantes des lieux, nous envisageons à terme de cartographier les espaces de travail des étudiants pour pratiquer le *patching* collaboratif hors-classe (salles informatiques, studios, bibliothèque universitaire, *etc.*) et nous nous demanderons s'ils sont adaptés à l'évolution du cours. Il sera intéressant de s'interroger également sur la mobilité de ce dispositif technique. Par exemple, déterminer si les apprenants peuvent être mobiles et se déplacer entre deux connexions pour pratiquer le *patching* collaboratif dans différents lieux ou s'ils sont au contraire contraints d'utiliser un certain type de matériel dans un lieu spécifique.

### 7.2. Point de vue de l'enseignant

*7.2.1. Conception et réalisation de la séance*

En ouverture de séance le professeur a rappelé la nature expérimentale du cours et demandé aux étudiants leur participation active en tant que testeurs d'un nouveau logiciel. Il a également proposé aux étudiants de concevoir cette séance comme une expérimentation rigoureuse faisant partie d'un programme de recherche sur la pédagogie de la programmation musicale. Cette introduction a permis aux participants de comprendre l'importance de leur implication dans ce retour d'expérience. Après avoir installé le programme et suivi le protocole de test du logiciel, le professeur a demandé aux étudiants de commencer à réaliser des *patches* collectifs. En premier le professeur a demandé à la classe de créer des objets *Kiwi* sur le *patcher*. Les participants ont pu apprécier les similitudes et les différences de la modalité de programmation *Kiwi* vis-à-vis de leur précédente expérience de *patching* avec *Max* et *Pure Data*. Les étudiants, divisés en groupes, ont été invités à réaliser un premier *patch* avec des objets simples permettant d'additionner des signaux. Les objets employés [osc~], [+~], [*~], [slider] et [dac~] ont pu être créés par tous les groupes et connectés de manière habituelle. Le professeur a pu constater l'exactitude des *patches* réalisés et interagir avec les étudiants.

Une fois tenue pour acquise la stabilité du logiciel et comprise la modalité de fonctionnement du cours collaboratif, le professeur a proposé aux étudiants de réaliser un *patch* plus complexe avec un nombre d'objets plus élevé. Il s'agissait de réaliser ensemble un *patch* qui permette de créer une modulation d'amplitude en utilisant des objets comme [line~] et [number]. Le professeur a également abordé l'implémentation d'une ligne à retard avec l'objet [delaysimple~]. La modulation de fréquence implémentée en fin de séance a permis d'introduire l'objet [sig~] et de créer des connexions et des contrôles sonores plus fins. De cette manière, les étudiants ont pu revisiter leur façon de programmer en expérimentant une interface alternative, qui ouvre des perspectives nouvelles.

*7.2.2. Évaluation technique et pédagogique*

Le professeur a pu constater les différences apportées par l'utilisation de *Kiwi*. La classe est en effet obligée de co-participer à la réalisation du *patch* à cause de la structure même du logiciel. Elle implique une connexion multiple au même *patch* : les étudiants partagent véritablement le même espace de rédaction du *patcher*. Ceci les oblige à développer des nouvelles compétences de travail collaboratif, comme par exemple des normes de politesse et des manières de communiquer et d'interagir détournées, comme par exemple l'utilisation de l'objet [comment] en tant que *chat*. Une telle interaction introduit une dimension ludique qui favorise les échanges et la bonne atmosphère au sein de la classe. Ces retombées et les comportements des étudiants n'ont cependant pas surpris les pédagogues car ils étaient pris en compte et prévus lors du *design* du cours.

*7.2.3. Bilan*

Le scénario prévu se révèle efficace pour prendre en main le logiciel. Le professeur a pu remarquer une participation plus efficace. En même temps, l'interaction avec la classe a fait en sorte que des innovations du nouveau scénario prévu aux points 3) et 4) émergent, notamment au niveau de la visualisation, en classe, des patches réalisés singulièrement par les groupes. En contrôlant qu'effectivement tous les groupes étaient capables d'utiliser le logiciel, certains *patches* des étudiants étaient partagés sur l'écran projeté au tableau,

par l'ordinateur du professeur. Cela a permis de travailler véritablement ensemble au niveau de la programmation et d'interagir avec toute la classe pour s'entraider sur la réalisation d'un *patch* d'un groupe particulier. Par cette situation le professeur pouvait solliciter la participation de la classe en posant des questions concernant la résolution du *patch*. Selon les auteurs, un tel aspect constitue une avancée pratique importante au niveau de l'enseignement car les étudiants en participant d'une manière plus active accroissent leur compréhension du logiciel et par conséquent de certains fondements de la musique électronique.

### 7.3. Point de vue du développeur

#### *7.3.1. Conception et réalisation de la séance*

Cette séance de test à l'université avait, du point de vue du développement et de la conception, deux objectifs principaux. Le premier était d'éprouver techniquement le logiciel dans des conditions réelles en termes de latence du réseau et de nombre d'utilisateurs connectés à une session. Le second était d'observer l'utilisation du logiciel *Kiwi* par les étudiants et de collecter leurs remarques afin de planifier de nouveaux développements à moyen et long terme.

#### *7.3.2. Évaluation technique et ergonomique*

La robustesse et le niveau de qualité d'un logiciel sont des notions évidemment primordiales de l'ingénierie logicielle. Cependant, dans le cadre d'un projet expérimental tel que *Kiwi*, l'accent est mis sur la conception de nouvelles *features* collaboratives innovantes obligeant parfois le développeur à faire des compromis. C'est pourquoi il était important de planifier une séance de test afin de relever les problèmes techniques et de pouvoir les corriger dans la phase finale précédent la sortie de la première version de *Kiwi*.

Comme évoqué précédemment, la qualité de l'apprentissage de l'audio numérique collaboratif est un point central dans le cadre de recherche du projet Musicoll. De plus, un de nos objectifs est de rendre accessible le *patching* audio collaboratif à une communauté large d'utilisateurs allant du novice souhaitant découvrir cette façon de composer à l'utilisateur expérimenté. C'est pourquoi l'ergonomie et l'expérience utilisateur doivent être particulièrement soignées. Or, il est parfois difficile pour les membres de l'équipe Musicoll, qui sont soit des utilisateurs aguerris d'outils de *patching* audio tel que *Max* ou *Pure Data* soit des personnes à l'aise avec l'utilisation de logiciels collaboratifs en réseau, de distinguer certaines difficultés d'utilisation.

#### *7.3.3. Bilan*

Du point de vue technique, cette expérience est une réussite car elle a permis d'observer des problèmes du logiciel difficiles à observer dans un contexte classique d'utilisation c'est-à-dire sans latence et avec seulement quelques utilisateurs. Ces problèmes seront bien entendu corrigés avant le lancement officiel du cours utilisant *Kiwi* comme support.

Nous avons pu constater grâce à ce test d'utilisation qu'une attention particulière devait être portée à l'ergonomie du logiciel liée à la connexion (notification de l'état connecté/non connecté, facilité du *login*).

La discussion avec les étudiants nous a également permis de distinguer des pistes de conception à plus long terme. Ainsi la question du jeu collaboratif, c'est-à-dire de la possibilité de pouvoir transmettre entre utilisateurs des paramètres de contrôle, a de nouveau été évoquée. Nous avons donc décidé de concrétiser le développement de cette fonctionnalité afin que les utilisateurs de *Kiwi* puissent en bénéficier dès la première version [19].

## 8. CONCLUSIONS

Le changement concret d'utilisation d'un environnement de programmation graphique implique une innovation dans les approches pédagogiques qui ont un impact sur la pratique musicale en elle-même et comportent des avantages en termes d'apprentissage. Du point de vue de la conception pédagogique du cours, l'approche liée à *Kiwi* nous enjoint de reconstruire le lien qui associe la question de l'enseignement du *patching* audionumérique aux problématiques de composition que rencontrent les étudiants en création musicale dans notre université. Selon nous le *patching* collaboratif apporte des innovations qui conduisent à :

- expérimenter l'intelligence collective dans ce type de pratique de programmation musicale ;
- penser les étudiants en terme de membres d'une communauté d'apprentissage ;
- repenser la façon d'évaluer les étudiants au sein d'un groupe ;
- commencer une réflexion / action au sein de l'équipe pédagogique sur l'apport de l'apprentissage en collaboratif numérique ;
- accepter de nous investir en tant qu'enseignants-chercheurs sur cette façon collaborative d'enseigner et de pratiquer la programmation musicale avec l'intuition qu'elle sera très développée à l'avenir.

Lors des tests effectués, nous pouvons avancer des considérations concernant ce paradigme collaboratif :

- nous avons remarqué une **implication différente** de la classe qui est amenée à dialoguer d'une manière profitable dans la réalisation des *patches* ;
- l'interaction entre les étudiants assume une **dimension ludique** qui crée une atmosphère

[19] L'objet [hub] a depuis été créé pour cela.

favorable. Le professeur a pu apprécier des aspects nouveaux au niveau de la conduite du cours ;

- le partage des différents *patches* des groupes permet d'impliquer les membres de la classe dans la résolution des questions liées à la programmation. Cet aspect a selon nous un impact remarquable sur la vitesse d'appréhension des concepts et des objets, leur profondeur et leur enracinement dans la pratique des étudiants.

Des aspects critiques accompagnent les éléments positifs de l'utilisation de *Kiwi* que nous venons d'évoquer :

- l'**instabilité des connexions** internet pourrait mettre en danger le déroulement des séances ;
- la mise en commun du support pourrait mener à des **déséquilibres** au niveau de la participation des étudiants. En effet le fait de travailler sur un même *patch* risque de favoriser les étudiants plus rapides et/ou aguerris. C'est pourquoi afin de remédier à cette éventualité, les enseignants ont demandé aux étudiants inscrits de remplir un questionnaire concernant leurs connaissances et pratiques audionumériques. Au cours du semestre ils veillent à la constitution de groupes d'environ 5-6 étudiants pour favoriser la proximité entre participants de niveau équilibré dans leur aptitudes (à modéliser, à mobiliser des connaissances, à expliquer leur démarches).

De fait, l'équipe pédagogique espère qu'à travers ce dispositif d'enseignement, les étudiants auront acquis une base de connaissances qui devraient leur permettre de terminer leur licence de manière efficace et d'envisager la poursuite de leurs études en ayant profité des innovations au niveau pédagogique que cette nouvelle logiciel propose. Pour mesurer le premier impact des scénarios pédagogiques envisagés, les enseignants préparent un bilan de fin de semestre en comparant les connaissances de la classe au début et à la fin du cours de *Kiwi*.

## 9. REMERCIEMENT

La majeure partie de la recherche exposée dans cet article est financée dans le cadre de la convention attributive d'aide de l'Agence Nationale de la Recherche n° ANR-15-CE38-0006-01.

## 10. RÉFERENCES

[1] A. Agostini, D. Ghisi, R. Grimaldi, E. Maestri, A. Sarto, (collectif */nu/thing*), *I mille fuochi dell'universo*, pour ensemble et électronique (2017). Commande du 26ème Festival Milano Musica et de la Fondation Siemens**.**

[2] C. De Lavergne, M-C. Heïd, « Former à et par la collaboration numérique », *tic&société* [on line], Vol. 7, n° 1, *online*. URL : http://ticetsociete.revues.org/1308; DOI: 10.4000/ticetsociete.1308.

[3] P. Galleron, « Développement d'une communauté de pratique de la composition musicale assistée par ordinateur en milieu scolaire : conception, parcours et modélisation », thèse de doctorat. de l'Université Paris 8, 2017, 724p.

[4] P. Galleron, A. Bonardi, M. Zacklad, « La documentarisation coopérative du processus de création musicale numérique dans un contexte pédagogique », *Actes des Journées d'Informatique Musicale 2015*, Montréal, Canada, <http://jim2015.oicrm.org/#actes>. <hal-01161641>.

[5] E. Paris, J. Millot, P. Guillot, A. Bonardi, A. Sedès, « Kiwi : vers un environnement de création musicale temps réel collaboratif premiers livrables du projet Musicoll », *Actes des Journées d'Informatique Musicale 2017*, Paris, 2017.

[6] *Partnership for 21st Century Learning*, http://www.p21.org/component/content/article/2224.

[7] J.P. Roux « Le travail en groupe à l'école » in Cahiers Pédagogiques, N°424, Mai 2004, page 2. http://www.cahiers-pedagogiques.com/IMG/pdf/Roux.pdf, consulté 22 janvier 2017.

[8] G. Stahl, T. Koschmann, D. Suthers, « Computer-supported collaborative learning: An historical perspective ». In R. K. Sawyer (Ed.), Cambridge handbook of the learning sciences. Cambridge, UK, 2006, pages 409-426, [Cambridge University Press].

[9] D. Tao, J. Zhang, D. Gao, "Reflective Structuration of Knowledge Building Practices in Grade 5 Science: A two years Design-Based Research", *CSCL 2017 Proceedings*, p. 644-647.

[10] J. Theureau, « Les entretiens d'autoconfrontation et de remise en situation par les traces matérielles et le programme de recherche « cours d'action », Revue d'anthropologie des connaissances, Vol 4, n° 2, 2010/2.

[11] L.S. Vygotsky, (1931/1983). История развития высших психических функций // Собр. соч.: В 6 т. М.: Педагогика. Т. 3, с. 5-328. [History of the Development of the Higher Mental Functions. In R.W. Rieber (Ed.), The Collected Works of Vygotsky, vol. 4 (1-251). (1997). New York: Plenum Press.

[12] É. Wenger, La théorie des communautés de pratique. Apprentissage, sens et identité, Québec. Presses de l'Université de Laval, 2005. [Communities of Practice: Learning, Meaning, and Identity, Cambridge University Press, 1998 traduit de l'original par Fernand Gervais.

[13] M. Zacklad, « Processus de documentarisation dans les Documents pour l'Action (DopA) : statut des annotations et technologies de la coopération associées ». *Le numérique : Impact sur le cycle de vie du document pour une analyse interdisciplinaire*, Montréal, Québec, 2005. Éditions de l'ENSSIB. http://archivesic.ccsd.cnrs.fr/sic_00001072/

# APPROPRIATING MUSIC COMPUTING PRACTICES THROUGH HUMAN-AI COLLABORATION

*Hugo Scurto*
Ircam - Centre Pompidou
STMS IRCAM–CNRS–SU
Hugo.Scurto@ircam.fr

*Frédéric Bevilacqua*
Ircam - Centre Pompidou
STMS IRCAM–CNRS–SU
Frederic.Bevilacqua@ircam.fr

**ABSTRACT**

This paper presents a prototypical computational framework for music computing appropriation. Putting Human-Computer Interaction at the center of the issue, it proposes to use Artificial Intelligence (AI) to assist humans in their first uses of music computing systems—namely, their exploration. We first review how interaction design may be central to exploration and appropriation of music computing systems, highlighting guidelines and potential directions to improve it. We then present our framework proposal, centered on exploration, detailing both its interactive workflow and the AI model at stake. We finally illustrate the pedagogical potential of our framework in two musical applications that we implemented, and discuss future research toward understanding how AI could be used by humans as collaborators for self-expression and appropriation of music computing practices.

## 1. INTRODUCTION

The introduction of the computer in music has brought a wealth of novel practices around sound and music. Cutting-edge technologies have been developed for sound synthesis, processing, analysis, and control, enabling the emergence of new music works, practices, notations, and performances.

However, such new technologies remain hard for people to appropriate. While music computing is now taught in many music institutions, it still suffers from its apparent complexity. As a consequence, many musicians stay attached to their classical practices and resign to appropriate music computing, while many musicians-to-be never take a chance to explore music computing systems. This is paradoxical as computing has become ubiquitous in the last ten years.

Can we think of a computer tool that would facilitate exploration and appropriation of music computing systems? In this paper we propose a computational framework for appropriating music computing. We first argue that the field of Human-Computer Interaction (HCI) may offer promising approaches to support appropriation by improving interface accessibility. We then propose to use Artificial Intelligence (AI) to provide users with human-centred interactions during their exploration of music computing tools. We finally illustrate our framework's potential with two example musical applications, and discuss future work to be done to better understand how AI could be used as collaborators by humans in their own appropriation of music computing and expression of musical ideas.

## 2. THE ROLE OF INTERACTION DESIGN IN MUSIC COMPUTING APPROPRIATION

We focus on a particular use case of music computing appropriation. The use case refers to the situation where a user makes use of a music computing system on his or her own, outside educational institutions. In this situation, appropriation issues arise during the first uses of the music computing system.

### 2.1. Appropriation in Music Computing

#### *2.1.1. Gathering Information on a System*

A first option to start using a system consists in gathering information on it—in a passive learning setup. Information on a system can be found in various media, from the most straightforward (*e.g.*, a text or video tutorial) to the most technical (*e.g.*, a research paper), as well as through online discussion (*e.g.*, an Internet forum). These activities are often time-consuming: users first have to find relevant information (which can be hard for obsolescent systems), then to filter it (*i.e.*, find what is useful for a specific goal), supposed that they have a specific goal in mind. Overall, time spent on passively learning hinders users to interact with sound and music directly, which might drive them away from experimenting with the system.

#### *2.1.2. Experimenting with an Interface*

A second option consists in starting interacting with the system's interface from scratch—in an active learning setup. Interacting with the interface implies trying many different actions directly to understand the functioning of the system (in a trial-and-error fashion), and eventually to achieve a specific goal. In our case of music computing, these activities are crucial as it is important for users to actively control sound so as to strengthen action-perception loops [15]. Yet, it is also possible that users get discouraged in interacting with the interface if they get too much

Hugo Scurto, Frédéric Bevilacqua, *Appropriating Music Computing Practices Through Human-AI Collaboration*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 115–120, 2018. 

**Figure 1**. Example of interface in a music computing system (here, the u-he Bazille VST [1] ).

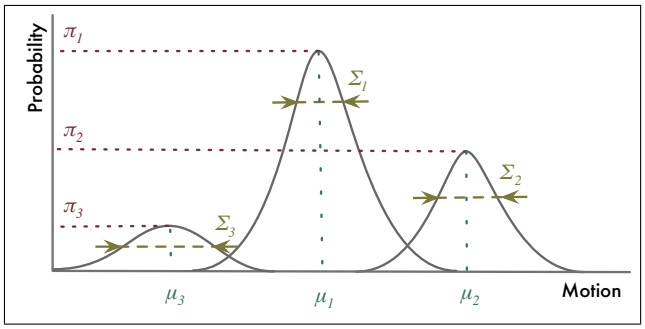



**Figure 2**. Example of model in a music computing system (here, a Gaussian Mixture Model for gesture-sound mapping [2] [13]).

error during their trials. Notions of appropriation [23] and novice to expert transition [7] are thus crucial in the design of interactions at stake in a given interface. Several works in the field of Human-Computer Interaction (HCI) have given guidelines for designing interactions in creative interfaces that facilitates appropriation [19]. We believe these could be applied to the design of interactions in music computing systems.

## 2.2. Human-Computer Interaction in Music Computing Systems

### *2.2.1. User Interface*

We identify two drawbacks of current music computing systems in the context of appropriation. First, most music computing systems' interfaces can look quite intimidating for completely novice users. Some of them directly derive from their analog ancestors (*e.g.*, sound synthesis engines, see Figure 1, or spatialization tools [4]): they are thus designed for expert users, not to facilitate interaction for novice users. There do have been attempts to improve interface accessibility of these systems [6, 5, 20]. Yet, these interface simplifications are often done to the detriment of the system's abilities: after having appropriated some tasks, users become limited by the interface's lack of sophistication.

### *2.2.2. Models and Representations*

The second drawback is that music computing systems often rely on complex models and representations that are not directly linked to sound or music. For example, improvisational systems [1] or gesture following systems [3] require users to have specific knowledge on models at stake (see Figure 2) to understand how parameters relate to system customization. Similarly, musical environemnts such as Max or PureData require to learn new representations as well as programming to start interacting. This drawback is common to both music novices and experts—who might know a certain amount of musical parameters, but do not know how they relate to new mathematical parameters—, and constitutes a major issue for music pedagogy—as such music computing systems have been shown promising for instrument learning [8] and embodied practice [2]. We believe redesigning interaction with these systems for improving appropriation could spread these pedagogical benefits among more people.

## 2.3. Interactive Machine Learning

### *2.3.1. Overview*

In the last decade, the field of Interactive Machine Learning has enabled more accessibility and appropriation of music computing systems by implementing human-centred interactions. At the crossroads between HCI and AI, it studies human interaction with algorithms, and integrates human actions in the design of algorithms themselves. Its applications has enabled the emergence of new music practices [11], as well as new computational frameworks [13]. As an example, the "mapping-by-demonstration" framework developed by Françoise [12] allows users to build custom gesture-sound mappings by directly demonstrating examples of gestures while listening to sounds, thus improving accessibility compared to previously-cited systems.

### *2.3.2. Pedagogical applications*

From a pedagogical point of view, the potential of interactive machine learning systems has been identified, yet little exploited. For gestural control of sound, they have been cited as allowing "learners to experience components of higher-level creativity and social interaction even before developing the prerequisite sensorimotor skills or academic knowledge" [18]. Interestingly, novel application domains, such as music therapy and musical expression for people with disabilities, have also emerged [21]. We believe extending interactive machine learning approaches to other music computing systems could constitute an opportunity to widen the reach of more music computing practices to more people.

[1] http://www.u-he.com/cms/bazille

[2] https://github.com/Ircam-RnD/xmm

## 3. CO-EXPLORATION: A FRAMEWORK PROPOSAL FOR AI-ASSISTED INTERACTION

We now present a framework proposal that focuses on assisting human exploration of music computing systems using AI. We first motivate the framework, then describe both its interactive workflow and AI modelling.

### 3.1. Motivation

#### *3.1.1. Why assisting exploration?*

Exploration is the early phase of learning during which a human iteratively acts on an interface and receives feedback information, allowing him or her to gradually grasp the system's functioning and qualities. As discussed previously, it is a crucial phase in appropriation regarding learning and skill development, as good or bad initial experience will determine the future degree of motivation and involvement of a learner term for a given task [7]. By aiming at a framework that assists this exploration phase, our wish is to lower the threshold for learners to directly interact with the system and sense its abilities, paving the way for further understandings of how the system actually works. Moreover, we argue that exploration may be a specific case of embodied interaction [15], in which expression plays a key role [16].

#### *3.1.2. How to assist exploration?*

To assist human exploration, we find it relevant to use the metaphor of transmission of knowledge between humans. Consider a human that has an idea but does not know how to convert it in a concrete realization. Usually, the human will ask assistance to a second human to realize this conversion—we call it the assistant. Iterative interaction between the two humans takes place, during which the assistant takes actions on the system and the human gives feedback on it—until converging to a final design. Our idea is to have an AI agent take the role of the assistant: AI acts on the system, upon which the human gives feedback. AI thus explores design possibilities in collaboration with the human, letting the human focus solely on aligning their conceptual space with the perceptual space offered by the AI—postponing the sensorimotor and/or academic learning phase to a later phase. Such co-adaptation phenomenon between the human and the machine has been shown as a useful mechanism for reducing the human's cognitive overload [17].

### 3.2. Workflow Definition

We propose to formalize the interactive workflow of such a framework—we call it "co-exploration" (see Figure 3).

#### *3.2.1. Design Through Co-Exploration*

Co-exploration stands for collaborative human-AI exploration of a given music computing system. The human explores the expressive abilities of the system (progressively learning interesting abilities in the system), while the AI explores the aesthetic preferences of the human (progressively learning which system parameters are relevant for the human). Design through co-exploration encapsulates the possibility for a (possibly novice) human to create a musical artifact from a (possibly unknown) computer system by collaborating with an AI agent in the exploration of design possibilities. A typical scenario would imply the agent generate an initial random solution to the human, who would progressively shape it through her preferences.

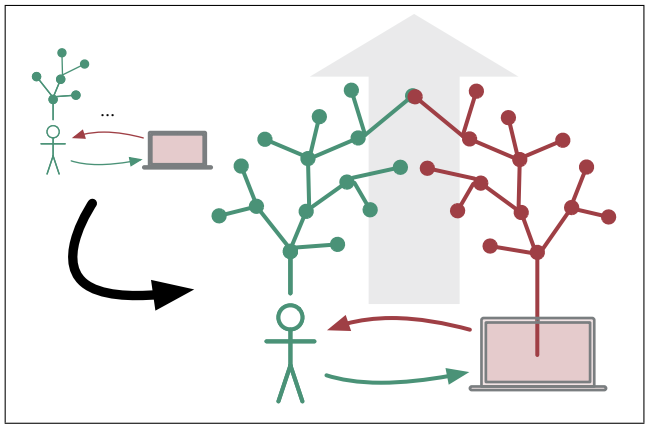


**Figure 3**. Co-exploration framework. In a standard situation (behind), a human explores a system by iteratively acting on it. In co-exploration (front), an AI agent explores a system in parallel to the human.

#### *3.2.2. Feedback-mediated interaction*

As we saw it, human interaction with the music computing system is mediated by an AI. Concretely, this means that the human does not interact with the system's interface: an AI agent does it instead. Instead, the human focuses on giving evaluative feedback on the AI agent's actions, judging the system's output on a perceptual level. Potentially, this feedback could be of any type—be it text-based, demonstration-based, or physiological.

#### *3.2.3. Reinforcement feedback*

We propose to investigate reinforcement feedback, which can be positive or negative. Advantages are threefold. First, it could encapsulate several kinds of feedback in one unique format, such as general advice (*e.g.* "this is good", "this is bad"), implicit knowledge (*e.g.* "do it more like this", "don't go that way"), as well as explicit specification (*e.g.* "this is exactly what I want", "never show me this again"). Second, it could be expressed relatively easily (compared to text-based feedback, which forces users to create a concrete verbalization of what they want). Third, it could give a sense of agency to the human (compared to physiological feedback, which most humans do not control). Overall, we hypothesize that communicating such high-level feedback could facilitate musical exploration of a system compared to specifying its low-level parameters.

### 3.3. AI Model

We propose to investigate the interactive use of a specific category of AI algorithms, called reinforcement learning, which applications in music computing have been few and far between [10, 9].

#### *3.3.1. Reinforcement learning*

Reinforcement learning defines a formal framework for the interaction between a learning agent and an environment in terms of states, actions, and rewards [22]. At time $t$, an agent senses its environment through an observation called state $S_t$ (typically, a vector of discrete parameters), and on that basis takes an action $A_t$ on it (typically, a set of discrete modifications on these parameters). At time $t+1$, in response to its action, the agent receives a reward $R_{t+1}$ from the environment, as well as a new state $S_{t+1}$. From this information, the agent iterates interaction, progressively learning how to optimize interaction with the environment so as to maximize the total amount of reward it receives over the long run.

#### *3.3.2. Learning from exploration*

Reinforcement learning algorithms differ from supervised (and unsupervised) learning algorithms. For the latters, learning typically occurs offline on the basis of a static training dataset, which is a set of labeled (or unlabeled) examples we would like the system to generalize behaviour from. In reinforcement learning, the agent learns online by directly interacting with its environment. As a result, a reinforcement learning agent must always balance between exploration and exploitation to improve its learning – exploration meaning trying new actions to discover which ones yield the most reward, and exploitation meaning choosing the best actions in terms of reward at the time of computation.

#### *3.3.3. Requirements for human interaction*

In the case of co-exploration, we must add another element to the formal framework defined above. We propose that a human would be responsible for giving reward to the learning agent, as a consequence of the agent's action toward the environment's state (see Figure 4). Our hypothesis is that the numerical reward could constitute a feedback channel from the human to the AI agent (*e.g.* telling the agent that its action has been good or bad from the human's subjective point of view).

Interactive agent teaching have been investigated in previous research, leading to the creation of efficient learning agents [14]. Our research differs from these works in the sense that it focuses on interactive agent teaching from the human point of view (how it is "efficient" for the human, not necessarily for the agent), and that the tasks to be learned in creative activities may have different properties than those in goal-oriented activities (such as learning how to play Tetris).

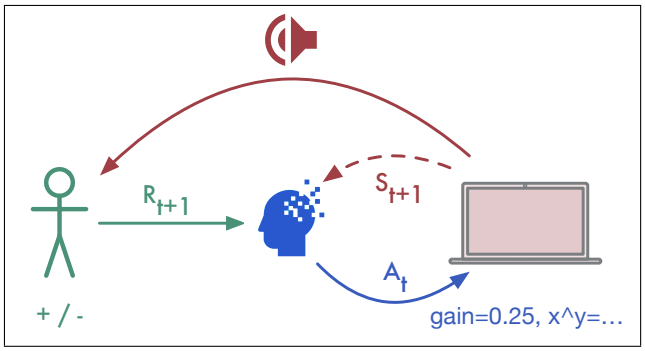



**Figure 4**. Co-exploration workflow. Interaction between the human and the music computing system is mediated by an AI agent (in blue). At time $t$, the agent acts directly on the system's parameters. At time $t+1$, the system generates a new state (for example, a sound), that is subjectively evaluated by the human through a reinforcement feedback. By iterating the loop, the agent learns how to co-explore.

## 4. EXAMPLE MUSICAL APPLICATIONS

We implemented a first prototype of our co-exploration agent—we call them "co-explorers"—and applied it within two existing music computing systems.

### 4.1. Implementation

We are currently implementing coax, a software library for collaborative human-AI exploration. Its architecture will allow users to choose among a variety of reinforcement learning agents, and to modify some high-level parameters that control their exploration. Agents can be connected to any kind of interactive music system that sends and receives OSC messages. The current prototype implements the Sarsa learning algorithm [22] upon a set of Python classes. Its interface consists in two buttons: one for indicating positive feedback, one for negative—we will discuss further improvements in next section. We implemented our two applications in the Max/MSP environment.

### 4.2. First application: VST Exploration

#### *4.2.1. Motivation*

The first application connects an AI agent to a VST (*u-he Bazille*, see Figure 1). VSTs are software interfaces that allows to create sounds by combining various synthesis algorithms and audio processings. Their interface typically consists of knobs and faders, directly linked to low-level parameters of synthesis and processing. While these music computing systems are widely used and effective among expert composers, their interface may constitute a huge barrier for novice users to launch into exploring and understanding their functioning, preventing them from appropriating digital sound creation.

#### 4.2.2. Description

The workflow consists of users first listening to a random sound generated by the VST, give positive or negative feedback on it depending of their subjective evaluation of the sound, then listen to a new sound that was synthesized taking into account previous feedback. Formally, the environment's state consists of a vector of twelve VST parameters; the agent's actions consists of moving one of these parameters up or down. The resulting collaboration consists in the user focusing on developing its listening abilities and tastes toward the VST, while the AI agent learns how to convert the user's tastes in terms of VST parametrization. A preliminary evaluation led with four expert computer musicians experimenting with our system with three different VSTs confirmed our initial conjectures on how useful co-exploration could be for appropriating a VST—we should discuss such results in a future publication.

### 4.3. Second application: Mapping Exploration

#### 4.3.1. Motivation

The second application connects an AI agent to a gesture-sound mapping (using the *XMM* library, see Figure 2). Mappings are complex functions that link gestural devices to sound synthesizers. They allow performers to focus on their physical gestures in their musical practice and embodied engagement with sound. However, they typically imply a phase of parametrization, which require knowledge on mathematical modelling and programming. This can hinder novice users as well as expert musicians to create their own gestural controller and develop self-expression.

#### 4.3.2. Description

The workflow consists of users first experimenting with a random gesture-sound mapping, give positive or negative feedback on it depending of their subjective evaluation of the mapping, then experiment with a new mapping that was generated taking into account previous feedback. Formally, the environment's state consists of a vector of twelve mapping parameters; the agent's actions consists of moving one of these parameters up or down. The resulting collaboration consists in the user focusing on developing its sense of corporeal engagement with sound, while the AI agent learns how to convert it in terms of mapping parametrization. Ultimately, as users give feedback on mappings that they experiment with, the AI agent would be able to generate mappings that either suit them (exploitation), or aim at surprising them (exploration).

## 5. CONCLUSION AND FUTURE WORK

In this paper we have formulated a scientific hypothesis on how human-AI collaboration could support music computing appropriation. Part of future work will consist of refining the design of learning algorithms, as to optimize human-AI exploration. Also, a general interface for co-exploration might be designed. We will keep on testing our framework with other music computing systems as well to spread the range of musical activities covered—for example, connecting co-explorers with sequential music content (such as notes or events).

An important part of future work will consist of evaluating our framework with expert users, *i.e.* users that work around music at a professional level. We are currently leading and building a set of case studies that aim at better understanding how AI agents could assist these users in their exploration of music computing practices. Specifically, our aims are (1) assessing whether co-exploration might be useful for them compared to their standard exploration strategies, and (2) studying how co-exploration might be useful on a longer term along their creative practice. These two-scale studies would allow us to better quantify the pedagogical benefits our framework might offer.

Finally, we will lead real-world applications with novice users, *i.e.* users that are neither musicians, nor computer specialists. The goal of these applications will rely on observing how co-exploration might constitute an entry point for these novice users to music computing practices. Based on previous works [18], we envision that our interactive machine learning workflow may allow novices to experience creative and social aspects of music before devoting time to learning and understanding technical aspects. Overall, we will be able to better understand how AI could be used by humans as collaborators for self-expression and appropriation of novel music computing practices.

## 6. ACKNOWLEDGEMENTS

We thank Baptiste Caramiaux, Jules Françoise and Benjamin Matuszewski for useful discussions and suggestions.

## 7. REFERENCES

[1] G. Assayag, G. Bloch, M. Chemillier, A. Cont, and S. Dubnov. Omax brothers: a dynamic topology of agents for improvization learning. In *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, pages 125–132. ACM, 2006.

[2] F. Bevilacqua, F. Guédy, N. Schnell, E. Fléty, and N. Leroy. Wireless sensor interface and gesture-follower for music pedagogy. In *Proceedings of the 7th International Conference on New Interfaces for Musical Expression (NIME)*, pages 124–129. ACM, 2007.

[3] F. Bevilacqua, B. Zamborlin, A. Sypniewski, N. Schnell, F. Guédy, and N. Rasamimanana. Continuous realtime gesture following and recognition. In *International gesture workshop*, pages 73–84. Springer, 2009.

[4] T. Carpentier, M. Noisternig, and O. Warusfel. Twenty years of ircam spat: looking back, looking forward. In *41st International Computer Music Conference (ICMC)*, pages 270–277, 2015.

[5] M. Cartwright and B. Pardo. Synthassist: Querying an audio synthesizer by vocal imitation. In *Proceedings of the 14th International Conference on New Interfaces for Musical Expression (NIME)*, 2014.

[6] M. Cartwright, B. Pardo, and J. Reiss. Mixploration: Rethinking the audio mixer interface. In *Proceedings of the 19th International Conference on Intelligent User Interfaces*, pages 365–370. ACM, 2014.

[7] A. Cockburn, C. Gutwin, J. Scarr, and S. Malacria. Supporting novice to expert transitions in user interfaces. *ACM Computing Surveys (CSUR)*, 47(2):31, 2015.

[8] A. Cont. Antescofo: Anticipatory synchronization and control of interactive parameters in computer music. In *International Computer Music Conference (ICMC)*, pages 33–40, 2008.

[9] N. Derbinsky and G. Essl. Exploring reinforcement learning for mobile percussive collaboration. In *Proceedings of the 2012 International Conference on New Interfaces for Musical Expression (NIME)*, 2012.

[10] R. Fiebrink and B. Caramiaux. The machine learning algorithm as creative musical tool. *arXiv preprint arXiv:1611.00379*, 2016.

[11] R. Fiebrink, D. Trueman, N. C. Britt, M. Nagai, K. Kaczmarek, M. Early, M. Daniel, A. Hege, and P. R. Cook. Toward understanding human-computer interaction in composing the instrument. In *Proceedings of the 2010 International Computer Music Conference (ICMC)*, 2010.

[12] J. Françoise. *Motion-sound mapping by demonstration*. PhD thesis, Université Pierre et Marie Curie, 2015.

[13] J. Françoise, N. Schnell, R. Borghesi, and F. Bevilacqua. Probabilistic models for designing motion and sound relationships. In *Proceedings of the 2014 International Conference on New Interfaces for Musical Expression (NIME)*, pages 287–292, 2014.

[14] W. B. Knox and P. Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM, 2009.

[15] M. Leman. *Embodied music cognition and mediation technology*. MIT Press, 2008.

[16] M. Leman. *The expressive moment: How interaction (with music) shapes human empowerment*. MIT press, 2016.

[17] W. E. Mackay. Responding to cognitive overload: Co-adaptation between users and technology. *Intellectica*, 30(1):177–193, 2000.

[18] D. Morris and R. Fiebrink. Using machine learning to support pedagogy in the arts. In *Personal and ubiquitous computing*, pages 1631–1635. Springer, 2013.

[19] M. Resnick, B. Myers, K. Nakakoji, B. Shneiderman, R. Pausch, T. Selker, and M. Eisenberg. Design principles for tools to support creative thinking. In *National Science Foundation Workshop on Creativity Support Tools. Washington DC.*, 2005.

[20] D. Schwarz, G. Beller, B. Verbrugghe, and S. Britton. Real-time corpus-based concatenative synthesis with catart. In *9th International Conference on Digital Audio Effects (DAFx)*, pages 279–282, 2006.

[21] H. Scurto, R. Fiebrink, et al. Grab-and-play mapping: Creative machine learning approaches for musical inclusion and exploration. In *Proceedings of the 2016 International Computer Music Conference*, 2016.

[22] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 2011.

[23] V. Zappi and A. McPherson. Dimensionality and appropriation in digital musical instrument design. In *Proceedings of the 2014 International Conference on New Interfaces for Musical Expression (NIME)*, pages 455–460, 2014.

# REDI-MUSIX :
# RESEAU POUR LA DISTRIBUTION DE LA MUSIQUE MIXTE

*Alexander Mihalic*
Musinfo
mihalic@musinfo.fr

*Mikhail Malt*
Musinfo
malt@musinfo.fr

## RÉSUMÉ

La mutation numérique de la société donne lieu à la production d'œuvres qui croisent les nouvelles technologies et les instruments traditionnels. Les assemblages d'outils destinés aux tâches multiples favorisent les installations uniques, difficilement maîtrisées par les interprètes ([1], [2], [3], [4], [5] et [6]). Pour jouer de la musique mixte, la seule solution viable est la création et la fabrication d'un instrument dédié. Si cette démarche permet la standardisation de l'instrument électronique, il reste en outre indispensable de sécuriser l'accès au répertoire existant. La mobilité et la nécessité de produire ces œuvres indépendamment du lieu et de leur complexité technologique nous amènent à proposer un projet de réseau de distribution de la musique mixte dans les conservatoires et les écoles de musique, quelles que soient leurs situations géographiques ou leurs tailles.

Le projet proposé RéDi-Musix, porté par Musinfo et soutenu par le Ministère de la culture et de la communication[1], consiste à mettre en place un service de distribution de configurations électroacoustiques au travers d'un réseau numérique. Ce réseau connectera les établissements musicaux en France et à l'étranger afin de favoriser la création et l'accès au patrimoine vers un public le plus large possible. Les configurations électroacoustiques seront distribuées sur des terminaux appelés « Sampo », placés dans les établissements partenaires et permettant de jouer la musique mixte de façon autonome.

## 1. INTRODUCTION

La musique mixte met en relation un instrument acoustique avec un environnement électroacoustique. Pour produire ce type de musique, nous avons besoin de partitions, mais aussi de configurations et d'éléments matériels et logiciels, indissociables d'une telle œuvre. La mise en place de ces configurations matérielles et logicielles ainsi que leur gestion (mise à jour des configurations et mise en route) sortent dans la grande majorité des cas du champ de connaissance et de maitrise des interprètes.

Si la distribution des partitions ne pose pas de problème particulier aujourd'hui, les partitions étant vendues sur les sites internet et pouvant être téléchargées directement sur les ordinateurs et imprimées chez les musiciens, le problème se pose pour les configurations matérielles et logicielles. Dans ces conditions, comment aller directement à l'essentiel, soit l'étude, la composition et la performance de cette catégorie d'œuvres, sans perdre trop de temps avec les éléments techniques ? De plus, comment échanger efficacement des informations, configurations et fichiers sons, lors de la phase de composition, entre compositeur et interprète, pour ce même dispositif, directement chez le musicien pour qu'il puisse travailler et jouer ? C'est dans ce contexte et pour palier à ces difficultés, que voit le jour le projet RéDi-Musix.

## 2. LE PROJET REDI-MUSIX

Le projet « RéDi-Musix - Réseau de distribution de la musique mixte »[2], repose d'un côté sur un instrument/terminal et de l'autre sur un service centralisé de distribution de contenus/configurations sous forme d'une base de données en ligne (**Figure 1**).

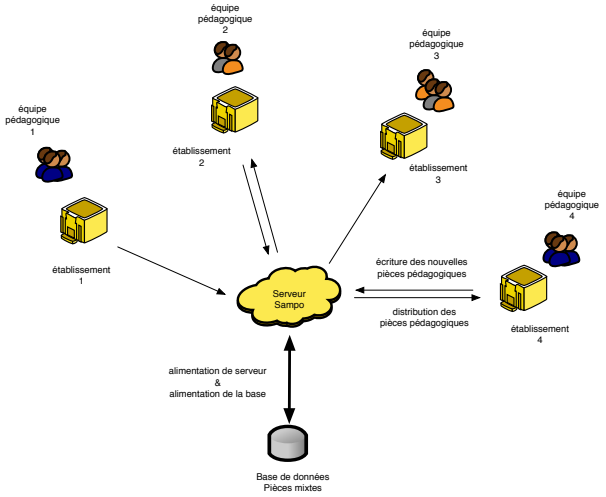



**Figure 1**. Schéma du réseau RéDi-Musix.

Ce projet propose de réunir tous les éléments nécessaires à la production de la musique mixte depuis sa création et sa distribution jusqu'à son archivage. Techniquement le projet se présente comme un service

[1] http://www.culturecommunication.gouv.fr/Aides-demarches/Appels-a-projets/Appel-a-projets-services-numeriques-innovants-2016

[2] http://www.redi-musix.com/

Alexander Mihalic, Mikhail Malt, *RéDi-Musix : réseau pour la distribution de la musique mixte*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 121–127, 2018. 

appelé communément « cloud » (nuage) ainsi que des éléments matériels et logiciels annexes :

- **terminal Sampo** - instrument électroacoustique autonome et universel et qui permet d'accéder au répertoire sur le cloud
- **logiciel P-Soft** - logiciel pour les musiciens désireux d'échanger leurs configurations et déposer des pièces dans la base à la disposition des autres musiciens.
- **base de données** - base de pièces du répertoire de la musique mixte
- **stockage des fichiers** - fichiers sons, fichiers de configurations du dispositif et accès aux fichiers
- **communauté d'utilisateurs** - les compositeurs/musiciens qui déposent leurs œuvres dans la base

## 3. INSTRUMENT/TERMINAL SAMPO

Sampo est un dispositif matériel intégrant tous les éléments nécessaires à la performance et à la composition de musique mixte (**Figure 2**). Il offre une solution complète, dès la captation du son par le micro, en passant par sa transformation à l'aide des pédales[3], jusqu'à la sortie sur les haut-parleurs intégrés. Pour les salles de concert, des sorties audio professionnelles sont prévues. Le jeu sur Sampo se fait à l'aide de pédales, ce qui rend cet outil plus maniable. De plus, son utilisation ne nécessite aucune connaissance technique préalable ni aucune installation logicielle. Les pédales servent d'interface de gestion de l'environnement électroacoustique. En fonction de l'œuvre jouée, les affectations des pédales permettent de contrôler directement des paramètres de processus temps réel (volume, temps de retard, transposition, etc.) ou / et la gestion des déclenchements de fichiers sons (arrêt, pause, déplacement en avant ou en arrière). C'est un dispositif qui se veut intuitif. Chaque geste (mouvements des pédales) affecte les paramètres du son de l'instrument, permettant ainsi au musicien d'appréhender immédiatement le résultat. Pouvant être utilisé pour la création sonore avec n'importe quel instrument acoustique, ainsi que dans les classes de composition, son ergonomie le rend particulièrement intéressant pour l'enseignement de la musique électroacoustique, des musiques actuelles et de l'improvisation dans les conservatoires et dans les écoles de musique[4].

[3] *Écart à l'Equilibre* pour violon et Sampo (Arturo Gervasoni) : https://www.youtube.com/watch?v=v7y7OvpoakQ

[4] https://www.youtube.com/watch?v=_3h_J3TRuX0 et https://www.youtube.com/watch?v=jGqrcVoY07g

### 3.1. P-Soft

Une application autonome sur PC ou Mac est en préparation pour les utilisateurs qui désirent créer des configurations et des pièces pour Sampo. Il s'agit principalement des compositeurs ou des enseignants et des élèves de classes de compositions dans les conservatoires. Une configuration est l'ensemble des modules de traitement temps réel avec un parcours audio spécifique contenant les assignations entre les pédales et les différents paramètres que contiennent les modules. Chaque œuvre possède sa propre configuration.

**Figure 2**. Boîtier/unité de traitement – élément principal de Sampo

L'application est un éditeur de configurations pour Sampo et son rôle principal est de rassembler tous les éléments nécessaires pour une pièce. La principale différence avec Sampo est la possibilité d'ajouter et d'organiser les fichiers sons d'une pièce, ainsi que tous les autres fichiers qui accompagnent la pièce, comme une partition PDF ou un exemple sonore de la pièce.

En effet, les fichiers sons utilisés pour la pièce contiennent des marqueurs ajoutés par l'utilisateur. Ces marqueurs déterminent les endroits dans les fichiers sons où l'interprète peut intervenir sur leur déroulement. Une fois que l'utilisateur a édité les fichiers sons et ajouté les marqueurs aux endroits voulus – au minimum un marqueur au début de chaque fichier son pour le déclencher – il dépose le fichier dans l'application. Il est aussi possible d'ajouter des fichiers sons « d'alerte audio » ou « click-track ». Ce sont des fichiers qui produisent des impulsions pendant la lecture de fichiers sons. Ces impulsions sonores sont envoyées vers le casque connecté au Sampo.

L'application est reliée à l'utilisateur et à la machine hôte et se connecte directement sur son compte utilisateur de la base. Contrairement au Sampo, l'utilisateur ne peut télécharger sur son ordinateur que les pièces qu'il a lui-même créées.

## 4. BASE DE DONNEES

La base de données est un élément central du projet et de la mise en place du réseau des Sampos et de leur interconnexion mutuelle. La nécessité de ce réseau est apparue avec les prototypes de Sampos fabriqués entre novembre 2014 et mai 2015.

En effet, l'un des avantages matériels de Sampo est la possibilité de l'envoyer par la poste, comme n'importe quel instrument, aux interprètes ou compositeurs dans le monde entier. Il garantit la restitution exacte dans les deux endroits, mais à la condition de disposer des mêmes configurations et des fichiers nécessaires pour l'exécution de la pièce.

Or, lors de la préparation d'un concert, il est fréquent de changer les œuvres et en demander d'autres qui ne soient pas présentes localement sur Sampo. Le problème qui se pose alors est que la configuration interne change en fonction de l'œuvre que l'on veut jouer et ceci nécessite de trouver un moyen d'accès aux configurations et fichiers nécessaires pour les œuvres directement à partir de Sampo. Chaque Sampo dispose d'une connexion Wifi, ce qui permet de déposer manuellement les configurations et les fichiers directement à partir de la base centrale (**Figure 3**) et sans l'intervention du musicien. Avec quelques prototypes il était ainsi possible de gérer les dépôts des configurations manuellement pour chaque Sampo. Toutefois, l'évolution actuelle, c'est-à-dire la fabrication des Sampo en séries, nécessite une autonomie de gestion des contenus de Sampo.

Nous avons donc mis en place le projet d'une base en ligne, actuellement en cours, sur laquelle les Sampos se connectent automatiquement à partir du moment où ils ont accès à internet. Pour l'utilisateur de Sampo, l'accès à la base est totalement transparent et en absence de connexion l'utilisateur ne parcourt que la base locale.


**Figure 3**. Affichage de la base dans la fenêtre sur Sampo.

Une fois connecté, l'affichage inclut la base centrale où l'utilisateur peut récupérer les configurations des œuvres dont il souhaite faire l'acquisition. Ainsi, le terminal Sampo se connecte sur la base, affiche son contenu pour permettre les uploads et downloads des configurations des utilisateurs. L'utilisateur qui travaille avec Sampo peut facilement créer et gérer des pièces, les déposer sur le serveur ou télécharger les pièces publiques ou celles partagées avec lui. La seule limite consiste en l'absence de possibilité d'édition des fichiers son et des partitions. Le logiciel P-Soft possédant des fonctionnalités d'édition plus poussées, permet de préparer les configurations et le contenu pour la base. Ces deux outils sont donc complémentaires.

Sampo dispose de trois fenêtres d'accès à la base de données en ligne :

- liste des pièces
- détails de la pièce
- gestion de partage dans les groupes

Les deux premières permettent de faire la recherche et d'accéder aux détails de la pièce. La troisième fenêtre affiche les outils pour la gestion de partage de la pièce et la création des groupes d'utilisateurs.

Cela permet à un compositeur d'écrire une pièce pour un interprète (avec Sampo ou avec le logiciel P-Soft) qui soit à un autre endroit et de synchroniser leur travail (les fichiers de configuration et fichiers son) par réseau.

### 4.1. Structure & terminologie

La base se compose de quatre principales tables :

- utilisateurs
- Sampos
- pièces
- compositeurs

#### *4.1.1. Utilisateur*

L'utilisateur s'inscrit dans la base, son compte est créé, et il lui est possible d'accéder aux « pièces » dont il a la permission. L'utilisateur peut aussi créer lui-même les pièces. L'utilisateur est lié ou non à un Sampo. S'il est lié, alors Sampo effectue la connexion automatique à la base et présente à l'utilisateur l'état de son compte. L'utilisateur peut recevoir l'application qui émule Sampo dans le but de composer pour Sampo. Ceci veut dire assembler les fichiers nécessaires pour une configuration et les télécharger sur la base pour les rendre accessibles aux Sampos.

#### *4.1.2. Pièce - configuration*

La « pièce » ou configuration est un ensemble des fichiers et de leurs liens pour présenter et exécuter une œuvre avec Sampo. Ceci concerne le fichier de configuration interne des effets et leurs affectations aux pédales, les fichiers sons avec les marqueurs, fichier PDF de la partition, fichier d'un enregistrement, informations sur la pièce telles que date de composition, instrumentation, description, etc.

L'utilisateur peut créer lui-même des pièces sur son Sampo et les déposer sur le serveur. L'utilisateur peut voir le contenu de la base visible - ce sont les pièces des compositeurs. Un compositeur est lié à une ou plusieurs pièces.

#### *4.1.3. Compositeur*

L'utilisateur n'est pas compositeur par défaut, mais il peut demander la création d'un profil compositeur avec son nom et les données qui lui seront demandées, comme biographie, photo, adresse, etc. Une fois le profil créé après vérification des données, l'utilisateur obtient l'accès à la création des pièces au nom du compositeur en question.

#### *4.1.4. Relations : utilisateur/pièce/compositeur*

Il peut y avoir en conséquence un ou même plusieurs utilisateurs qui écrivent des pièces au nom d'un compositeur. Il s'agit alors de « transcriptions » des pièces existantes pour un compositeur qui lui, en tant que personne physique, peut ne pas être utilisateur de la base.

Les pièces des compositeurs, contrairement aux pièces des utilisateurs, peuvent être déclarées publiques. Ceci veut dire que la pièce est verrouillée et qu'il n'est plus possible d'y faire de modifications. À partir de ce moment, la pièce devient alors visible pour tous les utilisateurs. Les utilisateurs peuvent voir les données de la pièce telles que titre, instrumentation, extraits sonores, extrait de partition, etc. L'accès à la partition de la pièce dans sa totalité est décidé par le compositeur. Cet accès peut être l'accès en libre téléchargement ou l'achat de la partition chez l'éditeur.

L'accès à la configuration de la pièce se fait au cas par cas. Si la pièce est déclarée libre par le compositeur, l'utilisateur peut télécharger la configuration immédiatement sur son Sampo. Si la partition de la pièce est éditée, il est possible que l'éditeur mette à la disposition libre les éléments pour jouer la partie électronique de la pièce, fichiers sons ou patchs, et dans ce cas le téléchargement de la configuration est disponible. Si l'éditeur fournit les éléments pour jouer la partie électronique uniquement avec l'achat de la partition, la configuration n'est disponible pour l'utilisateur qu'après son achat auprès de l'éditeur.

#### *4.1.5. Groupes*

Après avoir créé une pièce en dehors d'un profil compositeur, l'utilisateur est le seul à pouvoir la visualiser et éditer sa configuration. S'il crée la pièce avec l'application, et s'il est lié en même temps à un Sampo, il peut voir sur Sampo sa pièce et la télécharger.

S'il le souhaite, l'utilisateur peut partager sa pièce pour plusieurs utilisateurs et Sampos. Ce partage s'effectue à travers la création de groupes à partir de la pièce existante. Il est donc seulement possible de créer un groupe pour partager une pièce que l'utilisateur a créée, et il n'est pas possible de créer un groupe sans pièce.

Pour créer un groupe, l'utilisateur crée ou sélectionne sa pièce et ensuite choisit les utilisateurs ou les Sampos qui vont faire partie du groupe. L'utilisateur choisit si les membres du groupe peuvent modifier la configuration ou uniquement la lire. L'utilisateur peut créer un groupe pour chaque pièce. Si la pièce est effacée, le groupe l'est aussi.

L'utilisateur peut créer plusieurs groupes à partir d'une pièce. Chaque groupe peut contenir des utilisateurs et Sampos différents et chaque groupe est autorisé ou non à modifier la pièce. La dernière modification de la pièce sur le serveur est récupérée pour tous les utilisateurs du groupe.

Le groupe est une solution pour travailler à distance sur des pièces non publiques. Ceci est particulièrement utile dans le cadre d'une création de la pièce où le compositeur travaille avec le logiciel et l'interprète travaille chez lui avec son Sampo. L'utilisateur/compositeur crée une pièce et aussi le groupe dans lequel il inclut soit l'interprète utilisateur, soit le Sampo. La pièce est alors visible à partir du compte de l'utilisateur/interprète ou de son Sampo. L'utilisateur/compositeur peut changer la configuration en fonction des remarques de l'interprète qui peut même avoir la possibilité de changer les paramètres dans la configuration et échanger ainsi avec le compositeur.

Un autre but de la création des groupes est le travail collaboratif dans les conservatoires. C'est un moyen simple et efficace pour faire travailler les pièces des élèves des classes de composition aux élèves instrumentistes. De plus, les groupes permettent d'inclure d'autres conservatoires pour échange des pièces.

### 4.2. Accès à la base

#### *4.2.1. Environnements*

La base des pièces mixtes est accessible à partir de trois environnements (**Figure 4**) :

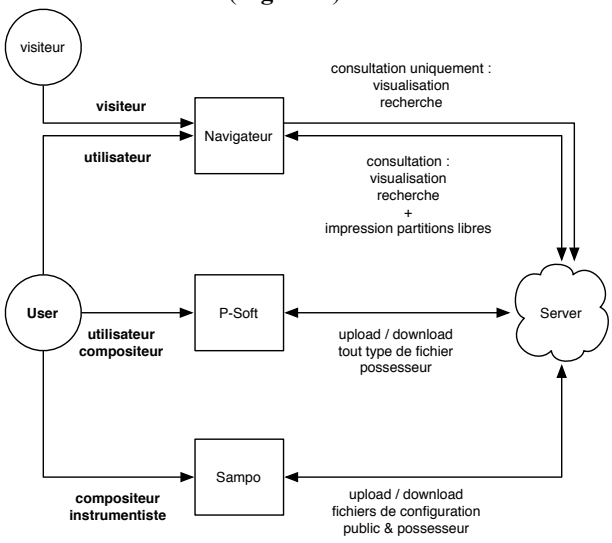



**Figure 4**. Environnements et accès à la base en ligne de RéDi-Musix.

Le premier accès se fait directement à partir de Sampo où l'utilisateur peut faire des recherches, téléchargements, mises à jour des pièces existantes, création de nouvelles pièces et leur dépôt dans la base.

De même, l'application P-Soft, destinée principalement aux compositeurs, permet la connexion à la base avec le login de l'utilisateur qui installe l'application. Si l'application permet d'envoyer les fichiers sons ou PDF, elle ne peut en revanche recevoir que les « pièces » composées par l'utilisateur lui-même.

Enfin, il est possible de se connecter à la base à partir d'un navigateur internet. Dans ce cas, l'utilisateur peut accéder aux informations dans son espace personnel, c'est-à-dire aux œuvres dont il dispose et à l'impression des fichiers PDF des partitions à partir de son espace.

### *4.2.2. Espace personnel*

Le contenu de la base est en grande majorité accessible uniquement aux utilisateurs inscrits. Le public ne peut se connecter à la base qu'à partir d'un navigateur et n'a accès qu'aux informations générales, comme la liste des compositeurs ou des pièces, la possibilité de faire des recherches dans la base, mais il ne peut pas télécharger les partitions ou déposer de nouvelles pièces.

Un utilisateur inscrit peut accéder à son espace où il peut télécharger les partitions libres, modifier les pièces qu'il a créées, déposer les nouvelles pièces privées, etc.

### *4.2.3. Navigateurs*

L'accès le plus simple à la base s'effectue à travers un navigateur internet sur n'importe quel ordinateur ou tablette. La base est présentée sur le site pour être librement consultable. Seules les informations de base sur la pièce sont proposées comme les titres, instrumentations, extraits de partitions ou exemples sonores. L'utilisateur inscrit dans la base a, en plus, l'accès au téléchargement des partitions libres déposées par les compositeurs.

## 5. LE SERVICE CENTRALISE DE DISTRIBUTION DE CONTENUS/CONFIGURATIONS

Le but du projet est, entre autres, d'interconnecter les conservatoires et les écoles de musique et permettre l'échange et la création d'œuvres pédagogiques de la musique mixte, facilitant ainsi la diffusion des contenus culturels. Pour cela nous avons mis à disposition dans les établissements partenaires des terminaux - Sampo - sur lesquels il est possible de jouer et composer la musique mixte dans les mêmes conditions et avec les mêmes résultats sonores dans tous les lieux. La distribution des configurations/compositions sur les terminaux (Sampo) se fait au travers d'un serveur et d'une base de données. Cette base est alimentée d'une part avec des pièces du répertoire et d'autre part par les compositeurs-élèves durant toute l'année scolaire. Les pièces pédagogiques sont ainsi disponibles dans tous les conservatoires disposant d'un Sampo. Les pièces composées sont immédiatement mises à la disposition de tous les partenaires du projet et travaillées par des enseignants et étudiants de ces établissements.

## 6. COMMUNAUTE D'UTILISATEURS

Nous avons, actuellement, un total de 8 partenaires culturels (**Figure 5**) dans le cadre du projet. La carte ci-contre représente les villes dont les conservatoires se joignent au déploiement du réseau de service RéDi-Musix.

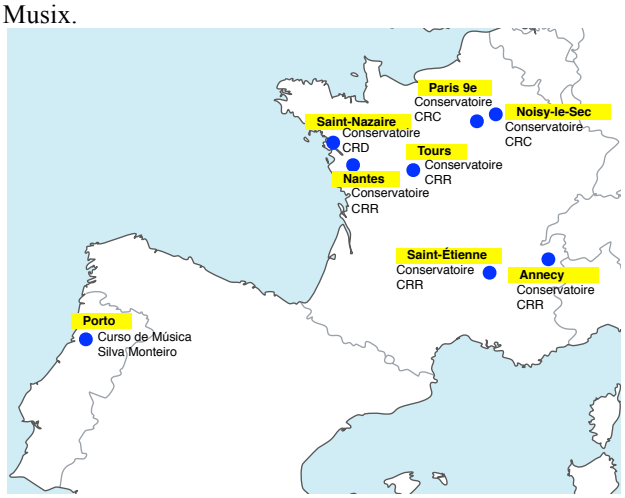



**Figure 5**. Carte des partenaires du projet RéDi-Musix.

Voici la liste des partenaires du projet :

- Conservatoire d'Annecy (CRR)[5]
- Conservatoire Massenet à Saint-Étienne (CRR)[6]
- Conservatoire de Saint-Nazaire (CRD)
- Conservatoire de Nantes (CRR)
- Conservatoire Francis Poulenc de Tours (CRR)
- Conservatoire de Noisy-le-Sec (CRC)
- Conservatoire du 9e arrondissement de Paris (CRC)
- Curso de Música Silva Monteiro à Porto (Portugal)

Dans chaque établissement nous avons mis en place une équipe composée d'enseignants d'instruments (par exemple, flûte, violon, clarinette...) et de leurs élèves, ainsi que d'enseignants de composition avec leurs élèves qui composeront pour les divers instruments acoustiques accompagnés du terminal Sampo.

### 6.1. Pratique dans les conservatoires

Sampo permet de sensibiliser les instrumentistes aux musiques électroacoustiques, mixtes, improvisées, contemporaines, jazz… en puisant dans le répertoire déjà écrit. Comme avec son instrument acoustique, le musicien est invité à écouter le son qu'il produit et réfléchir à la manière d'obtenir un effet particulier. Un click-track et la possibilité de se déplacer à volonté dans

[5] https://www.youtube.com/watch?v=ArRLnUfrzB4
[6] https://www.youtube.com/watch?v=6av2w8UbjOA

le fichier son sans quitter l'instrument de ses mains, facilitent le travail du musicien.

### 6.2. Catégories de pièces

Les pièces - ensembles de configurations électroacoustiques et de contenus fixes - sont accessibles à travers de l'interface graphique sur l'écran tactile de Sampo. Une fois sélectionnée, la pièce est chargée et immédiatement prête à être jouée.

Deux catégories de pièces sont travaillées dans le cadre du projet.

En premier lieu, les compositeurs et les élèves dans les classes de composition composent de courtes pièces pédagogiques mixtes pour les instruments choisis au début du projet. Ces pièces sont placées sur le serveur à la disposition de tous les autres établissements pour les classes d'instruments. Les élèves de tous les établissements peuvent ainsi préparer les pièces pour les concerts de restitution qui auront lieu à la fin du projet.

En plus des pièces nouvellement composées pour l'occasion, les étudiants et les enseignants peuvent travailler les pièces mixtes déjà existantes et qui sont ou seront transcrites pour le terminal Sampo. Les pièces sont disponibles sur le serveur et sont distribuées directement sur le terminal à la demande des établissements. Les musiciens avec les instruments acoustiques pourront ainsi interpréter les pièces dans des conditions identiques dans tous les établissements en France ou à l'étranger.

## 7. EXEMPLE D'UNE ACTION PEDAGOGIQUE

Les *Reflets*[7] (2000-2002) et les *Contemplations*[8] (2004) sont deux ensembles de pièces pédagogiques, pour instrument solo et support fixe commandés par le Conservatoire de Bagnolet. Ces pièces ont été composées pour initier de jeunes instrumentistes au jeu avec l'électronique, et conçues en étroite collaboration avec les professeurs d'instruments. Malgré la volonté de faciliter au maximum la mise en œuvre des pièces, prenant le soin de les adapter au niveau des étudiants, en concevant une électronique qui tenait sur un CD et un CD d'étude avec une piste métronome, la mise en route de ces œuvres et leur étude et performance étaient toujours complexes. Insérer un CD dans la chaine, le besoin d'une deuxième personne pour démarrer la lecture du CD et le déroulement complexe des répétitions et séances d'étude (arrêt, « fast forward » ou « rewind » pour caler le CD sur des positions temporelles précises correspondant aux mesures) freinaient le travail des élèves.

Lors d'une réalisation au Conservatoire de Saint-Nazaire (02/12/2017), nous avons pu constater que l'environnement proposé par RéDi-Musix avec le Sampo permettait un accès plus simple et direct aux œuvres. Il est aisé au jeune musicien de démarrer la bande à la pédale, arrêter et pouvoir choisir une mesure pour redémarrer, jouer avec ou sans click-track. Autant l'étude que la répétition pour le concert se fait avec aisance de la part de l'instrumentiste. L'environnement proposé s'est montré efficace permettant une approche directe des courtes pièces, évitant de confronter les jeunes instrumentistes à des problèmes techniques hors de leur intérêt et du contexte musical, leur permettant un accès direct à l'interprétation.

D'autres actions pédagogiques de ce type dans les conservatoires partenaires confirment la pertinence de la solution de jouer les œuvres mixtes avec le terminal Sampo.

## 8. CONCLUSIONS

La musique mixte, avec des éléments temps réel et/ou avec des supports fixes, est actuellement tributaire de la mise en place d'environnements matériels et logiciels qui, par leur complexité et la nécessité de connaissances techniques spécifiques, découragent souvent les interprètes. En plus, l'agencement et la mise en marche de ces environnements est souvent chronophage, laissant peu ou pas de place à l'interprétation musicale, cela même pour des œuvres dites « pour bande ». La mise en place d'environnements matériels et logiciels est au même niveau que la lutherie pour un interprète qui ne fabrique pas l'instrument sur lequel il joue. Il y a donc une claire différence entre les compétences nécessaires pour les deux métiers.

On constate que ce n'est pas le type de répertoire, mais son inaccessibilité qui est la raison principale de son absence chez les interprètes. Ce que nous avons remarqué ces derniers mois est le fait que proposer aux interprètes, professeurs d'instruments et élèves d'instrument un environnement de travail dans le contexte de RéDi-Musix (comprenant la base de données et le terminal Sampo), leur a permis un accès à ces musiques aussi simple qu'ouvrir une partition pour en faire une lecture.

Pour la suite du projet, nous prévoyons de stabiliser la base de données pour l'alimenter ensuite avec des transcriptions des pièces de répertoire existant et ainsi contribuer à la pérennisation de ce répertoire.

Concernant les prochaines étapes, nous prévoyons de définir un protocole de transcription d'œuvres mixtes pour Sampo et de le proposer aux partenaires intéressés par ce type de travail. En parallèle, nous allons élargir les possibilités du logiciel Sampo en y autorisant

[7] *Cinq reflets sur fond bleu* (2000), Mikhail Malt. Cinq pièces pédagogiques pour piano, hautbois, saxophone alto, violoncelle, violon et électronique sur support fixe. Création décembre 2000 à l'auditorium de Bagnolet. Et *Le sixième reflet* (2002), Pièce pédagogique pour batterie et électronique sur support fixe. Commandes du Conservatoire de Bagnolet.

[8] *Six contemplations* (2004), Mikhail Malt. Six pièces pédagogiques pour trompette, flûte, alto, guitare, clarinette et électronique sur support fixe. Commande du conservatoire de Bagnolet. Création avril 2005 à l'auditorium Pablo Neruda de Bagnolet, direction Wim Hoogewerf.

l'intégration de patchs Max. Ainsi, Sampo permettra de jouer les pièces avec fichiers son, les pièces avec accès gestuel, et enfin, les pièces avec déclenchements d'évènements.

## 9. REFERENCES


[1] Barthélemy, Jérôme ; Bonardi, Alain ; Orlarey, Yann ; Lemouton, Serge ; Ciavarella, Raffaele ; Barkati, Karim. (2010). "Toward An Organology Of Virtual Instruments" In *Computer Music*. 369-372.

[2] Bonardi, Alain (2015) « Rejouer une œuvre avec l'électronique temps réel : enjeux informatiques et musicologiques » *Genèses musicales*, Presses de l'Université Paris-Sorbonne, pp. 239-254.

[3] Bonardi, Alain (2013) « Copier/coller ou recopier ? La transmission entre artistes des œuvres musicales avec dispositif numérique » In *Technique et science informatiques* Volume 32 - n° 3-4.

[4] Bonardi, Alain ; Barthelemy, Jérôme (2008) "The preservation, emulation, migration, and virtualization of live electronics for performing arts: An overview of musical and technical issues", *Journal on Computing and Cultural Heritage*, 1, 1, p. 6:1-6:16

[5] Lemouton, Serge (2016). La préservation des œuvres musicales du répertoire de L'ircam : présentation du modèle Sidney et analyse des dispositifs temps réel. Computer Music Interpretation In Practice.

[6] Lemouton, Serge ; Ciavarella, Raffaele ; Bonardi, Alain (2009) "Peut-on envisager une organologie des traitements sonores temps réel, instruments virtuels de l'informatique musicale ?", Proceedings of the Fifth Conference on Interdisciplinary Musicology (CIM09), Paris.

# Posters

# MUSIC FROM AURA: MUSIC GENERATION USING EEG AND KIRLIAN PHOTOGRAPHY FOR TEMPORAL DIFFERENCE LEARNING

*Pierre-Henri Vulliard*
SCRIME/LaBRI
pierre-henri.vulliard@u-bordeaux.fr

**ABSTRACT**

Our aim is to fill the gap between computer music and musical emotions. We use a 2D emotional space (2D ES) both to analyze listener's emotions captured from physiological measure and to drive a music generator with a Reinforcement Learning algorithm. We present hereafter the Reinforcement Learning algorithm and the different parts of our framework used as input (emotions classification with Emotiv EEG headset and Kirlian photography device, to improve the machine learning by using data that have no correlation between them), and output of the algorithm (rhythm and scale/chords generation). Some current and future applications are also briefly depicted.

## 1. INTRODUCTION

*How to represent emotions both in music and in brainwaves analysis:* The most used model is the representation of emotions in 2D valence/arousal space, where valence represents the way one judges a situation, from unpleasant to pleasant and arousal expresses the degree of excitement felt by people, from calm to exciting [1].
*Emotions as a model for the interaction between brainwaves and music:* Numbers of experimentations have been pursued for the sonification of brainwaves. However, the fuzzy aspect of both music and emotions spaces makes traditional algorithms inefficient. Thus we choose for this experiment two more appropriate algorithm from Reinforcement Learning family, QLearning and Sarsa($\lambda$) [2].
One characteristic of our work is to include machine learning tools with a great symmetrical between its inputs (listener's emotions captation from his brainwaves analysis) and its outputs (music generation based on emotionnaly connoted parameters). From an algorithmic point of view, this symmetry has no particular meaning; but beyond a purely artistic approach, it should create conditions for a retroactive loop, it is widely admitted that music has an undisputable influence upon listener's mood and emotion, and numerous studies have adressed its effect [3, 4].

## 2. REINFORCEMENT LEARNING

### 2.1. Reinforcement Learning features

Reinforcement learning (RL) is learning by interactions with the environment. Its aim is, roughly speaking to map situations to actions in order to maximize a numerical reward signal. Clearly, a learning agent must be able to sense the state of the environment and to take actions that affect this state, and it also needs a goal to achieve. Temporal difference learning (TD) is a combination of Monte Carlo approach and Dynamic Programming. As for the Monte Carlo approach, TD can learn directly from the experiment (no need of a model of the environment dynamics), and as for the Dynamic Programming approach, TD updates its estimate based on prior estimation without waiting the end of the experiment and the final outcome.

### 2.2. Interest of Reinforcement Learning for music

*Bio-mimetism:* Reinforcement Learning methods take their insight from animal psychology studies, like those analy- sing how an animal can learn from trial and error to adapt to its environment, or those based upon language learning.
They are from the family human-like algorithms, and so seem well adapted to musical language learning.
*Ways emphasis vs single target:* One of the special features of the algorithm is that, in addition to the final target (tonic for tonal resolution), it also takes into account paths (harmonic walks).
*Early effectiveness:* Reinforcement learning differs from standard supervised learning in that correct input/output pairs are never presented, so it does not need prior calibration, and can be effective since early bars of music.
*Short or long term optimisation:* One of the properties of the RL approach is the trade-off between exploration (of uncharted territory) and exploitation (of current knowledge). This is a way of optimising for a fast result or for a long term optimised solution. *Poietic process:* Could help to describe musical rules directly from their attributes, for example using functional value of chords (dominant preparations, dissonances, resolutions) in mainstream tonal music based on tension/relaxation.

Pierre-Henri Vulliard, *Music from Aura (poster)*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 131–134, 2018. 

### 2.3. Reinforcement Learning in MuZICO

A module of Reinforcement Learning is implemented in MuZICO as a PureData interface using C++ functions

Following [2] a RL task can be expressed as a Markov Decision Problem (MDP) provided that we assume decisions and values to be functions of the current state only. At each discrete time $t$, the agent observes the environment state $s_t$ - in our case it is the projection of EEG signal of the listener into the 2DES. It selects an action $a_t$ - in our case the musical parameters driving the music generation. The action is performed and one step later (time $t+1$) the agent receives a reward $r_{t+1}$ and reaches a new state, the reward is the distance between the classification of the EEG signal projected in the 2DES space and the goal to reach. The last component is the *policy* $\pi_t$ where $\pi_t(s,a)$ is the probability that $a_t = a$ when $s_t = s$ which is used to update the estimates.

Choice for the Reinforcement Learning algorithm are: five parameters (temperature, epsilon ($\epsilon$), lambda ($\lambda$), earningRate ($\alpha$) and gamma ($\gamma$)), two possible policies (egreedy and softmax), two learning methods (qlearning and sarsa).
Temperature and $\epsilon$ are parameters for the policies, $\alpha$ is the learning rate used to update the expected reward of the states, $0 \leq \lambda \leq 1$ is the parameter which drives the number of states updated ($\lambda = 0$ the state updated is $s_t$ ; $\lambda = 1$ states $s_0, \ldots, s_t$ are updated). $0 \leq \gamma \leq 1$ is the discount factor used to compute the long run reward, if $\gamma = 0$, the agent maximizes the immediate rewards.

## 3. ACTIONS

### 3.1. Muzico

The actions are changes of the generative music settings to aim a target emotion. According to [5], the musical parameters related to the valence of emotions induced by music are tonality and complexity. We propose here harmonic, rhythmic and melodic generative models, as well as a description of their parameters from the point of view of musical complexity.
These parameters, as they are used in the MuZICO environment, have either discrete or continuous values, and are either defined by adjectives (fuzzy classes) or by numerical values. For example, the spectrum of a sound can take values from "muted" to "bright", corresponding to specific energy values that are measurable in the signal and are linked to its spectral centroid. The other parameters related to energy in music are: the tempo of the music, the attack of the notes' dynamic envelope, the pitch of the notes, the number of instruments playing simultaneously, the more or less "natural" sound of the instruments, the percussive or sustained aspect of sounds, as well as the number of simultaneous notes [5].
All these parameters are characterized by bounded intervals used by MuZICO to generate music. Those values, corresponding to low and high energies induced by the music, can be fuzzy values (such as muted, bright) or intervals for physical data. The parameters that have undefined units are translated by MuZICO into various units according to the algorithms used to produce the sounds (synthesis, sample playing, audio effect control). For instance one can translate the brightness of a sound by a particular setting of the modulation index in a FM synthesis algorithm, or by the control of the cutoff frequency of a low-pass filter applied to a sample player.

#### *3.1.1. Pitch scales*

Usually scales are generated by an integer as interval, following Eq. 1.

$$\begin{cases} U_{n+1} & = \left(U_n \times \omega^{\frac{V}{\theta}} - U_0\right) \times \omega^{-p} \\ U_0 & = f_f \end{cases} \quad (1)$$
$$with\ p \in \mathbb{N}^+ such\ that\ U_0 < U_{n+1} < \omega \times U_0$$

where $\omega$ is the octave ratio, $\theta$ the divider of the octave to obtain the temperament, $f_f$ is the root frequency of the generated scale, and $V$ the interval seed of the scale.

We consider that the complexity of a pitch scale relies on the number of iterations needed to generate it, and on the values of the different parameters $\omega, \theta,$ and $V$.
For example, taking $\omega = 2$, $\theta = 12$ (tonal occidental music). Let $V = (7)$ (generation of the pitch scale by iterating through steps of fifths), and N be the number of iterations. This gives: the pentatonic scale or the Maj 6/9 chord for N = 4, the diatonic scale for N = 6, the chromatic scale for N = 11
These scales have an increasing harmonic complexity.
Scales of different complexities can be generated by varying $V$: $V = (4)$ and $N = 2$ generate a pitch scale corresponding to an augmented chord, $V = (3)$ and $N = 3$ give a diminuated chord, $V = (3, 4, 4, 3)$ and $N = 6$ give the ascending melodic minor scale.

#### *3.1.2. Rhythmic patterns*

To keep a human-like style in beat generation, we model our inspiration upon the rhythms accompanying work songs during community repetitive labour, as activities in the field (threshing and winnowing) or domestic work (the pestle pounding the mortar). Each participant produces one single regular beat but the collective result is a more complex rhythm.

The rhythmic patterns are created by layering various iterative rhythmic patterns, all synchronized to the same underlying pulse, each pattern being defined by an offset from the common initial pulse and by a rhythmic density (number of pulses between two onsets in the pattern ). The superimposition is done by a logical OR operation, considering the superimposed patterns as bit vectors, a value of 1 representing an onset, and a value of 0 no onset:
density = 0 offset = 0 → 1 1 1 1 1 1 1 1
density = 0 offset = 2 → 0 0 1 1 1 1 1 1
density = 1 offset = 2 → 0 0 1 0 1 0 1 0
density = 3 offset = 0 + density = 5 offset = 3
1 0 0 0 1 0 0 0 + 0 0 0 1 0 0 0 0 = 1 0 0 1 1 0 0 0

For the rhythmic complexity, we use a model based on Toussaint's complexity measure [6] or other techniques of rhythmic complexity evaluation, that assign a weight to each pulsation.

### 3.2. Generative attributed grammar

*Chords series:* In MuZICO, the chords sequences generation takes the context into account to oversee the transpositions related to the modulations. The complexity of a chord depends on several factors that we identified:
the number of notes that make it up, the complexity of the scale on which it is built, the order of appearance of its notes in the building of this scale by iterations.
The complexity of a chords sequence also depends on several factors: the complexity of the chords that make it up, the number of different chords in the sequence, the harmonic complexity of the chords sequences, mainly taking into account the cadences and resolutions in the context of modern modal and tonal music, and the modulations in the context of the latter.

## 4. REWARD

Reward is calculated from Arousal and Valence axis of emotions, according to the Two-Dimensional Emotion Space (2DES) [1]. It is the distance between the current projection of the classification of the EEG to the final state at aim in the 2DES.

### 4.1. Emotions captation system

*Hardware:* As the title of the poster indicates, we use different sources for data acquisition: EEG headsets, connected watches, G.D.V (kirlian photography). Since the data is digital, the most scientific data (EEG, pulse) should make it possible to evaluate the relevance of the more fanciful data (aura, meditation analysis by Emotiv helmet).
For EEG, we use an Emotiv headset for its convenience and its growing dissemination. We acquire data from the headset with the Emotiv C++ SDK, and transmit them via UDP protocol for further use.

*Software:* OpenViBE software[7], designed for Brain Computer Interface work, provides all the necessary tools we need for our experiment setup.

### 4.2. EEG Data Analysis

*Filters:* First level is managed by the Emotiv headset itself. By request to the C++ api, we receive two types of data from the sensors: on one hand, raw data on the fourteen channels, on the other hand, filtered face muscles information (EMG), i.e. bottom or upper face tensions.
In OpenViBE software. EEG data channels are sorted and a first stage of band filters is applied, according to traditionnal brainwaves categories. Then comes the Common Spacial Pattern (CSP) filter stage, to improve the classification [8]. A CSP filter of dimension four is used for each band of brainwaves.

## 5. EXAMPLES OF APPLICATIONS

### 5.1. Short term convergence: Live Music Performing

*Saxophone and generative music:* Interactive music is used in real time in a saxophone improvisation show, (sometimes with dance and video projections).

### 5.2. Long term convergence: Relaxation, Music Therapy

Aware of the highly controversial statements coming from the alternative medicine community about audio stimulations which may help to induce relaxation, meditation, creativity and other desirable mental states, we implemented, nevertheless, some of these stimulations and synchronized them with the music beat.

The main clock of MuZICO calculates equivalences of tempo in BPM, duration of 16th notes, and difference in Hz for binaural beats. For instance, if bpm=120, the 16th note duration is 0.125 second, the isochronic sounds have a frequency of 8 HZ, and the left/right binaural beats channels are respectively raised and lowered from 8/2 = 4Hz.

Some of the mostly used audio processes:
*ASMR (Autonomous Sensory Meridian Response):* Rhythmic noises and sounds used to provoke a pleasant sensation of tingling or chills in the skull or scalp, or as a sleep aid.
*Mindmachines:* Audio impulsions are generated on an independant stereo output to control flashing leds glasses to make a "brainwave synthesizer" (or "mindmachine").
*Binaural beats:* Binaural beats, or binaural tones, are auditory processing artifacts, or apparent sounds, produced by a slight difference in the frequencies percieved by each ear of the same tone.
*Isochronic sounds:* Isochronic tones are evenly spaced tones, quickly turned on & off;

## 6. CONCLUSION - FUTURE WORKS

In this poster we presented the MuZICO framework and the core learning algorithm as an attempt to fill the gap between computer music and musical emotions.
Preliminary results have shown that a musical construction can be deduced from EEG analysis and Reinforcement Learning. The difficulty which remains is to assert that the state reached in the 2DES is effectively the intended real emotional state.
Exploratory experiments of various scenarii evaluating the different parameters of the MuZICO framework (LDA, CSP, TD on-policy, TD off-policy, long run rewards such as maximum valence and maximum arousal for music shows, or maximum valence and minimum arousal for relaxation) are undergoing.
The operation of the algorithm requires a large amount of data, so we created a smartphone app called Personal Hypno Vibes for listening to music and to acquire physiological data in various use cases (meditation, self-hypnosis, tests on the aura).

## 7. REFERENCES


[1] E. Schubert, "Measuring emotion continuously: validity and reliability of the two dimensional emotion space." *Australian Journal of Psychology*, vol. 51, no. 3, pp. 154–165, 1999.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 1998. [Online]. Available: webdocs.cs.ualberta.ca/~sutton/book/the-book.html

[3] L. B. Meyer, *Emotion and Meaning in Music.* University of Chicago Press, 1956.

[4] M. M. Bradley and P. J. Lang, "Affective reactions to acoustic stimuli." *Psychophysiology*, vol. 37, pp. 204–215, 2000.

[5] S. Livingstone, R. Muhlberger, A. Brown, and W. Thompson, "Changing musical emotion: A computational rule system for modifying score and performance." *Computer Music Journal*, vol. 34, no. 1, pp. 41–64, 2010.

[6] G. Toussaint, "Computational geometric aspects of rhythm, melody, and voice-leading." *Computational Geometry*, vol. 43, no. 1, pp. 2–22, jan 2010.

[7] Y. Renard, F. Lotte, G. Gibert, M. Congedo, E. Maby, V. Delannoy, O. Bertrand, and A. Lécuyer, "Openvibe: an open-source software platform to design, test, and use brain-computer interfaces in real and virtual environments." *Presence: teleoperators and virtual environments*, vol. 19, no. 1, pp. 35–53, 2010.

[8] F. Lotte and C. Guan, "Regularizing common spatial patterns to improve bci designs: unified theory and new algorithms." *Biomedical Engineering, IEEE Transactions*, vol. 58, no. 2, pp. 355–362, 2011.

# LA THEORIE DE LA GRAVITATION TONALE

*Emanuele Di Mauro*
emanuele.dimauro@gmail.com

## RÉSUMÉ

La théorie de la gravitation tonale a comme objectif premier d'identifier les liens entre l'univers de l'harmonie (aspect vertical de la musique) et celui de la mélodie (aspect horizontal).

Ces liens sont obtenus grâce à l'application de trois principes mathématiques qui constituent la base du calcul algorithmique et combinatoire exploité par le logiciel en ligne `www.gravitation-tonale.fr`.

L'idée principale de la théorie est d'exclure de l'univers musical un son particulier, disharmonique et cacophonique. Ce son, appelé ***condensation tonale***, est constitué par la superposition de trois notes contiguës de la gamme chromatique.

L'étude faite par l'auteur montre que cette superposition est le seul son à 3 notes distinctes qui n'est jamais présent dans le corpus des accords connus, à la différence de n'importe quelle autre superposition de trois notes de la gamme chromatique pour laquelle on trouve toujours au moins un accord qui la contient.

## 1. LES TROIS PRINCIPES

L'exclusion de la condensation tonale est, avant tout, un principe harmonique. On considère l'harmonie non pas à travers des règles relationnelles entre les notes, mais en définissant une limite extrême, une frontière au-delà de laquelle le son ne peut plus être considéré harmonieux.

L'extension de ce ***principe d'exclusion*** à l'aspect horizontal de la musique donne lieu à un corpus de gammes dites « verticalisables ». Cela permet d'identifier tous les liens accords/gammes avec un simple ***principe d'inclusion*** : la gamme verticalisable doit contenir toutes les notes de l'accord avec lequel elle se lie.

Un troisième ***principe de complétude*** est appliqué à l'objet « gamme » : une gamme verticalisable est complète seulement si aucune note peut être ajoutée sans produire une condensation tonale[1]. Ce principe permet de définir une gamme sans aucune ambiguïté. N'importe quelle autre gamme non verticalisable peut toujours être obtenue par combinaison de deux ou plusieurs gammes verticalisables.

[1]Par exemple la gamme de Do majeur est une gamme complète, car aucune autre note ne peut être ajouté sans produire une condensation tonale

## 2. L'UNIVERS MELODIQUE PRIMAIRE

Si on considère le principe d'exclusion et le principe de complétude on peut facilement calculer toutes les gammes complètes possibles. L'analyse de ces résultats et l'association avec les gammes connues a conduit à identifier et classer ces gammes comme suit :

Gamme majeure ou mineure «naturelle»
Gamme mineure mélodique
Gamme mineure harmonique
Gamme majeure harmonique
Gammes par ton
Gammes par ton, demi-ton
Gamme par demi-ton / tierce mineure

Ce corpus de gammes est appelé par la théorie « **univers mélodique primaire** ». Le principe d'exclusion de la condensation tonale peut alors s'appliquer à l'ensemble de tous les accords et aux gammes de l'univers mélodique dans un total de 57 transpositions (12 par gamme majeure ou mineure, 2, 3 ou 4 pour celles à transpositions limitées)

## 3. LA GRAVITATION TONALE

Dans une approche mathématique on recherche le moyen d'un processus de segmentation, où les accords sont groupés d'une façon telle que chaque groupe puisse exprimer un sens musical propre. Segmenter `n` accords veut dire trouver `m` groupes d'accords (**segments**), chacun constitué par un ou plusieurs accords. Le passage d'un groupe à l'autre doit exprimer un changement d'univers musical.

La question est alors d'identifier un processus capable de lier un segment à une ou plusieurs gammes et de déterminer les conditions nécessaires afin que cette association soit possible.

### 3.1 Définition

La segmentation d'une suite d'accords en segments conduit à la notion de « **gravitation tonale** ». Chaque groupe d'accords ainsi constitué est capable de générer des gammes verticalisables. Une gravitation tonale est donc identifiée par un centre de gravité (un ou plusieurs accords) et par toutes les gammes que ce centre de gravité peut produire. Chaque gamme doit contenir toutes les notes du centre de gravité qui l'a générée (principe d'inclusion). L'ensemble de toutes les notes du centre de gravité ne doit pas, alors, contenir de condensation tonale, car, pour le principe d'exclusion, aucune gamme verticalisable ne la contient. Une gravitation tonale ne

Emanuele Di Mauro, *La théorie de la gravitation tonale (poster)*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 135–138, 2018. 

peut donc exister que si aucune condensation tonale n'est présente dans l'ensemble de toutes les notes des accords qui constituent le centre de gravité.

### 3.2 Exemples

Les exemples suivants montrent les gravitations tonales pour la suite d'accords G | Am (*Figure 1*) et pour Dm | G7 (*Figure 2*):

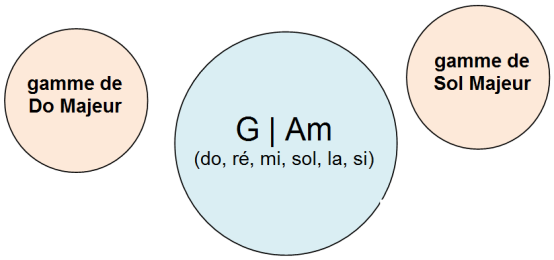



**Figure 1**. Gravitation tonale pour les accords *G | Am*

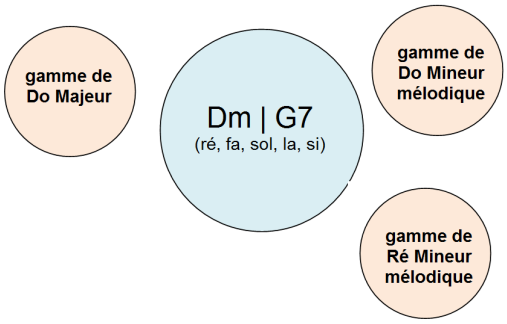



**Figure 2**. Gravitation tonale pour les accords *Dm | G7*

## 4. LA SEGMENTATION

Un processus de segmentation est sensé fournir toutes les façons possibles de grouper les accords afin que chaque groupe d'accords puisse générer une gravitation tonale. Cette problématique aboutit, en général, à plusieurs solutions.

### 4.1 La segmentation atomique

Une première réflexion porte à dire qu'on peut toujours segmenter les accords dans des groupes ayant chacun un seul accord, et réaliser ce qu'on appelle une **segmentation atomique**. Chaque accord aura alors sa propre gravitation tonale et cela est toujours possible car aucun accord ne contient une condensation tonale.

### 4.2 Groupement des accords

La découverte des segmentations possibles découle de la segmentation atomique. Pour obtenir toutes les segmentations le processus est celui de grouper, si possible, les gravitations tonales adjacentes. Cela peut être fait si l'ensemble des notes des deux gravitations adjacentes ne contient pas de condensation tonale. Dans l'exemple en figure 3 on applique cet approche à la suite d'accords : `Am | A7 | Dm | G7 | C`

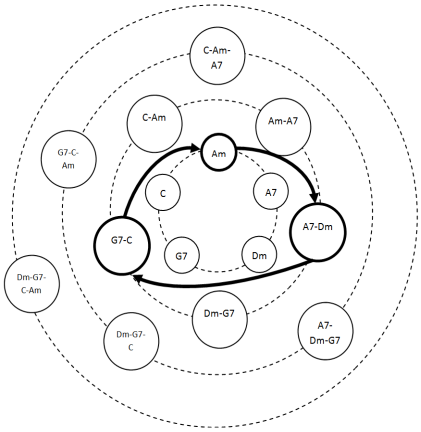



**Figure 3**. Possible segmentation `[Am][A7|Dm][G7|C]` pour la suite d'accords `Am | A7 | Dm | G7 | C`

## 5. LE CHOIX DES GAMMES

### 5.1 Problématique

Le processus de segmentation ne donne pas, en soi, d'informations sur les gammes utilisables pour chaque gravitation tonale. En général on peut imaginer une suite de $m$ gravitations tonales comme une suite d'ensembles de gammes. Dans une logique purement combinatoire on obtient alors un nombre $N$ de combinaisons de gammes qui est le produit du nombres de gammes $n_1, \ldots n_m$ associées à chaque gravitation tonale :

$$N = n_1 * n_2 * \ldots * n_m$$

Une suite de 10 accords majeurs ou mineurs, par exemple, fournit plus de 60 milliards de combinaisons de gammes (chaque accord, mineur ou majeur est lié à 12 gammes verticalisables) :

$$N = 12^{10} = 61.917.364.224$$

Le processus de segmentation (*Figure 4*) est alors essentiel pour réduire considérablement le nombre de combinaisons des gammes impliquées. En effet la création d'une gravitation tonale entre 2 ou plusieurs accords fait augmenter les notes du centre de gravité, et cela réduit d'une façon considérable les gammes associées.

### 5.2 Le choix des gammes

Quoi qu'il en soit, le problème du choix des gammes s'impose à partir du moment où plusieurs gravitations tonales existent et s'enchaînent. On se doit de choisir la gamme qui sera utilisée dans chaque gravitation tonale. Or, le choix d'une gamme, pour une gravitation tonale donnée, ne peut pas se faire dans l'absolu, et il doit tenir compte des gammes liées aux autres gravitations tonales adjacentes. Pour cela, on exploite une notion de *similitude entre gammes* afin de choisir la combinaison des gammes qui rend minimale cette distance.

### 5.3 La similitude entre gammes

La similitude entre gammes doit donc faire l'objet d'un calcul mathématique aux caractéristiques suivantes :

- La comparaison entre une gamme `g1` et une autre gamme `g2≠g1` donnera lieu à un score qui ne sera pas maximal.
- La comparaison entre n'importe quelle gamme et elle-même (auto-similitude) devra avoir une valeur maximale et normalisée.

Ce deuxième point est important. N'importe quelle gamme doit avoir une auto-similitude normalisée.

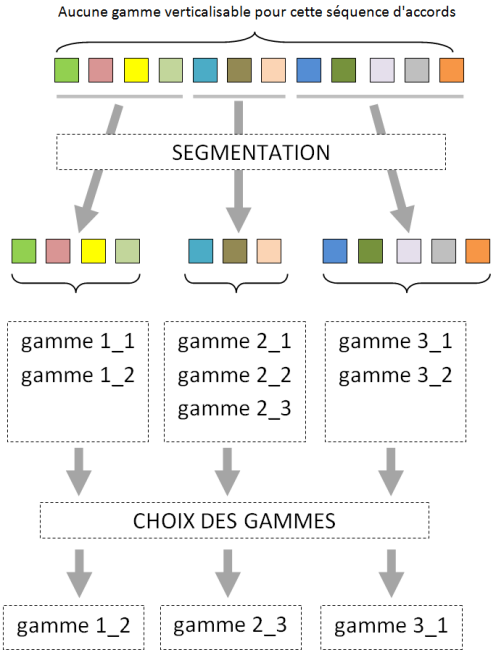



**Figure 4**. Choix des gammes pour une segmentation donnée.

### 5.4 Algorithme de similitude entre gammes

L'algorithme proposé par la théorie (qui n'a pas de valeur absolue, il s'agit d'une proposition ouverte) est alors le suivant :

```
-> On considère deux gammes à comparer
-> On part d'un indice de similitude de 0
-> Pour chacune des 12 notes chromatiques (do,
do#, ré…) on vérifie si la note est présente
dans les deux gammes

   -> Si la note est présente dans les deux
   gammes l'indice de similitude est incrémenté
   d'une valeur 1.

   -> Si la note est absente dans les deux
   gammes l'indice de similitude est incrémenté
   d'une valeur 1.

   -> Si la note est présente dans une gamme,
   mais pas dans l'autre l'indice de similitude
   n'est pas incrémenté.
```

Le score obtenu va de 0 à 12. La valeur 12 est toujours le score d'auto-similitude. Ce score nous permet alors de trier les résultats combinatoires par similitude, ou par moyenne de similitude, et éventuellement par variance de similitude. En triant par ordre décroissant, on obtient en premier lieu les résultats qui expriment les transitions les plus douces en passant d'une gravitation tonale à l'autre.

## 6. LOGICIEL EN LIGNE

### 6.1 Le logiciel *gravitation-tonale*

Le logiciel en lige `gravitation-tonale.fr` exploite les résultats de la théorie. Quatre outils sont mis à disposition, selon l'objectif visé :

***Objectif 1*** : Déterminer à partir d'un accord, toutes les gammes possibles verticalisables liées.
***Objectif 2*** : Déterminer, à partir d'une suite d'accords, si possible, toute les gammes liées.
***Objectif 3*** : Obtenir à partir d'une suite d'accords toutes les segmentations possibles.
***Objectif 4*** : Pour une segmentation donnée, obtenue en objectif 3, le logiciel propose toutes les combinaisons des gammes possibles, triées en ordre décroissant de similitude (moyenne et variance).

## 7. CONCLUSIONS

Sur des séquences d'accords issues d'un univers musical où on peut reconnaître une notion de tonalité globale ou locale, les liens accords/gammes obtenus en appliquant les trois principes de la théorie paraissent cohérents avec la vérité du terrain. Le logiciel suggère, par exemple, la bonne gamme diatonique, pour des séquences II-V, ou II-V-I, même en présence de modulations et de changements de tonalité.

Sur des séquences d'accords inhabituelles, où la notion de tonalité parait insaisissable, l'application des trois principes de la théorie conduit à des liens accords/gammes difficilement imaginables sans faire recours au calcul assisté par ordinateur. L'explosion combinatoire des résultats est fortement limitée par la segmentation automatique et par l'identification des gravitations tonales, chacune réduisant fortement le nombre des gammes impliquées.

Le logiciel *gravitation-tonale,* pourrait en premier lieu, être un support intéressant pour l'exploration de nouveaux univers musicaux, sur des séquences d'accords complexes. Mais il peut être aussi un support pédagogique valide sur des séquences classiques (sous-dominante, dominante-tonique, cycle des quintes, modulations et changements de tonalités locales...).

## 8. REMERCIEMENTS

Je souhaite adresser mes remerciements les plus sincères au professeur émérite Nicolas Meeus (Université Paris-Sorbonne) pour sa disponibilité, son écoute, sa générosité, ses enseignements et ses conseils tout au long de ce travail de recherche.

## 9. RESSOURCES

[1] Logiciel en ligne : http://www.gravitation-tonale.fr
[2] Aspects algorithmiques :
http://www.gravitation-tonale.fr/algos.pdf

# POLYPHONIC SINGING PRESENTED AS A MULTIPLAYER CLASSROOM ONLINE GAME

*Jonathan Bell*
Aix Marseille Univ, CNRS, PRISM
« Perception, Representations, Image, Sound, Music »
Marseille, France.
jonathan.BELL@univ-amu.fr

**ABSTRACT**

This paper proposes to use web technologies in a classroom environment, in order to help singers and/or instrumentalists of all levels read and perform polyphonic music. Recent tests in conservatoires and universities have proved that networked music performances of this sort can foster interest in music reading among various groups of music students. The application presented here, *SmartVox*, is hosted on a server, either run locally or via the internet. The *server* provides a web page accessible by the *client* through an IP address (*e.g.* **smartkids.smartvox.eu**) [1], which the students type into the browser of their smartphone. Students typically access the application together, in the same classroom - but each with his/her own device - and request the individual parts of the polyphony corresponding to their tessitura. The conductor's interface, a 2nd type of *client* (**smartkids.smartvox.eu/conductor**), controls the global state of the distributed application (*e.g.* play, pause...). The coupling of auditory and visual signals, as well as the ability to render and synchronize different parts on each performer's device makes *solfège* and various forms of elaborate contrapuntal musical situations accessible to untrained musicians.

## 1. INTRODUCTION

This article presents the continuation of the research published in *SmartVox, a web-based distributed media player as notation tool for choral practices* [3]. Based upon a doctoral thesis investigating *audio-scores* [2], *SmartVox* was developed at IRCAM in Paris in 2015, by Benjamin Matuszewski and the author.

[1] Click on this link or type-in **smartkids.smartvox.eu** directly in the address bar of a browser connected to the internet. On the homepage, click on the screen, choose one part, and press the *send* button. Once the part loaded, the video needs to be unlocked, pressing the *play* button of the device's media player.

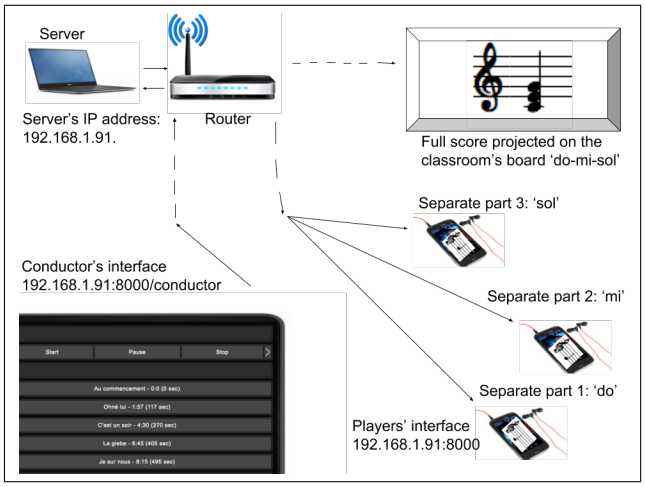



**Figure 1**. *SmartVox* running on a *local area network*

### 1.1. Technical presentation

*Smartvox* is open source [2], it was developed within the Javascript *Soundworks* [9] framework, running on node.js (server-side javascript). The single-page web application consists of two web clients: the *player* (see Figure 1, right-hand side) and the *conductor* (Figure 1, bottom left), that can be executed in any recent web browser on mobile devices (smartphones, tablets and laptops). The real-time communication between clients is achieved through the *WebSocket* protocol [3]. The scores are composed and/or transcribed in the *Bach* [1] environment for Max/MSP.

### 1.2. Nature and function of the application

*Smartvox* serves multimedia (audio + visual) scores in the mp4 format. Once the players have chosen their part, the application synchronizes all the videos over a shared timeline. Figure 1 shows a typical *Local Area Network* classroom setup, where no access to the internet is required, and where communication is realized through a router that generates a wifi network. A computer runs the node.js

[2] The code can be dowloaded at the following address: https://github.com/belljonathan50/SmartVox0.1
[3] https://www.w3.org/TR/websockets/

Jonathan Bell, *Polyphonic Singing presented as a Multiplayer Classroom Online Game (poster)*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 139–142, 2018. 

| Pieces | Player's address | Conductor's address |
|---|---|---|
| 1. And the sea | smartvox.eu | smartvox.eu/conductor |
| 2. Nuages | nuages.smartvox.eu | nuages.smartvox.eu/conductor |
| 3. Smartkids | smartkids.smartvox.eu | smartkids.smartvox.eu/conductor |
| 4. Josquin | josquin.smartvox.eu | josquin.smartvox.eu/conductor |
| 5. Vitry | avignon.smartvox.eu | avignon.smartvox.eu/conductor |
| 6. Tallis | tallis.smartvox.eu | tallis.smartvox.eu/conductor |
| 7. Dunstaple | dunstaple.smartvox.eu | dunstaple.smartvox.eu/conductor |
| 8. Dufay | dufay.smartvox.eu | dufay.smartvox.eu/conductor |
| 9. Canons | canons.smartvox.eu | canons.smartvox.eu/conductor |

**Table 1**. Pieces currently available on the web. The links can be copied into a browser or directly clicked on this page.

server (see the top left of the figure), which can then be accessed by all the players by typing the server's IP address into their phone's browser. Figure 1 also highlights the polyphonic nature of the application: three players each receive different pitches (*do*, *mi*, *sol*), and the resulting chord is displayed on the classroom board.

## 2. CASE STUDIES

Since most smartphones can today run reliable browsers (such as Chrome, Safari or Firefox), this form of web-based distributed application is an extremely simple and cost-effective way to connect and synchronize all the devices present in the same room (or on the world-wide web, but it is less relevant in this particular case). *SmartVox* was originally designed for the rehearsal and performance of my own compositions. The cues provided by this application allow singers of all levels to perform long and sometimes challenging pieces in public concerts, wearing headphones and watching their score scrolling on the screen, thus highly improving their confidence and the quality of their performance. This participative aspect also allows collaboration between professional and amateur ensembles (see Section 2.3). In the meantime, since its conception in 2015, *SmartVox* quickly found a wide range of applications in pedagogical environments, from *solfège* or *formation musicale* in conservatoires (see Section 2.1 and Table 1, piece n°3: smartkids), to more advanced courses about Renaissance polyphony for 2nd year university students in musicology (see Section 2.2 and Table 1, pieces n°4-9).

### 2.1. *Solfège* in conservatoires: smartkids.smartvox.eu

The *SmartKids* piece was tested in several French conservatoires [4], with children of different ages/grades, in instrumental, mixed, and *a capella* versions. For children, a distributed mobile application can be evocative of a multiplayer game. This playful aspect helps to focus their attention on the demanding notions of music theory or *solfège* (see Figure 2). In a pedagogical piece composed for this system, the notation purposefully conveyed the same pitch information in four different ways:

[4] Fontenay-aux-Roses, CRI. Sussy-en-Brie, CRI. Conservatoire du 5ème arrondissement, Paris. Vitry-sur-Seine, CRD.

**Figure 2**. A *SmartKids* reading session during a French conservatoire *formation musicale* lesson.

- Sound frequency: a synthetic voice sings on a given pitch, *e.g.*, A = 440 Hz (audio).
- Spoken words: the synthetic voice pronounces the corresponding phoneme '*La*' (audio).
- Symbolic notation: the corresponding pitch is displayed on the musical stave (visual).
- Written text: the phoneme '*La*' is written below the stave (visual).

The pitch content of the score was sight-readable and self-explanatory. The notation of time did not require any specific knowledge either, since it displayed a scrolling playhead in proportional notation [5], thus bypassing the conventional 'bars and beat' rhythmic notation. At first sight, groups of ten to twenty children were able to sing in tune complex three-parts polyphonies. [6] After a few run-throughs, the children had memorized elements of the polyphony, and the piece could be used as source material for written melodic dictations. These classroom experiments also seemed to prompt students to propose their own musical ideas, a process later leading to forms of collaborative composition. This potential creative output associated with the use of information and communication technology (ICT) resonates with studies by Reynolds [8] and Pitts-Kwami [7], which highlights a correlation between compositional skills and the use of ITC in primary and secondary schools.

[5] The scores were realized in *bach.roll* [1]. This object allows a refined control of synthesizers (i.e. vocal synthesizers), controlled via the metadata contained in each note of the object, and subsequently sent to Max in realtime when played, *e.g.* : https://youtu.be/EcsdBrQf6Jw

[6] Live recording of a *SmartKids* reading session: https://youtu.be/syBZ3D8Pnjo

### 2.2. *Renaissance Polyphony* at AMU: tallis.smartvox.eu

The pieces n°4-9 of Table 1 were used extensively at Aix-Marseille University with 2nd year students in '*musique et musicologie*' (Arts Department). In this course on Renaissance polyphony, *SmartVox* was used on a weekly basis in order to let the students sight-sing the pieces that were subsequently analyzed. This practice-based approach to musical analysis allowed the students to engage personally with pieces of extremely complex construction (such as Ockeghem's *Missa Prolationum* or Mouton's *Nescient Mater*, both available at www.canons.smartvox.eu). From a pedagogical point of view, having the full score projected on the classroom's board proved to be very instructive [7] : at any time in the piece, each student could see his own part on his phone, hear through headphones a piano sound prompting him the pitches he is asked to sing, with the resulting polyphony sung by his colleagues, and displayed on the board. With this exhaustive representation, the identification of imitations between voices, canons and cadences became very clear on the score, and was at the same time directly related to a practical musical experience. At the end of the semester, the web application was used again with the students in a church, in order to record in a relevant acoustic the pieces studied during this course. [8] This form of computer-aided performance allowed us to superimpose the different takes of the recording, all perfectly in tune and in time with each other, thus making possible various cuts and edits. [9]

### 2.3. Productions

As well as a learning-aid used in a teaching environment, *SmartVox* is also used in diverse participative compositional experiments, which will now be discussed. These participative projects emerged from the observation that choral singing requires competencies such as vocal skills, intonation, music reading, and confidence in performance situation. This often restricts ancient and contemporary repertoire to a small group of specialists. The audio and visual guides provided by therefore seek to offer accessibility and exposure to works otherwise judged too difficult, for choirs of all levels.

#### *2.3.1. SmartVox, the piece*

The SmartVox piece/production [10] , for 5 soloists, choir, and tape, was premiered in Nantes in March 2017 [11] . Involving a local choir each time, this project has a participative aspect that makes it financially viable. SmartVox was therefore performed again in Rouen in April 2018. The use of audiovisual notation for this piece was justified by its microtonal language, because of the confusion that the notation of such intervals may cause for some singers.

#### *2.3.2. Le temps des nuages* [12]

This piece (see Table 1, piece n°2), based on poems by French philosopher Michel Onfray, premiered in January 2018, used *SmartVox* on a very large scale for the first time. This piece was written for 5 singers (the *De Caelis* ensemble), 5 percussionists (the *Links* ensemble), 4 channels of electronics, and 75 junior high-school students, who were placed all around the audience. The setup required about 70 simultaneous connections. The size of the concert hall (600 seats) and the number of connexions required a powerful wifi network relying on several antennas placed where the singers stood. The recording of the piece (see footnote below) clearly manifests, in terms of pitch and timing, but also hearing the children's conviction and confidence, that such a degree of accuracy could not have been achieved in any other way.

## 3. CONCLUSION

Music technology is still hardly ever used at school. This reveals, according to Hitchcock: "the dichotomy between the ubiquity of music technology in the music world and technology's relative paucity in the school curriculum" [6]. Furthermore, smartphones are often considered as a parasite in the classroom environment: "most school's policies completely forbid mobile usage within lessons" [4]. Bridges between technology and music education, therefore, are often judged with mixed feelings, and yet, with digital-native generations, innovative solutions must be found to accompany a mutation which Georg Hajdu describes as a "slow but steady shift away from textualization in digital media" [5]. With such mutation, the classical score-based music-making apparatus, hitherto reserved to an elite, could be shared with a larger community. According to Wise, Greenwood and Davis, the challenge consists in "moving technology from its position as an *add-on* in the music curriculum to a position of being embedded within the curriculum" [10].

The originality of the present *setup* (the French term *dispositif* is here of particular relevance) consists in turning mobile devices into a learning-aid fostering collective music singing improvement. With its strong focus on music reading, it hopes to convey among young students a revival of interest in music notation and thus benefit to the school curriculum. It constitutes an original contribution to the difficult questions raised by the use of technology in music education. Whether in conservatoires [13] or uni-

[7] The computer plugged to the beamer just connects to the server like the performers (unless it is running the server itself, in which case the address is, for instance, localhost:8000) and displays the *conducteur* part.

[8] Recording by the students with the help of *SmartVox*: https://www.youtube.com/watch?v=bofWvTCNNKI

[9] Editing process exemplified in *Reaper*: https://youtu.be/HExNA5Z7eFY?t=34

[10] Brief description of the project: http://www.decaelis.fr/fr/programmes/534

[11] Live recording with animated full score of the piece: https://www.youtube.com/watch?v=8R4Twc1A7Ks

[12] Live recording of the piece: https://youtu.be/SyFdR2HiF00

[13] Live recording of a *SmartKids* reading session: https://youtu.be/syBZ3D8Pnjo

versities [14] , the use of *SmartVox* proved each time to be an invaluable tool for helping unexperienced music readers, thus disinclining non-readers to switch off, and putting 'on the same track' singers of different musical backgrounds.

## 4. REFERENCES


[1] A. Agostini and D. Ghisi. Bach: An environment for computer-aided composition in max. Ljubljana, Slovenia, 01 2012.

[2] J. Bell. Audio-scores, a resource for composition and computer-aided performance http://openaccess.city.ac.uk/17285/. 2016.

[3] J. Bell and B. Matuszewski. Smartvox. a web-based distributed media player as notation tool for choral practices. In H. L. PALMA, M. SOLOMON, E. TUCCI, and C. LAGE, editors, *Proceedings of the International Conference on Technologies for Music Notation and Representation - TENOR2017*, pages 99–104, A Coruña, Spain, 2017. Universidade da Coruña.

[4] M. Gall. *Technology in Music Education in England and across Europe*. Oxford University Press, aug 2017.

[5] G. Hajdu. Disposable music. *Computer Music Journal*, 40(1):25–34, 2016.

[6] M. Hitchcock. *Generating Intersections between Music and Technology*. Oxford University Press, aug 2017.

[7] A. Pitts and R. M. Kwami. Raising students' performance in music composition through the use of information and communications technology (ict): a survey of secondary schools in england. *British Journal of Music Education*, 19(1):61–71, 2002.

[8] N. Reynolds. *Technology and Computers in Music and Music Education*, pages 333–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[9] N. Schnell and S. Robaszkiewicz. Soundworks – A playground for artists and developers to create collaborative mobile web performances. In *'Proceedings of the Web Audio Conference (WAC'15)*, Paris, France, 2015.

[10] S. Wise, J. Greenwood, and N. Davis. Teachers' use of digital technology in secondary music education: illustrations of changing classrooms. *British Journal of Music Education*, 28(2):117–134, 2011.


[14] Recording by students of the Aix-Marseille University (AMU) music department: https://www.youtube.com/watch?v=bofWvTCNNKI

# LEARNING GEOMETRY AND MUSIC THROUGH COMPUTER-AIDED MUSIC ANALYSIS AND COMPOSITION: A PEDAGOGICAL APPROACH

*Sonia Cannas*
Università di Pavia and IRMA/Université de Strasbourg
sonia.cannas01@universitadipavia.it

**RÉSUMÉ**

In this paper we will focus on pedagogical approaches via HexaChord, a computer-aided music analysis environment based on spatial representation of musical objects inspired by the *Tonnetz* and other geometric models. We will report some past experiences with children and adolescents during two Festival of Science. We will also suggest future laboratories for children.

## 1. INTRODUCTION

The results of learning geometry through interactive geometry software (IGS) and dynamic geometry software (DGS) such as Cabri and GeoGebra are well known in mathematics education [7]. At the same time, the employment of the information technology (IT) in music education shows an increasing interest [1]. Could pedagogical approaches on learning music and geometry be developed together through the IT tools? The same abstract notions in different concrete settings are useful for the development of abstraction capabilities. Although there are studies and experiments on higher education [2], interdisciplinary pedagogical approaches among mathematics, music and computer science are often scarce in primary and secondary school. We will analyze some pedagogical aspects via HexaChord [3, 4], reporting experiences with children and adolescents, and we will suggest future pedagogical laboratories.

## 2. GEOMETRIC MODELS IN MUSIC

### 2.1. Musical graphs and simplicial complexes

Geometric structures such as graphs and simplicial complexes are music-analytical tools commonly used to visualize and describe parsimonious voice leading and relations among musical objects. In music theory two different kinds of graphs are used: note-based and chord-based graphs. In the first ones each vertex is labeled by a pitch-class, in the second ones, by contrast, each vertex is labeled by a chord. Chord-based graphs are associated with note-based graphs by duality.

The most famous example of note-based graph is the *Tonnetz*. It is a graph in which pitch classes of twelve-tone equal temperament are organized along intervals of fifth in the horizontal axis, major and minor thirds in the diagonal axis. In this construction each triplet of distinct vertices, which are adjacent two by two, is a triangle representing a major or a minor triad. We can also define it as an infinite 2-dimensional simplicial complex, where 0-simplices represent pitch classes, and 2-simplices identify major and minor triads. The dual graph of the Tonnetz, known as *Chicken-wire torus* [6], is a chord-based graph in which vertices represent major and minor triads.
The edge-flips in the Tonnetz and the edges in the Chicken-wire torus represent the neo-Riemannian musical operations $P$ (parallel), $R$ (relative) and $L$ (leading tone), commonly used to describe parsimonious voice leading [5].

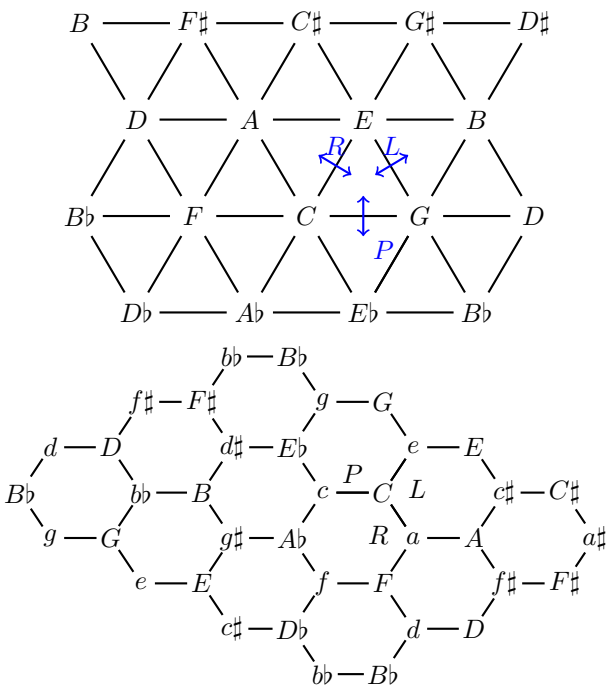



**Figure 1**. The Tonnetz (above) and the Chicken-wire Torus (below)

### 2.2. Circular geometric models in music

There are also two other well known geometric models in music: the *circle of fifths* and the *tone-clock*.
The first one is a circle in which the 12 major and minor scales, and their corresponding key signatures, are represented along a circle at a distance of fifth.

Sonia Cannas, *Learning Geometry and Music through Computer-aided Music Analysis and Composition : A Pedagogical Approach (poster)*, Journées d'Informatique Musicale (JIM 2018), Amiens, France, éd. L. Bigo, M. Giraud, R. Groult, F. Levé, pp. 143–146, 2018. 

The *tone-clock*, also known as *musical-clock*, is a circular representation of the 12 pitch-classes at a distance of a semitone. Each pitch class is labeled with a number from 0 (in the position of the 12 in the clock) to 11, starting from $C = 0$ (see fig.3). It allows to represent chords: a 2-chord is a segment, a 3-chord a triangle, a $n$-chord a polygon of $n$ vertices inscribed in a circle.

## 3. GEOMETRIC MODELS IN COMPUTATIONAL MUSIC ANALYSIS

There are several softwares in which musical graphs, simplicial complexes and other geometric models are integrated. One of them is HexaChord, a computer-aided music analysis environment based on spatial representation of musical objects, developed by Louis Bigo [3] [4].
The main interface is simple and easy to use, and it is represented in fig. 2. The first block is used to play music files in MIDI format. Below it, there is a second block used to visualize geometric models in music: the *Tonnetz* as a simplicial complex or other simplicial complex isomorphic to it (button "display/hide complex"), the musical-clock (button "circle 1"), the circle of fifths (button "circle 5"), and the voice-leading space (button "display graph"). During the execution of the MIDI file, it is possible to visualize in real time each chord in the different geometric models. Clicking on "trace on" it is also possible to visualize in the Tonnetz the chord sequence of the musical piece as a trajectory in the space.
In the part below the latter there is the block "Vertical compactness", to compute the compactness of the trajectory related to the musical piece in the *Tonnetz*. In the lower part there is the block "Trajectory Transformation", used to apply a geometric transformation to the original simplicial complex. One of them is the discrete translation, that musically corresponds to a transposition.

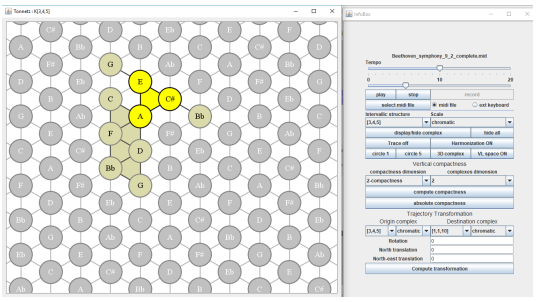


**Figure 2**. HexaChord

Moreover with Hexachord it is also possible to connect a piano keyboard and play/record musical pieces. All functionality are available also for the performance.

## 4. PEDAGOGICAL EXPERIENCES AND RESULTS

### 4.1. Festival of Science in Strasbourg

During the *Festival of Science* (2017) at the Vaisseau of Strasbourg, an IT-mathemusical laboratory for children has been organized. Different materials were proposed which included: a computer with HexaChord, a piano keyboard connected to it, loudspeakers, a musical-clock made with a wooden model (see fig.3), another wooden model representing both the *Tonnetz* and the *Chicken-wire Torus* (see fig.4). Many children played with the wooden musical-

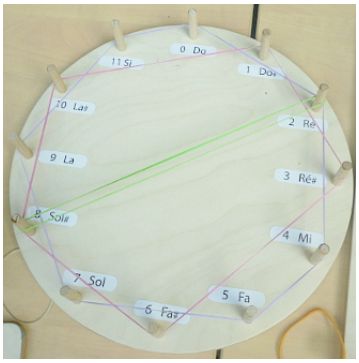



**Figure 3**. Wooden musical-clock. The chords are represented using a rubber band, and the twelve points are labeled with a number from 0 to 11 and the correspondent note

geometric models, more than the children who tried to play the piano and to understand HexaChord. Their attention focused more on wooden models for several reasons. First of all, since all children have already had gaming experiences with geometric shapes and rubber bands, they had no difficulties to recognize it as an entertaining game. On the contrary, the piano keyboard and HexaChord were new for most of them, and they require a first phase of comprehension in order to be used correctly. During the absence of a guide to the piano and the computer, some children tried to play with the piano looking the geometric models represented on HexaChord at the screen. Especially the younger ones did not understand how the musical sequences were displayed, then they moved on the geometric wooden models, more familiar to already known games. Moreover this underlines the different levels of learning musical and mathematical skills. The mathematical skills required in the construction of chords on geometric models are easier than the musical ones. Listening, playing, and understanding how the musical sequence is represented on the geometric model require a musical awareness that children begin to have towards around the age of 9. Furthermore, the children have to perform various tests for learning the latter, therefore it is necessary that they try one by one. On the other hand, activities on geometrical models can be carried out by several children at the same time. In the moments when a guide explained the representation of the musical sequences through Hex-

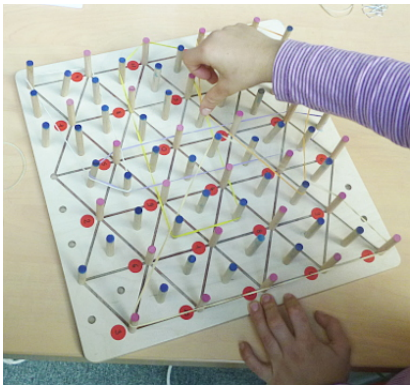


**Figure 4**. Wooden Tonnetz

aChord, the children showed interest and they understood very quickly its behavior. By testing them directly on the piano and observing the screen, their learning has become active. It is in fact well known that experiential learning is more efficient than passive learning such as reading or listening.

### 4.2. Festival of Science in Cagliari (Italy)

In the last edition of the *Festival of Science* in Cagliari (2017) I presented geometric models in music and their usefulness in music analysis through HexaChord. Most of the audience consisted of two classes of students belonging to the last year of a high school (19 years old) with a musical program. Thanks to their musical background, they easily understood the features of HexaChord and its utility for music analysis. During their musical studies they studied traditional music analysis, and their reactions to computational approaches to it based on geometric models were very positive. In addition to the surprise of knowing computer-aided music analysis, their greatest surprise was to discover that mathematics can also be a useful tool for music. Often, at school level, mathematics is taught and perceived as a set of many formulas and rules, whose meaning is unclear. Interdisciplinary learning can be an additional tool to overcome this misunderstanding of mathematics as arid subject without meaning. In conclusion, it was a very positive experience for the students: on the one hand they discovered a new type of musical analysis that brings together music, computer science and mathematics, on the other hand they discovered the beauty and utility of geometry.

## 5. FUTURE LABORATORIES FOR CHILDREN

The laboratories we are planning include some different activities, designed with different levels of difficultly. We want to propose game activities for children, in order to develop musical and geometric-spatial skills.
For all activities we need: a computer with which to use HexaChord, a piano keyboard connected to it, and a wooden musical-clock or a wooden Tonnetz. The aims of these activities are different. The children will become familiar with the musical-geometric models and the representation of the notes on them through HexaChord. They will learn the correspondence between number of notes of a chord and number of vertices of the correspondent figure on the musical-clock. They will start to become aware of the musical meaning of chord. Moreover they will observe that in the musical-clock and in the piano keyboard the notes are repeated in the same way, therefore they will become aware of the fact that one can work using only one octave. They will also begin to become familiar with the piano keyboard.

### 5.1. Reconstruction of chords

These laboratories are useful for children at the age of 9 and up.

#### *5.1.1. First activity*

The first activity consists in listening to a single chord on HexaChord, observing its geometric shape in the musical-clock integrated in the software and reproducing the latter in the wooden musical-clock with the rubber band. One can choose whether to prepare various MIDI files of individual chords to be played, or play them directly on the piano keyboard during the activity.
After that, children can do the contrary. In the wooden musical-clock a plane geometric figure is represented with the rubber band, the children have to reproduce it using the piano keyboard. The visualization on the musical-clock of HexaChord will help to this scope.

#### *5.1.2. Second activity*

The aim of the second activity is the same as the first. The children have to reproduce a chord in the wooden musical-clock but, this time, they will visualize the representation of the chord on the *Tonnetz*. To reach the aim, children can use the piano keyboard and observe their chord reproduced on the Tonnetz. This activity can be proposed if the children have already some musical background, otherwise it is recommended to carry out the first activity.

### 5.2. Musical sequences

This laboratory is useful for children at the age of 7 and up.
The aim is to recognize that a musical sequence in the *Tonnetz* and its transpositions have trajectories with the same shape. The children will listen a musical phrase (played in real time or previously recorded) and they will see the trajectory on the Tonnetz integrated in HexaChord. They will have to sing it and to represent the trajectory in the wooden Tonnetz. Afterwards, the musical phrase will be transposed, and the children will have to sing it and to represent the new trajectory. After some examples, the children will have to represent the new trajectory without displaying it on HexaChord and having only the initial note.

## 6. REFERENCES


[1] A. Anatrini, *The state of the art on the educational software tools for electroacoustic composition*, Proceedings of SMC Conference 2016, 2016.

[2] M. Andreatta, C. Agon Amado, T. Noll, E. Amiot, *Towards Pedagogability of Mathematical Music Theory: Algebraic Models and Tiling Problems in computer-aided composition*, Proceedings Bridges. Mathematical Connections in Art, Music and Science, London, 2006, p. 277-284.

[3] L. Bigo, M. Andreatta, J. L. Giavitto, O. Michel, A. Spicher, *Computation and Visualization of Musical Structures in Chord-Based Simplicial Complexes*. In: Yust J., Wild J., Burgoyne J.A. (eds) Mathematics and Computation in Music. MCM 2013. Lecture Notes in Computer Science, vol 7937. Springer, Berlin, Heidelberg, 2013.

[4] L. Bigo, D, Ghisi, A. Spicher, M. Andreatta, *Spatial transformations in simplicial chord spaces*. In: Proceedings of the Joint International Computer Music Conference - Sound and Music Computing, Athens, 2014, pp. 1112-1119.

[5] R. Cohn, *Neo-Riemannian Operations, Parsimonious Trichords, and their "Tonnetz" Representation*, Journal of Music Theory, 1997.

[6] J. Douthett, P. Steinbach, *Parsimonious Graphs: A Study in Parsimony, Contextual Transformation, and Modes of Limited Transposition*, Journal of Music Theory, 42/2, 1998.

[7] R. Stäßer, *Cabri-Geometre: Does Dynamic Geometry Software (DGS) Change Geometry and Its Teaching and Learning?*, International Journal of Computers for Mathematical Learning, 6(3), 2001, pp. 319-333.

# Auteurs


Bell, Jonathan, 139
Bevilacqua, Frédéric, 115
Bimbot, Frédéric, 5, 71, 81
Bonardi, Alain, 11, 105

Cannas, Sonia, 143
Carpentier, Thibaut, 35, 45
Chehab, Jean-Paul, 3
Clara, Sébastien, 53

de Paiva Santana, Charles, 97
Di Mauro, Emanuele, 135
Donat-Bouillud, Pierre, 25

Foscarin, Francesco, 87
Fournier-S'Niehotta, Raphaël, 87

Galleron, Philippe, 105
Guichaoua, Corentin, 71
Guigue, Didier, 97
Guillot, Pierre, 15

Jacquemard, Florent, 87

Larrieu, Maxence, 63
Lemouton, Serge, 11
Libermann, Nathan, 81

Maestri, Eric, 105
Malt, Mikhail, 121
Mihalic, Alexander, 121
Millot, Jean, 105

Pachet, François, 7
Paris, Eliott, 105
Pottier, Laurent, 11

Rigaux, Philippe, 87

Scurto, Hugo, 115

Vincent, Emmanuel, 81
Vulliard, Pierre-Henri, 131

Warnier, Jacques, 11