Journées d'Informatique Musicale

Captation, Transformation, Sonification

21 au 23 mai 2014

Bourges

ÉCOLE NATIONALE SUPÉRIEURE D'ART DE BOURGES

JIM 2014

Journées d'Informatique Musicale

Captation, Transformation, Sonification

21 au 23 mai 2014

Bourges

Sommaire JIM 2014

Session 3, Outils pour la création

Journée Transformation, jeudi, 22 mai 2014

Dominante de la Journée (Keynote) 2 -

Transformation, la projection de différentes modalités dans l'espace créatif

Session 4, Studio Report

Session 5, Analyse

Session 6, Analyse & préservation

Journée Sonification, vendredi, 23 mai 2014

Dominante de la Journée (Keynote) 3,

Session 7, Composition

Table Ronde,

Annexes

Le son a une place particulière dans la vie de tous les jours et son utilisation dans un environnement scientifique, comme vecteur de présentation des données, est de plus en plus répandue. Nous consacrons ainsi, lors des *Journées d'Informatique Musicale* à Bourges, une journée à cette présentation de la connaissance sous forme sonore - *Journée Sonification*. Elle prend sa place aux côtés des journées dédiées au geste - *Journée Captation* - et au traitement du signal - *Journée Transformation*.

Bourges a une remarquable histoire dans la recherche et le développement de projets concernant la musique électroacoustique, l'informatique musicale et la sonification en particulier. Tout d'abord c'est le Centre National de Création Musicale qui a contribué au développement de la recherche dans le domaine de l'informatique musicale, de même que le Conservatoire avec sa classe de composition électroacoustique et l'École Nationale Supérieure d'Art (ENSA) avec ses Ateliers son. Enfin, l'un des permiers enseignements en France de la sonification a été dispensé à l'Institut Universitaire de Technologie (IUT).

C'est en s'inspirant de ces sources historiques que MusInfo, avec le soutien de l'Association Française d'Informatique Musicale et en collaboration avec l'ENSA et l'IUT de Bourges, reprend la relève et organise les *Journées d'Informatique Musicale* à Bourges, accompagnées de la première édition des *Journées Art & Science* en Région Centre avec un nouveau *Concours de composition électroacoustique*. Nous souhaitons ainsi ouvrir une nouvelle page dans l'histoire de la musique électroacoustique et de la recherche en informatique musicale au centre de la France.

Alexander Mihalič
Président de MusInfo

Comités JIM 2014

Comité d'organisation

Administration
Alexander Mihalic, IMéRA Marseille / IRCAM Paris

Communication
Fabien Cothenet, Mémoire Magnétique, Imphy

Scientifique
Mikhail Malt, IRCAM, IReMus, MINT –OMF, Sorbonne, Paris

Artistique
Robert Rudolf, France Musique Radio France / Conservatoire de Noisy-le-Sec

Comité de pilotage, AFIM

Daniel Arfib, Gérard Assayag, Marc Chemillier, Myriam Desainte-Catherine, Dominique Fober, Mikhaïl Malt, Yann Orlarey, François Pachet, Laurent Pottier, Julien Rabin, Jean-Michel Raczinski et Anne Sèdes.

Comité scientifique

Karim Barkati, Ircam, France ; Olivier Baudouin, MINT-OMF, Paris-Sorbonne, France ; Gregory Beller, Ircam, France ; Didier Guige, University of Paraiba, Brésil ; Alain Bonardi, Ircam, France ; Bruno Bossis, APP Université Rennes 2 — OMF — MINT — Paris — Sorbonne, France ; Jean Bresson, UMR STMS: IRCAM-CNRS-UPMC, France ; Claude Cadoz, Lab. ICA, Grenoble Inst. of Techn. /ACROE, Min. de la Culture et de la Communication, France ; Thierry Coduys, Le Hub, France ; Arshia Cont, Ircam, France ; Pierre Couprie IReMus, Paris-Sorbonne University — ESPE, France ; Eric Daubresse, Ircam, France ; Myriam Desainte-Catherine, LaBRI, Université de Bordeaux, France ; Frédéric Dufeu, CeReNeM — University of Huddersfield, Royaume Uni ; François-Xavier Féron, LaBRI — CNRS (UMR 5800), France ; Jean-Louis Giavitto, Ircam — UMR STMS 9912 CNRS, France ; Karim Haddad, Ircam, France ; Fernando Iazzetta, University of Sao Paulo, Brésil ; Florent Jacquemard, INRIA – Ircam, France ; Emmanuel Jourdan, Ircam, France — Cycling74, E.U. ; Serge Lemouton, Ircam, France ; Marco Liuni, Ircam — CNRS STMS, Analysis/Synthesis Team, France ; Università di Firenze, Dip. di Matematica « U. Dini », Italie ; Grégoire Lorieux, Ircam, France ; Mikhail Malt, Ircam – IreMus – MINT – OMF, Sorbonne, France ; Jônatas Manzolli, University of Campinas — NICS, Brésil ; Tom Mays, CNSMD de Paris, CICM Paris VIII, France ; Nicolas Viel, Paris IV — Sorbonne, France ; Geoffroy Peeters, Ircam, France ; Laurent Pottier, UJM-CIEREC, France ; Julien Rabin, GMEA, France ; Jean-Michel Raczinski, Arkamys, France ; Geber Ramalho, Universidade do Pernanbuco, Brésil ; Patrick Sanchez, CNRS — LMA Equipe SONS, France ; Stephan Schaub, University of Campinas — NICS, Brésil ; Anne Sedes, Paris VIII, France ; Peter Sinclair, Locus Sonus, France ; Benjamin Thigpen Musicien indépendant ; Vincent Tiffon, Univ. de Lille-Nord de France, CEAC — EDESAC — IRCAM-CNRS, France ; Todor Todoroff, UMONS/TCTS/ARTeM, BElgique

Sessions & Conférences

Journée Captation
mercredi, 21 mai 2014

MOTION CAPTURE FOR PERFORMANCE ANALYSIS AND INTERACTIVE APPLICATIONS

DOMINANTE DE LA JOURNEE (KEYNOTE)

Prof. Marcelo M. Wanderley
Director - Centre for Interdisciplinary Research in Music Media and Technology
CIRMMT
www.cirmmt.mcgill.ca
Input Devices and Music Interaction Laboratory
www.idmil.org

Movements and gestures are an integral part of music performance. Musicians playing acoustic instruments or digital musical instruments make a variety of movements that are either essential or ancillary to the production of sound, contributing to the experience of the performer as well as to the way and audience perceives the music.

In this talk I will discuss various research projects at the Input Devices and Music Interaction Laboratory (IDMIL) at McGill University dealing with the measurement of movement in music performances, focusing on motion capture of a variety of instruments (e.g. clarinet, violin, timpani) as a means to a) quantify the movements present in performance with acoustic and digital musical instruments, b) evaluate the repeatability of ancillary performer movements, c) create pedagogical tools to improve the learning of musical instruments, and d) provide control data for the synthesis of a virtual musician (timpanist).

Examples will be provided to illustrate the benefits (and limitations) of this approach.

EXTENSION DE GUIDO AUX NOTATIONS CONTEMPORAINES

Camille Le Roi, Colas Decron, Dominique Fober
Grame - Centre national de création musicale
{cleroi, decron, fober}@grame.fr

RÉSUMÉ

Le projet GUIDO regroupe à la fois un format textuel de description de partitions musicales, un moteur de rendu basé sur ce format, et une librairie qui fournit aux développeurs d'applications un support de haut niveau pour l'ensemble des services liés au format GUIDO et à son rendu sous forme graphique. A l'origine, le projet GUIDO traite de la notation traditionnelle de la musique occidentale. Le projet a été récemment étendu, tant du point de vue du format que du point de vue du rendu, pour adresser également les besoins les plus courants de la musique contemporaine. Cet article présente ces extensions et montre comment les choix de design fournissent à l'utilisateur une large palette de solutions pour la notation de la musique.

1. INTRODUCTION

Le projet GUIDO est né il y a presque 20 ans, tout d'abord comme format textuel de représentation de la musique [7, 8], puis comme moteur de rendu graphique de partitions musicales [14]. Lors de sa conception, le format GUIDO se différencie des approches proposées par SCORE [10], MuseData [5], DARMS [15] ou encore Humdrum [9] notamment parce qu'il est pleinement lisible par l'utilisateur. L'approche développée par GUIDO repose sur l'idée d'adéquation, qui invite à noter simplement les concepts musicaux simples, et à ne recourir à une syntaxe complexe que pour les notions musicales complexes.

Postérieurement au format GUIDO, d'autres représentations textuelles de la musique ont vu le jour, telles que Lilypond [12, 11] qui est assez similaire dans son format, ou MusicXML [3], orientée vers l'échange de partitions. Toutefois, le projet GUIDO reste unique car il est le seul qui propose à la fois une syntaxe, un moteur de rendu et une librairie C/C++ permettant d'embarquer des capacités de rendu de partitions dans des applications indépendantes. Par ailleurs, la rapidité et l'efficacité du moteur de rendu le rendent utilisable dans un contexte *temps réel* (pour des partitions dont le niveau de complexité n'est pas trop élevé), ouvrant la voie à des formes d'interactivité inédites [6, 2].

Pour répondre aux besoins les plus courants de la musique contemporaine, le format GUIDO ainsi que le moteur de rendu ont été étendus. Après un bref rappel sur le format GUIDO, cet article décrit ces extensions (nouveaux symboles puis améliorations des symboles existants), et situe les choix faits en matière de rendu graphique dans les pratiques identifiées dans la littérature contemporaine. Nous montrerons également comment une forme de *langage générique* permet de combiner ces représentations et offre à l'utilisateur un large éventail de solutions pour la représentation de la musique.

2. LE FORMAT DE NOTATION GUIDO

Ci-dessous figurent les principes généraux du format de notation GUIDO. Ils sont fournis pour permettre la compréhension du contenu des sections suivantes. Pour plus de détails on peut se reporter à [1] ou [8].

Le format de base de GUIDO recouvre les notes, silences, altérations, voix indépendantes, et les concepts les plus courants de la notation musicale comme les clefs, métriques, armures, articulations, liaisons, etc. Les notes sont représentées par leur nom `(a b c d e f g h)`, une altération optionnelle (`#` et `&` pour dièse et bémol), un numéro d'octave optionnel (`1` par défaut) et une durée optionnelle exprimée sous forme de fraction (`1/4` pour la noire, `1/8` pour la croche, `1/2` pour la ronde, etc.).

Les *tags* sont utilisés pour donner des informations musicales supplémentaires, comme les liaisons, clefs, armures, etc. Un *tag* a une des formes suivantes :

```
\tagname<param-list>
\tagname<param-list>(note-series)
```

où *param-list* est une liste optionnelle de paramètres (chaînes de caractères ou nombres), et où *note-series* est l'ensemble des notes ou accords concernés par le *tag*. Dans ce cas nous parlerons de *range-tag* (*tag* possédant une durée explicite).

3. EXTENSIONS DE LA NOTATION

3.1. Micro-tonalité

La micro-tonalité est limitée graphiquement à la représentation du quart de ton, alors que, du point de vue du langage, elle s'exprime comme un nombre flottant de demi-tons qui sont arrondis au quart de ton le plus proche pour le rendu graphique :

```
1)  \alter<detune>
2)  \alter<detune>(note-series)
```

La première forme (1) introduit la notion d'*altération courante* : l'altération reste valide jusqu'à une indication contraire. Dans la deuxième forme (2), l'altération ne s'applique qu'aux notes indiquées.

EXEMPLE

```
[
  a&& \alter<-1.5>(a) a& \alter<-0.5>(a)
  a \alter<0.5>(a) a# \alter<1.5>(a)
  \alter<1.8>(a)
]
```

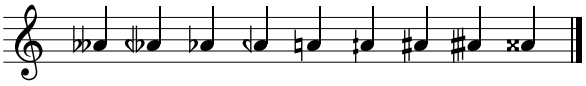


Figure 1. Micro-tonalité arrondie au quart de ton

La micro-tonalité est également introduite dans les armures de clef libres. De manière similaire, elle s'exprime sous forme de nombres flottants indiqués entre crochets.

EXEMPLE

```
[
  \key<"free=c[1.5]g#b&a[-1.5]"> a g#
  \alter<0.5> g  a \alter<1.5> g
]
```

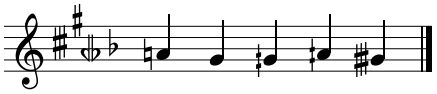


Figure 2. Armure de clef libre avec micro-tonalité

3.2. Têtes de notes

La librairie GUIDO a été étendue avec 6 nouveaux symboles de têtes de notes. Son implémentation repose sur le tag `\noteFormat`, déjà existant, qui modifie l'apparence des notes qui le suivent (couleur, taille, offset, etc.), et auquel nous avons ajouté un attribut de style, permettant à l'utilisateur de choisir l'apparence des têtes de notes.

```
\noteFormat<style=noteHeadStyle>
```

où `noteHeadStyle` est parmi :
- `x` : une croix
- `diamond` : un losange
- `round` : un rond
- `square` : un carré
- `triangle` : un triangle, pointe en haut
- `reversedTriangle` : un triangle, pointe en bas

La première portée de l'exemple de la figure 3 est générée avec le code suivant :

```
[
  \noteFormat<style="x"> a
  \noteFormat<style="diamond"> a
  \noteFormat<style="round"> a
  \noteFormat<style="square"> a
  \noteFormat<style="triangle"> a
  \noteFormat<style="reversedTriangle"> a
]
```

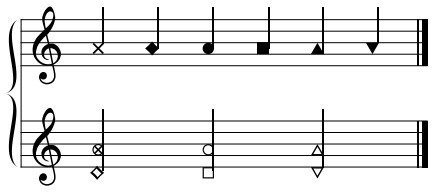


Figure 3. Les 6 nouveaux *noteheads* ajoutés

Pour l'apparence graphique de ces éléments, nous nous sommes principalement inspirés des ouvrages de E. Gould [4] et de K. Stone [16].

Des ajustements particuliers ont été nécessaires pour tenir compte des différences graphiques entre les têtes de notes :
- insertion d'un offset horizontal variable selon le symbole choisi, l'orientation de la hampe, l'utilisation en accord ou non (les accords pouvant posséder des *noteheads* différents pour chacune de leurs notes) ;
- adaptation de la longueur de la hampe selon le type de symbole utilisé (par exemple, différence de longueur entre la croix et le losange).

3.3. Métriques complexes

La métrique complexe permet d'indiquer une subdivision de la métrique par une somme en place et lieu de la métrique habituelle. La syntaxe est la même que pour une métrique normale, elle fait usage du tag `\meter` et il suffit de remplacer le numérateur par une somme, comme le montre l'exemple ci-dessous, illustré en figure 4.

EXEMPLE

```
[ \meter<"3+3+2/4"> a b g ]
```

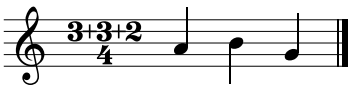


Figure 4. Métrique complexe

3.4. *Clusters*

Un *cluster* est un accord musical contenant plusieurs notes consécutives. Il peut par exemple être effectué sur un piano où il se joue en déclenchant des touches contiguës (en utilisant son avant-bras, par exemple).

Un *cluster* est indiqué par un range-tag s'appliquant à des accords de deux notes : les notes extrêmes du *cluster* (les autres notes éventuelles seront ignorées).

```
\cluster<param-list>(chord-series)
```

Comme l'indique en partie la figure 5, les choix, faits à partir d'ouvrages référents [4, 16], ont été les suivants :
- le *cluster* est représenté par un rectangle ;

EXEMPLE

```
[
  \cluster({a, d} {c/8, f}
      {a/2, d2} {c1/1, f2})
]
```

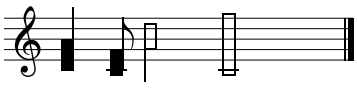


Figure 5. *Cluster*

– ce rectangle est plein ou vide, selon la durée du *cluster* ;
– la hampe est conservée pour garder l'indication de la durée.

Le tag `\cluster` supporte également les paramètres de contrôle standards (`size`, `dx`, `dy`, `color`) auxquels ont été ajoutés des paramètres `hdx` et `hdy` permettant d'introduire un décalage s'appliquant à la tête de *cluster*.

Un *cluster* est vu comme une note et supporte donc le tag `\noteFormat` ainsi que le contrôle de l'orientation des têtes de notes (`\headsReverse`, `\headsLeft` et `\headsRight`) ainsi qu'illustré en figure 6.

EXEMPLE

```
[
  \cluster<color="red", hdx=1, hdy=3>({a})
  \cluster<size=0.5>({f, c2})
  \noteFormat<color="purple">
  \headsReverse
  \cluster<color="green", size=2>({f, g2})
  \cluster<"blue">({d1/2, g})
]
```

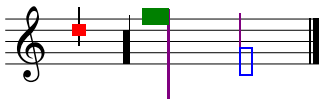


Figure 6. *Clusters* : interactions avec d'autres tags

3.5. *Glissando*

Le *glissando* se présente classiquement comme une ligne joignant deux notes décalées dans le temps (figure 7), indiquant ainsi un glissement, continu ou non suivant le type d'instrument, d'une hauteur à une autre.

La syntaxe choisie a été la suivante :

```
\glissando<params>(notes)
params :
  - fill = "true" ou "false" :
    option de remplissage
  - thickness : épaisseur de la ligne
```

Il paraissait important de donner la possibilité à l'utilisateur de n'écrire qu'une seule fois le tag pour décrire plusieurs *glissandi* consécutifs, contrairement à la description de lilypond où le tag doit être répété entre chaque note. Ainsi le range-tag apparaissait comme la meilleure solution pour décrire un ou plusieurs *glissandi* : toutes les notes comprises dans les parenthèses seraient prises en compte et liées les unes aux autres par des *glissandi*.

Concernant le moteur de rendu, l'algorithme de démultiplication d'un même tag et de répartition de celui-ci est fortement inspiré de celui du `\tie`, ou liaison de prolongation, qui doit également créer une liaison entre chacune des notes affectées.

EXEMPLE

```
[ \glissando(g b d) ]
```

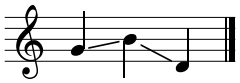


Figure 7. *Glissando*

3.6. Liens de croches en soufflet (*feathered beaming*)

Décrite par Gardner Read comme "a highly graphic representation of rythmic flexibility" [13] (p. 94), cette notation contemporaine d'un *accelerando* ou *ritardando* à travers les liens de croches est, de manière générale, encore peu répandue dans les ouvrages de théorie, et se résume souvent à des cas simples, partant ou arrivant à la simple croche, c'est à dire partant de, ou arrivant vers, un unique point. Pourtant il paraît intéressant de pouvoir offrir la possibilité aux compositeurs de décrire le passage entre n'importe quelles valeurs : par exemple d'une double-croche à une triple-croche, ou d'une quadruple-croche à une double-croche, etc. (figure 8).

EXEMPLE

```
[
 \fBeam(a/16 a a a/32) \fBeam(a/64 a a a a/16)
]
```

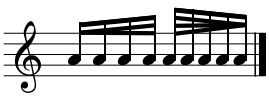


Figure 8. *Feathered beams* plus complexes

Dans un désir de simplicité, il a été décidé de décrire par un seul range-tag un seul groupe de notes liées, correspondant à un point de départ et un point d'arrivée uniques.

Il est cependant possible de chaîner plusieurs *feathered beams* en utilisant la forme `Begin / End` des range-tags

(i.e \fBeamBegin et \fBeamEnd), permettant alors, non seulement de chaîner deux *beams* par une note (figure 9), mais aussi de les faire se chevaucher sur plusieurs notes.

EXEMPLE

```
[
  \fBeamBegin:1 c/8 d e/16
  \fBeamBegin:2 f/32 \fBeamEnd:1
  e/16 d/8 c \fBeamEnd:2
]
```

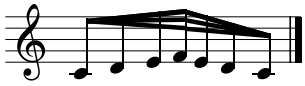


Figure 9. *Feathered beams* chaînés

On remarque que, même dans un ouvrage tel que celui d'E. Gould [4], peu de détails sont donnés sur cette notation, et on se heurte rapidement à des incohérences telles que celle de la figure 10 tirée de ce même ouvrage : on veut faire tenir en un temps une croche et quatre notes de durées inférieures à la croche mais supérieures à la triple-croche.

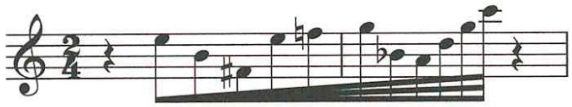


Figure 10. Exemple d'incohérence entre la durée totale et les durées individuelles des notes d'un groupe lié (Behind Bars, p. 158).

Ce manque d'une exacte cohérence entre tempo et aspect graphique est également souligné par Kurt Stone [16] : "Besides, the gradual increase or decrease in the number of beams makes exact indications of beat-units impossible" (p. 124).

Ainsi on voit l'importance des choix de design de ce symbole, sur la manière de le décrire textuellement, de le dessiner en particulier dans le cas de plusieurs voix simultanées, et de manière générale sur la liberté laissée à l'utilisateur.

Concernant les possibilités de description données à l'utilisateur, nous avons pu voir que plusieurs critères, parfois incohérents entre eux, pouvaient intervenir dans la description et devaient pouvoir être imposés par l'utilisateur :

- le nombre de notes dans le groupe ;
- leurs durées individuelles et intrinsèques qui définissent également leur espacement ;
- la durée totale du groupe, directement dépendante des durées individuelles ;
- les durées de départ et d'arrivée qui définissent l'aspect graphique du *feathered beam*.

Ces différents niveaux de description nous poussent à faire une claire distinction entre l'aspect temporel et l'aspect graphique de notre groupe de notes. Il a été décidé de laisser au compositeur la liberté de donner à son *feathered beam* l'aspect graphique qu'il désire, indépendamment des durées internes de ses notes. Il pourra ainsi jouer sur les espacements entre notes, sur le nombre de notes, et sur la durée totale de manière "classique" en imposant à chaque note une durée, et en veillant à ce que la somme de toutes les notes corresponde à la durée totale souhaitée. Cette durée totale peut être indiquée sur la partition grâce au paramètre drawDuration (figure 11).

EXEMPLE

```
[ \fBeam<drawDuration="true"> (
    a/8 a a/16 a a a/32 a )]
```

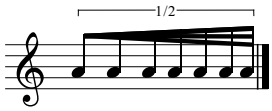


Figure 11. Représentation de la durée

Sans plus d'indication de sa part, l'aspect graphique suivra les durées réelles des notes et prendra comme points de départ et d'arrivée les durées des première et dernière notes. C'est le cas de la figure 11.

En revanche, le *param* durations pourra lui permettre d'imposer un aspect graphique tout autre, lui donnant ainsi la possibilité d'indiquer à l'interprète un changement de tempo plus ou moins important, et pas nécessairement cohérent avec les durées intrinsèques des notes, mais ayant du sens dans l'interprétation (figure 12).

EXEMPLE

```
[ \fBeam<durations="1/16, 1/64",
     drawDuration="true">(
    a/8 a/16 a a a/32 a )]
```

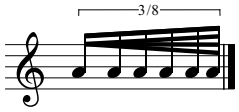


Figure 12. Aspect graphique imposé par l'utilisateur

En résumé, la syntaxe générique est la suivante :

```
\fBeam<params>(notes)
params :
  - durations = "firstDur", "lastDur" :
    durée des première et dernière notes
  - drawDuration :
    indication de la durée totale
```

3.7. Tags *\staffOff* et *\staffOn*

Les tags \staffOff et \staffOn permettent de faire disparaître et apparaître des parties de la partition, en particulier pour cacher une voix muette pendant un certain temps (figure 13).

Son usage est généralisé à tous les éléments et toutes les voix de la partition et permet des *montages* plus complexes (figure 14).

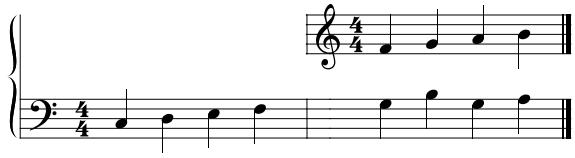


Figure 13. \staffOff classique

```
{
 [
  \clef \meter<"3/4"> a b/2 e/4
  \staffOff f g
 ],
 [
  \meter<"3/4"> \staffOff \clef c
  \staffOn d g | \staffOff f g \staffOn b
 ]
}
```

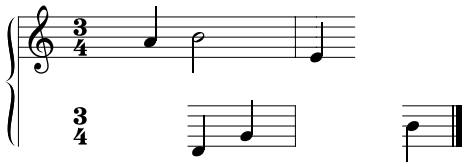


Figure 14. \staffOff plus complexe

L'implémentation graphique est le pendant de la description textuelle : tous les éléments inscrits après un tag \staffOff dans la description textuelle ne sont pas dessinés jusqu'au prochain \staffOn dans la séquence courante. Pour plusieurs éléments simultanés, c'est donc bien l'ordre dans la description textuelle qui indiquera le comportement à adopter. Ainsi :

```
[ \meter<"4/4"> \clef \staffOff
  a b \staffOn c ]
```

n'est pas équivalent à :

```
[ \staffOff \meter<"4/4">
  \clef a b \staffOn c ]
```

Quant à la portée, elle commence, ou s'arrête, aux positions correspondant au début des éléments suivant le tag.

3.8. Graphiques arbitraires

Le tag \symbol permet d'insérer des graphiques arbitraires dans la partition, sous forme de fichier image externe. Les formats graphiques supportés sont de type *png*, *jpg* ou *bmp*. Les paramètres de contrôle permettent d'ajuster la taille et la position de l'image dans la partition.

Sa syntaxe est la suivante :

```
1) \symbol<params>
2) \symbol<params>(notes)
  params :
    - filePath : chemin du fichier
    - position = [top | mid | bot] :
      position de l'image
    - size : taille de l'image
    - w/h : largeur/hauteur de l'image
```

Le tag \symbol peut être utilisé comme un tag simple ou comme un range-tag :

1. le symbole ne possède pas de durée mais prend de la place sur la portée, selon sa taille ;
2. le symbole s'applique aux notes indiquées (sans étirement).

Le fichier est indiqué par un chemin absolu ou relatif. Dans le second cas, le chemin est relatif au répertoire contenant le fichier source ou au répertoire *home* de l'utilisateur.

EXEMPLE

Sur la première portée, le symbole ne possède pas de durée, contrairement à la deuxième portée.

```
{
  [
    \meter<"4/4"> c f
    \symbol<file="silence.png", dx=-5,
        dy=-10>
    c d f
  ],
  [
    \meter<"4/4"> a d
    \symbol<file="ronds.png",
        position="bot"> (f g) g
  ]
}
```

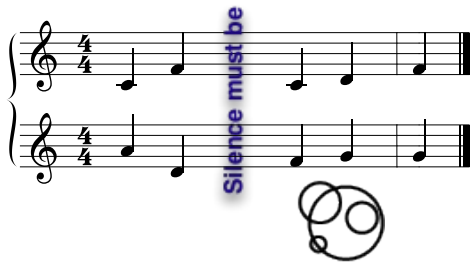


Figure 15. Graphiques arbitraires

4. CONTRÔLE DE RENDU

Après avoir exposé les nouvelles notations implémentées dans GUIDO, nous allons à présent nous intéresser aux améliorations et subtilités rajoutées aux notations existantes.

4.1. Nouveaux paramètres

De nouveaux paramètres ont été introduits pour plusieurs tags déjà existants, permettant un contrôle étendu du rendu graphique :

- `\staffFormat` : nouvelle possibilité de contrôle de l'épaisseur des lignes de la portée
- `\tuplet` : nouvelle possibilité de contrôle de l'épaisseur des lignes, de la taille du texte, etc.
- `\cresc` et `\decresc` : maintenant paramétrable en position, épaisseur, couleur.

On se reportera à la documentation utilisateur pour le détail de ces paramètres.

EXEMPLE

L'exemple suivant illustre ces améliorations et génère la partition de la figure 16.

```
{
  [ \tuplet<"-3-", lineThickness=0.4,
        bold="true", textSize=2, dy1=10> (
      \cresc<dm="fff", dx1=-5, dx2=5,
          dy=1, deltaY=5, color="red",
          size=1.7> (a a a) )
  ],
  [ \staffFormat<lineThickness=0.5>
    \decresc<thickness=0.8> (g d e f)
  ]
}
```

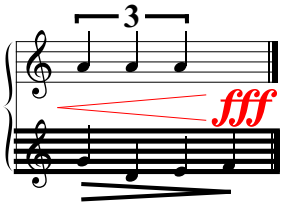


Figure 16. Contrôle de rendu étendu

4.2. Améliorations des *trilles*

```
\trill<params>(chord-series)
 params:
  - tr = [true | false]
  - anchor = [note | tr]
```

Un *trille* est un ornement musical qui consiste à alterner rapidement la note de base, sur laquelle est noté le *trille*, et la note située juste au-dessus.

Dans la version précédente de GUIDO, le *trille* à proprement parler n'était indiqué que par le symbole ***tr*** placé au-dessus de la note. Il paraissait pourtant important que la notation comprenne également la ligne ondulée du *trille* qui suit celui-ci et en indique la durée. Comme ce symbole était déjà en partie implémenté, il fallait pouvoir lui offrir plus de souplesse sans pour autant en complexifier la syntaxe.

De plus, nous avons voulu donner la possibilité de choisir de dessiner ou non le signe ***tr***, ainsi que de pouvoir changer l'ancrage de la ligne pour le déplacer sur la tête de la note. Cela fut implémenté grâce au *param* `tr` qui accepte comme valeurs `true` ou `false` et à `anchor` qui accepte comme valeurs `note` ou `tr` (figure 17).

Le reste des modifications graphiques pouvant être effectuées manuellement par l'utilisateur à travers les paramètres classiques de déplacement (`dx`, `dy`), de couleur (`color`), et de taille (`size`).

EXEMPLE

```
[ \trill<tr="false", anchor="note">
  ( {g} {a/2} ) ]
```

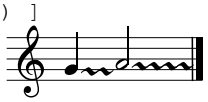


Figure 17. Cas du *trille* ancré à la tête de note, et ***tr*** optionnel

Enfin, une subtilité au niveau du contrôle du rendu a également été ajoutée. Il fallait définir le comportement quant aux notes liées. La solution adoptée est de dessiner la ligne du *trille* jusqu'à la fin de la dernière note liée si celle-ci provient d'une note plus longue découpée automatiquement, mais de s'arrêter normalement à la prochaine note si la liaison a été explicitée par l'utilisateur, qui pourra alors décider de répéter le *trille* sur cette autre note, ou non. (figure 18)

EXEMPLE

```
{
  [
    \meter<"2/4"> \trill({a} {a/2})
  ],
  [
    \meter<"2/4"> \trill(
      {a} \tie({a} {a})
    )
  ]
}
```

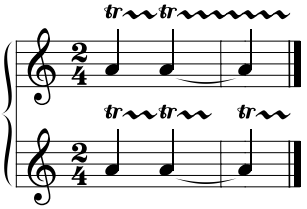


Figure 18. Cas du *trille* appliqué à des notes liées

4.3. *Glissandi* et accords

Un *glissando* entre deux accords soulève la question de la répartition des *glissandi* entre les notes des accords. La solution proposée est celle qui nous a paru la plus intuitive : c'est l'ordre des notes dans la description textuelle de l'accord qui détermine les relations entre les notes. Par exemple, pour 2 accords A et B, la première note de l'accord A sera liée à la première note de l'accord B, la seconde de A à la seconde de B, etc. (figure 19).

EXEMPLE

```
[ \glissando({e,a} {f,b} {a,d}) ]
```

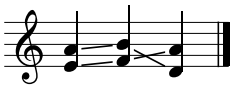
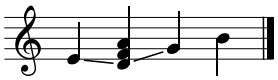


Figure 19. *glissandi* entre accords

L'unique problème avec ce choix peut se poser lors d'un conflit dans l'ordre imposé par le glissando précédent et par le suivant, comme dans le premier cas de l'exemple de la figure 20. Il est alors possible de faire appel à une seconde voix, et au tag `\staff<numéro de portée>` qui pourra créer sur une même portée d'autres *glissandi* en parallèle et indépendants des premiers (figure 20).

EXEMPLE

```
[ \glissando(e {d,f,a} g) b ]
```



```
{
  [ \glissando(e {d,f}) empty b ],
  [ \staff<1> empty \glissando(a g) empty ]
}
```

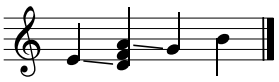


Figure 20. Cas de recours à une seconde voix pour le *glissando* (`empty` représente une note invisible.)

Enfin, lors de combinaisons de *glissandi* avec des accords ou des *clusters*, il nous a paru intéressant, comme on peut le voir dans [4] (p. 143) ou dans [16] (p. 61), de donner la possibilité de remplir l'espace entre ces *glissandi* grâce à un *param* `fill`. Une certaine flexibilité peut être trouvée dans le dessin grâce aux *params* graphiques `dx1`, `dx2`, `dy1`, `dy2`, ainsi que `thickness` qui permettent de modifier l'aspect du *glissando* (figure 21).

EXEMPLE

```
[
  \glissando<fill="true", dx1=-2, dx2=2,
      thickness=2.2> (
    \cluster({e,g} {c,b})
  )
  \glissando<fill="true">(
    {c,e,g} {a,c2,f1}
  )
]
```

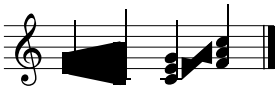


Figure 21. *Glissandi* entre accords et *clusters* avec *param* de remplissage

4.4. Combinaisons de *beams*

Pour finir, concernant le *feathered beaming* comme le *beaming* classique, il est maintenant possible de créer des hiérarchies de *beams* : c'est à dire d'englober plusieurs *beams* dans un plus grand, afin de les joindre par une barre principale commune (figure 22).

EXEMPLE

```
[
  \beam(
    \fBeam(c/8 e d f g/32)
    \fBeam(a/16 f e d c/64)
  )
]
```

Figure 22. *Feathered beams* englobés dans un plus grand

5. CONCLUSION

Le projet GUIDO se différencie des approches de type gravure musicale telles que Lilypond ou MuseScore (pour ne citer que les outils *libres*) en ce sens qu'il privilégie un rendu efficace et qu'il met en œuvre un grand nombre d'automatismes pour le calcul de la partition. Il est cependant plus limité en matière de complexité et de notations supportées.

Les extensions qui ont été présentées comblent pour partie le fossé entre le moteur GUIDO et les approches citées précédemment. Elles ont été réalisées principalement à la demande de compositeurs et au service de créations en cours. Elles s'intègrent dans le moteur existant dans le respect des différentes combinaisons du langage. Leur implémentation a soulevé des problèmes de design qui ont

été résolus en laissant à l'utilisateur le choix entre les différentes possibilités. Elles sont disponibles avec la version 1.52 de la librairie GUIDO. Le projet GUIDO est un projet *open source* hébergé sur Sourceforge [1] .

6. REFERENCES

[1] C. Daudin, Dominique Fober, Stephane Letz, and Yann Orlarey. La librairie guido – une boite à outils pour le rendu de partitions musicales. In ACROE, editor, *Actes des Journées d'Informatique Musicale JIM'09 – Grenoble*, pages 59–64, 2009.

[2] Dominique Fober, Yann Orlarey, and Stephane Letz. Inscore – an environment for the design of live music scores. In *Proceedings of the Linux Audio Conference – LAC 2012*, pages 47–54, 2012.

[3] M. Good. MusicXML for Notation and Analysis. In W. B. Hewlett and E. Selfridge-Field, editors, *The Virtual Score*, pages 113–124. MIT Press, 2001.

[4] E. Gould. *Behind Bars : The Definitive Guide to Music Notation*. Faber Edition Series. Faber Music Limited, 2011.

[5] Walter B. Hewlett. MuseData : Multipurpose Representation. In Selfridge-Field E., editor, *Beyond MIDI, The handbook of Musical Codes.*, pages 402–447. MIT Press, 1997.

[6] Richard Hoadley. Calder's violin : Real-time notation and performance through musically expressive algorithms. In ICMA, editor, *Proceedings of International Computer Music Conference*, pages 188–193, 2012.

[7] H. Hoos, K. Hamel, K. Renz, and J. Kilian. The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music. In *Proceedings of the International Computer Music Conference*, pages 451–454. ICMA, 1998.

[8] H. H. Hoos and K. A. Hamel. The GUIDO Music Notation Format Specification - version 1.0, part 1 : Basic GUIDO. Technical report TI 20/97, Technische Universitat Darmstadt, 1997.

[9] David Huron. Humdrum and Kern : Selective Feature Encoding. In Selfridge-Field E., editor, *Beyond MIDI, The handbook of Musical Codes.*, pages 376–401. MIT Press, 1997.

[10] Smith Leland. SCORE. In *Beyond MIDI, The handbook of Musical Codes.*, pages 252–280. MIT Press, 1997.

[11] Han-Wen Nienhuys. Lilypond, automated music formatting and the art of shipping. In *Forum Internacional Software Livre 2006 (FISL7.0)*, 2006.

[12] Han-Wen Nienhuys and Jan Nieuwenhuizen. LilyPond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, May 2003.

[13] G. Read. *Music notation : a manual of modern practice*. Crescendo book. Allyn and Bacon, 1969.

[14] Kai Renz. *Algorithms and Data Structures for a Music Notation System based on GUIDO Music Notation*. PhD thesis, Technischen Universität Darmstadt, 2002.

[15] E. Selfridge-Field. DARMS, Its Dialiects, and Its Uses. In *Beyond MIDI, The handbook of Musical Codes.*, pages 163–174. MIT Press, 1997.

[16] K. Stone. *Music Notation in the Twentieth Century : A Practical Guidebook*. W W Norton & Company Incorporated, 1980.

1. `http://guidolib.sf.net`

EXECUTING HIERARCHICAL INTERACTIVE SCORES IN REACTIVEML

Jaime Arias, Myriam Desainte-Catherine, Sylvain Salvati
Univ. Bordeaux, LaBRI, Bordeaux, F-33000, France
CNRS, UMR 5800, Bordeaux, F-33000, France
IPB, LaBRI, Bordeaux, F-33000, France
{jaime.arias, myriam, sylvain.salvati}@labri.fr

Camilo Rueda
Departamento de Electrónica y
Ciencias de la Computación
Pontificia Universidad Javeriana
Cali, Colombia
crueda@javerianacali.edu.co

RÉSUMÉ

Le modèle des partitions interactives permet d'écrire et d'exécuter des scénarios multimédia interactifs. Le logiciel I-SCORE implémente ce modèle au moyen des Hierarchical Time Stream Petri Nets (HTSPN). Cependant, cette implémentation est très statique et l'ajout de certaines fonctionnalités peut nécessiter une reconception complète du réseau. Un autre problème de I-SCORE est qu'il ne fournit pas un bon retour visuel de l'exécution d'un scénario. Dans cet article, nous définissons et implémentons un interprète de partitions interactives avec le langage de programmation synchrone REACTIVEML. Dans ce travail, nous tirons parti de l'expressivité du modèle réactif et de la puissance de la programmation fonctionnelle pour obtenir un interprète plus simple et plus dynamique. Contrairement à l'implémentation basée sur les réseaux de Petri, cette approche permet de définir précisément l'aspect hiérarchique et permet de prototyper facilement de nouvelles fonctionnalités. Nous proposons aussi une visualisation temps réel de l'exécution en utilisant l'environnement INSCORE.

ABSTRACT

Interactive scores proposes a model to write and execute interactive multimedia scores. The software I-SCORE implements the above model using Hierarchical Time Stream Petri Nets (HTSPN). However, this model is very static and modelling new features would require a complete redesign of the network or sometimes they cannot be expressed. Another problem of I-SCORE is that it does not provide a good visual feedback of the execution of the scenario. In this work, we define and implement an interpreter of interactive scores using the synchronous programming language REACTIVEML. Our work takes advantage of the expressiveness of the reactive model and the power of functional programming to develop an interpreter more dynamic and simple. Contrary to the Petri Net model, our approach allows to model precisely the hierarchical behaviour, and permits the easy prototyping of new features. We also propose a visualization system using the environment INSCORE that provides a real-time visualization of the execution of the score.

1. INTRODUCTION

Interactive scores [10] proposes a model to write and execute interactive scenarios composed of several multimedia processes. In this model, the temporal organization of the scenario is described by means of flexible and fixed temporal relations among temporal objects (i.e., multimedia processes) that are preserved during the writing and performance stage.

The implementation of interactive scores in the software I-SCORE [1] is based on Petri Nets [17]. Such implementation provides an efficient and safe execution, but implies a quite static structure. Indeed, only elements that have been planned during the composition process can be executed. Therefore, it is not possible to modify the structure of the scenario during execution, for example, dynamically add a new element that was not written before execution. In addition, modelling new features for I-SCORE such as conditionals, loops or handling streams would require a complete redesign of the network. Therefore, this model is not suitable for compositional development and integration of new features that composers increasingly need to write more complex scenarios.

In this paper, we explore a new way to define and implement interactive scores, aiming at a more dynamic model. For this purpose, we use REACTIVEML [16], a programming language for implementing interactive systems (e.g., video games and graphical user interfaces). This language is based on the synchronous reactive model of Boussinot [8], then it provides a global discrete model of time, clear semantics, and unlike Petri nets, synchronous and deterministic parallel composition and features such as dynamic creation of processes. Moreover, REACTIVEML has been previously used in music applications [4, 5] showing to be very expressive, efficient, capable of interacting with the environment during the performance of complex scores, and well suited for building prototypes easily.

The rest of the paper is organized as follows. In Section 2 we present the I-SCORE system and we briefly introduce

[1] http://i-score.org

the REACTIVEML programming language. In Section 3 we describe the implementation in REACTIVEML of the new interpreter of interactive scores. Next, in Section 4 we present the improved visualization system in INSCORE. Finally, in Section 5 we present related work, conclusions, and ideas for future work.

2. PRELIMINARIES

In this section we present the I-SCORE system and the necessary notions of REACTIVEML language.

2.1. I-SCORE

I-SCORE [3, 17] is a software for composing and executing interactive multimedia scenarios [2]. It consists of two sides: *authoring* and *performance*. In the authoring side, the composer designs the multimedia scenario while in the performance side the performer executes the scenario with interactive capabilities. Next, we present in more detail both sides.

2.1.1. Authoring side

In interactive scores [10], multimedia elements are temporal structures represented as boxes. These boxes can be either *simple* or *complex*. A simple box represents a multimedia process that will be executed by an external application such as PURE DATA [2] or MAX/MSP [3]. I-SCORE controls such applications by means of OSC [4] messages. On the other hand, a complex box allows to gather and execute a set of boxes making possible the design of large and complex scenarios. However, this box does not execute a multimedia process.

The temporal organization of the score is partially defined by temporal relations. They indicate a precedence relation between boxes using Allen's relations [1], as well as a delay between them. A temporal relation can be either *rigid* or *flexible*. In a rigid relation, the duration of the delay is fixed whereas in a flexible relation, the duration is partially defined by an interval of time (i.e., it has a minimum and a maximum duration).

The composer can add interaction points to boxes. They allow to modify the preceding relations (i.e., start date) and the duration (i.e., end date) of boxes during the execution. In the case of a complex box, an interaction point stops abruptly the box with its children. It is important to know that the temporal organization of the score is preserved during composition and performance. Therefore, the performer can interpret the same score in different ways within the constraints of the composer. In I-SCORE, an interaction point is triggered by sending the OSC message defined by the composer. Additionally, all preceding relations of a box with an interaction point are flexible, otherwise they are rigid.

[2] http://puredata.info
[3] http://cycling74.com/products/max/
[4] http://opensoundcontrol.org/introduction-osc

In Figure 1 we illustrate an interactive scenario with all its temporal relations. Here, the horizontal axis represents time and the vertical axis has no meaning. Additionally, the start and end of boxes are partially defined by temporal relations that are represented as solid (rigid relation) and dashed (flexible relation) arrows. Moreover, tabs represent the interactions points of boxes.

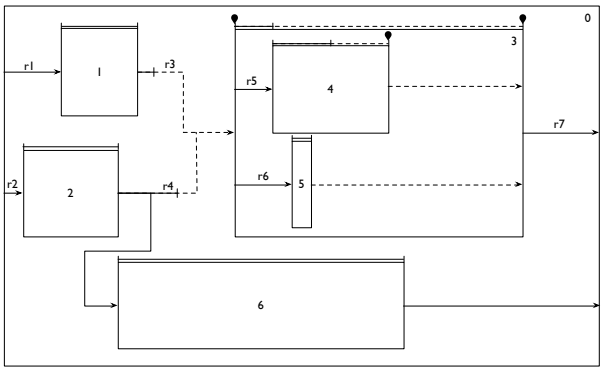


Figure 1. Example of an interactive scenario.

2.1.2. Performance side

In order to execute the scenario designed in the previous side, I-SCORE translates the score into a Hierarchical Time Stream Petri Net (HTSPN) [19]. The Petri net model allows to trigger the interactive events, and also it denotes and preserves the temporal organization of the score during the execution. The reader may refer to [17] for further information on generating the HTSPN structure from the score. The following example illustrates the execution of a scenario.

Example 1. Consider the interactive score in Figure 1 with the following configuration :

- Box 1 starts 3 seconds after the start of the scenario. Its duration is 4 seconds.
- Box 2 starts 1 second after the start of the scenario. Its duration is 5 seconds.
- Box 6 starts immediately (i.e., 0 seconds) after the end of the box 2. Its duration is 15 seconds.
- Box 3 is a complex box with two children; the box 4 and 5. The start of the box is defined by the flexible relations $r3$ and $r4$ whose durations are $[1, 4]$ and $[3, 8]$, respectively. Therefore, at any instant in which the above relations are satisfied, the box may be started by triggering the interaction point. In Section 3 we elaborate more on the semantics of temporal relations and interaction points.
- The duration of the box 3 is the interval $[2, \infty]$, then the box may be stopped after 2 seconds of its starting by triggering the interaction point.

- Box 4 starts 2 seconds after the start of its parent (i.e., box 3). Its duration is the interval $[3, 6]$, then the box may be stopped after 3 seconds of its starting by triggering the interaction point. It is important to know that the box will stop when it reaches its maximum duration (i.e., 6 seconds) if the interaction point is not triggered before.
- Box 5 starts 3 seconds after the start of the box 3. Its duration is 1 second.
- The scenario finishes when the box 6 finishes and 4 seconds have elapsed after the end of the box 3.

Following [17], we translated the score into its equivalent HTSPN structure (Figure 2). For the sake of simplicity, we do not show the time interval on the arcs. Note that transitions with double stroke are those with an interaction point.

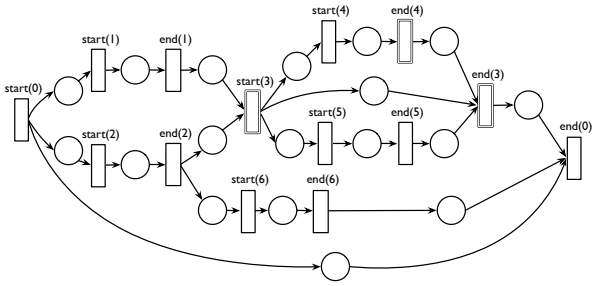


Figure 2. HTSPN structure of the scenario specified in Example 1.

We show in Figure 4 an execution of the above scenario where only the interaction point of the box 3 is triggered. All other intervals reach their maximum duration. Note that box 3 must be stopped, otherwise it will never end because it has an infinite duration. `merge(r3,r4)` represents the interval of time in which the box 3 may start. In this interval, the relations $r3$ and $r4$ are satisfied. As can be seen, the interaction point is not triggered, then the box starts when the interval reachs its maximum duration. We can conclude that this scenario finishes in 27 seconds only if the interaction point at the end of the box 3 is triggered at 23 seconds.

2.2. REACTIVEML

REACTIVEML [16] is a synchronous reactive programming language designed to implement interactive systems such as graphical user interfaces and video games. It is based on the reactive model of Boussinot [8] and it is built as an extension of the functional programming language OCAML [5]. Therefore, it combines the power of functional programming with the expressiveness of synchronous paradigm [6].

The reactive synchronous model provides the notion of a global logical time. Then, time is viewed as a sequence of logical instants. Additionally, parallel processes are executed synchronously (*lock step*) and they communicate with each other in zero time. This communication is made by broadcasting signals that are characterized by a status defined at every logical instance: *present* or *absent*. In contrast to ESTEREL [7], the reaction to absence of signals is delayed, then the programs are causal by construction (i.e., a signal cannot be present and absent during the same instant). Moreover, the reactive model provides dynamic features such as dynamic creation of processes. Indeed, REACTIVEML provides a toplevel [15] to dynamically write, load and execute programs.

In REACTIVEML, regular OCAML functions are instantaneous (i.e., the output is returned in the same instant) whereas *processes* (**process** keyword) can be executed through several instants. Next, we use the program shown in Figure 3 to describe the basic expressions of REACTIVEML.

```
1  let process killable_p p s =
2    do
3      run p
4    until s done
5
6  let process wait tic dur =
7    for i=1 to dur do await tic done
8
9  let process emit_tic period tic =
10 let start = Unix.gettimeofday () in
11 let next = ref (start +. period) in
12 loop
13   let current = Unix.gettimeofday() in
14     if (current >= !next) then begin
15       emit tic ();
16       next := !next +. period
17     end;
18     pause
19 end
```

Figure 3. Example of REACTIVEML language.

Two expressions can be evaluated in sequence (`e1;e2`) or in parallel (`e1||e2`). In REACTIVEML is possible to write higher order processes like the process `killable_p` (line 1) which takes two arguments: a process `p` and a signal `s`. This process executes `p` until `s` is present. The expression **`run`** executes a process (line 3). There are two important control structures: the construction **`do`** `e` **`until`** `s` to interrupt the execution of `e` when the signal `s` is present, and the construction **`do`** `e` **`when`** `s` to suspend the execution of `e` when the signal `s` is absent.

Signals can be emitted (**`emit`**), and awaited (**`await`**). For instance, the process `wait` (line 6) takes two arguments: a signal `tic` and an integer `dur`. The purpose of this process is similar to a timer; it waits for the signal `tic` to be emitted a number `dur` of times. The expression **`await`** `s` waits for `s` to be emitted and it finishes in the next instant whereas the expression **`await immediate`** `s` waits for `s` to be emitted and it terminates instantaneously. An important characteristic of the REACTIVEML implementation is the absence of busy waiting: nothing is computed when no signal is present. The process `emit_tic` (line 9) takes two arguments: a float `period` and a signal `tic`. It works like a clock; it gets the current time by using the

[5] `http://ocaml.org`

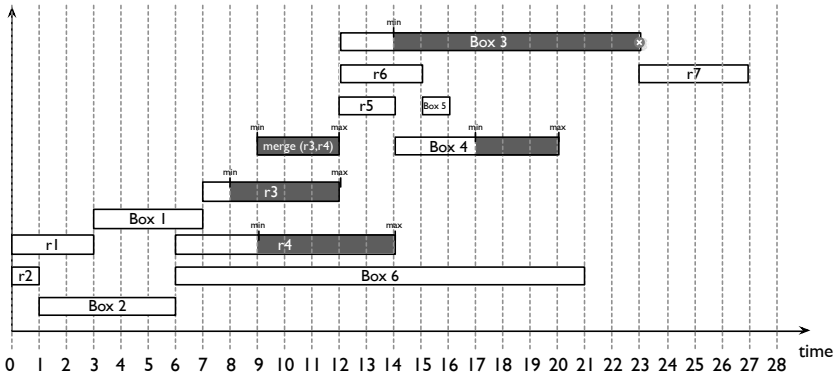


Figure 4. Execution of Example 1 where only the interaction point of the box 3 is triggered.

function `Unix.gettimeofday` from the `Unix` module, and emits the signal `tic` (line 15) whenever the period of time expires (line 14). The **`pause`** (line 18) keyword awaits for the next instant. The construction **`loop`** `e` **`end`** iterates infinitely `e`.

REACTIVEML also provides valued signals. They can be emitted (**`emit`** `s value`) and awaited to get the associated value (**`await`** `s (pattern)`**`in`** `expression`). Different values can be emitted during an instant (*multi-emission*). In this case, it is necessary to define how the emitted values will be combined during the same instant (**`signal`** `name` **`default`** `value` **`gather`** `function` **`in`** `expression`). The value obtained is available at the following instant in order to avoid causality problems. For example, the process `add` (Figure 5) declares the local signal `num` (line 2) with an initial value `0` and a function which adds two integers. The process `gen` (line 3) generates a set of values that are emitted through the signal `num` at the same instant. The process `print` (line 6) awaits for the signal `num`, and then it prints the value in `n`. Note that `n` contains the sum of all values generated by the process `gen`.

```
1  let process add max =
2    signal num default 0 gather fun x y -> x+y in
3    let process gen =
4      (for i=1 to max do emit num i done)
5    in
6    let process print =
7      await num (n) in
8      print_endline (string_of_int n)
9    in
10     run gen || run print
```

Figure 5. Example of multi-emission of signals.

3. SYNCHRONOUS MODEL OF INTERACTIVE SCORES

In this section, we present a new execution model of interactive scores using the reactive programming language REACTIVEML. The implementation of the interpreter is divided into two main modules: `Time` and `Motor`. The module `Time` interfaces the abstract time relative to the tempo (in beats) and the physical time (in ms). This module is based on the work of Baudart *et al.* [4, 5]. The module `Motor` interprets the interactive score and interacts with the environment by listening external events and triggering external multimedia processes. The implementation fulfills the operational semantics of interactive scores [10]. In the following, we describe the above modules.

3.1. Modelling Time

REACTIVEML, like other synchronous languages, provides the notion of a global logical time. Then, time is viewed as a sequence of logical instants. The process `emit_tic` (Figure 3), explained in Section 2.2, is the interface between the physical time and the logical time. Its purpose is to generate the clock of the system by emitting a signal in a periodic time. Therefore, from this signal of clock, we can define a process to express delays by waiting a specific number of ticks (process `wait` in Figure 3).

3.2. Execution of Interactive Scores

Interactive scores [10] are basically composed of: boxes that represent multimedia processes; temporal relations that define the temporal organization of the score (i.e., the start and the end of boxes); and interaction points that transform a static score into dynamic by allowing the performer to modify the temporal relations during the execution. In the following, we present the implementation of the above elements in REACTIVEML.

3.2.1. Temporal relations

Temporal relations partially define the temporal organization of the scenario. They represent delays that allow to specify the start and the duration of boxes. Relations can be either *rigid* or *flexible*. In a rigid relation, the duration of the delay is fixed whereas in a flexible relation, the duration of the delay is partially defined by an interval of time. This interval can be modified by triggering an *interaction point* during the execution.

Hence, we define two types of intervals in order to represent temporal relations. The *fixed interval* represents a rigid relation and the *interactive interval* represents a flexible relation with an attached interaction point (Figure 6). The interaction point allows to stop the interval during the defined interval of time (i.e., between the minimum and the maximum duration) by triggering a specific event.

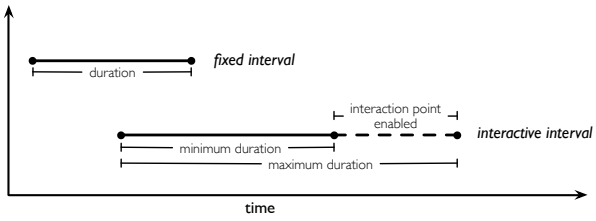


Figure 6. Fixed and flexible intervals.

In REACTIVEML, we represent a fixed interval as a tuple `(d,s)` where the signal `s` is emitted when the duration `d` has elapsed. On the other hand, the interactive interval is defined as a tuple `(min,max,ip)` where `min` and `max` are fixed intervals that represent the minimum and maximum duration of the interval, and `ip` is its interaction point. It should be noted that the duration of a flexible interval can be either finite or infinite.

Interaction points are represented as OSC messages that are sent from the environment and transmitted through a signal. An OSC message is represented as a tuple `(t, a)` where `t` is the address and `a` is the list of arguments with their type. For example, `('/light/1', [String ' luminosity'; Int32 90])`.

Hence, we define a temporal relation between two boxes as a tuple `(from,to,intrvl)` where `from` and `to` are the identifiers of the boxes involved in the relation, and `intrvl` is the interval that defines the delay between them.

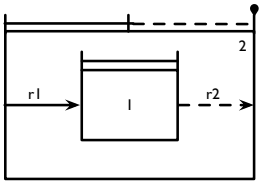
3.2.2. Boxes

Boxes can be either *simple* or *complex*. A simple box represents a multimedia process that is executed by an external application. For this reason, the interpreter only sends OSC messages in order to control its start and end. On the other hand, a complex box gather and execute a set of boxes with their own temporal organization. Recall that the duration of a box is defined by an interval. Then, it can be either fixed or interactive. In the rest of the paper, we call rigid boxes as process boxes and complex boxes as hierarchical boxes.

A process box is defined as a tuple `(id,intrvl,s_msg ,e_msg)` where: `id` is the identifier of the box; `intrvl` is the interval that defines its duration; `s_msg` and `e_msg` are the OSC messages to start and stop the external process, respectively. We define a hierarchical box as a tuple `(id, b_list, r_list, intrvl)` where: `id` is the identifier of the box; `b_list` is the list of its children; `r_list` is the list of the relations; and `intrvl` is the interval that defines its duration. The following example illustrates the representation of boxes and relations in REACTIVEML.

Example 2. Consider the hierarchical box shown in Figure 7a. This box, identified with the number 2, has a process box as child and an interaction point at the end. The duration of the hierarchical box is the interval $[2, \infty]$. The process box, identified with the number 1, starts 2 seconds after the starting of its parent and its duration is 2 seconds.

We show the specification in REACTIVEML of Example 2 in Figure 7b.



(a) Graphical representation.

```
1 signal pb1_s, hb1_s, hb2_s, r1_s, r2_s;
2 let ip = ("/stop",[Int32 3]);
3 let start_m = ("/box/1",[String "start"]);
4 let stop_m = ("/box/1",[String "stop"]);
5 let r1 = Fixed (Finite 2, r1_s);
6 let r2 = Fixed (Finite 0, r2_s);
7 let p_box = Process (1, Fixed (Finite 2, pb1_s), start_m,
       stop_m);
8 let h_box = Hierarchical (2, [p_box], [(2,1,r1);(1,2,r2)
      ], Interactive ((Finite 2, hb1_s), (Infinite, hb2_s)
      , ip))
```

(b) Specification in REACTIVEML.

Figure 7. Specification of a hierarchical box in REACTIVEML.

Firstly, we define the signals that will be emitted when the intervals reach their durations (line 1), the OSC message of the interaction point (line 2), and the OSC messages to start (line 3) and stop (line 4) the external multimedia process. Then, we define the interval `r1` (line 5) that determines the start of the process box (i.e., box 1). It is important to note that we need to specify both the type of the interval (i.e., `Fixed` or `Interactive`) and the duration (i.e., `Finite` or `Infinite`). Since the parent of the box 1 has an interaction point, the duration of the interval `r2` is not relevant, therefore we define its duration as 0 seconds (line 6). Next, we define the process box `p_box` (line 7) with a fixed duration of 2 seconds and the messages defined previously. Finally, we define the hierarchical box with its child `p_box`, its internal relations `r1` and `r2`, and its duration. Note that the duration of box 2 is defined by means of an interactive interval because it has an interaction point at the end. As in the definition intervals, we need to specify the type of the box (i.e., `Process` or `Hierarchical`).

The execution of a box is performed by a REACTIVEML process (Figure 8). It first waits for the preceding intervals of the box are satisfied (line 3). Then, it executes the box depending of its type (line 4). Finally, once the box has finished, it launches its succeeding intervals (line 5). The above processes must be killed if the parent of the box is

stopped. In the following we describe the processes that decode each type of box.

```
1 let rec process run_generic box w_rels s_rels stop_f
      id_box =
2   (* .. *)
3   run (killable_p (wait_intervals w_rels id_box)
        stop_f);
4   run (run_box box);
5   run (killable_p (run_intervals s_rels) stop_f)
```

Figure 8. Process that executes a box.

A process box, implemented in the process `run_p_box` (Figure 9), first gets its identifier (`ident`), the interval that defines its duration (`interval`), and the OSC messages to start (`start_m`) and stop (`end_m`) the external process (line 2). Then, it starts the external process by sending the corresponding OSC message (line 4). Next, it runs the interval of its duration (line 5) and waits until it ends (line 6). Finally, once the box stops, it immediately sends the corresponding OSC message to stop the external process (line 7). The box and its external process finish suddenly if the signal `stop_f` is emitted (**`do/until`** construction) by its parent.

```
1 let process run_p_box p_box =
2   let (ident, interval,star_m,end_m) = p_box in
3   do
4     emit output (star_m);
5     (run (run_intervals [interval]) ||
6     run (wait_intervals [interval] ident));
7     emit output (end_m);
8   until stop_f -> emit output (end_m) done
```

Figure 9. Process that executes a process box.

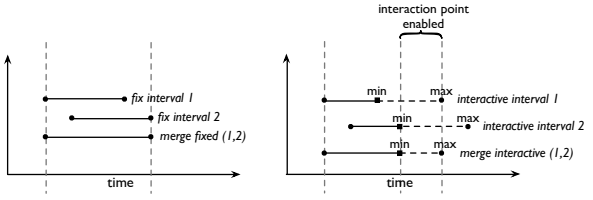
On the other hand, a hierarchical box is executed by the process `run_h_box` (Figure 10). It first gets the parameters of the box (line 2): its identifier (`ident`); its children (`boxes`); the temporal relations of the sub-scenario (`relations`); and the interval that defines its duration (`interval`). Then, it executes in parallel: a monitor that emits the signal `stop_box_h` when the parent of the box finishes suddenly (line 5); the interval that defines its duration (line 7); a monitor that emits the signal `stop_box_h` when the box stops because of an interaction point (line 12); the relations that describe the temporal organization of the subscenario (line 15); a monitor that waits for the relations defining its end (line 16); and its children with their preceding and succeeding intervals (line 18). Hence, the hierarchical box and its children will finish abruptly when the signals `stop_box_h` or `stop_f` are emitted. Otherwise, the hierarchical box will finish when its duration and all internal relations have finished.

3.2.3. Synchronization

Boxes can have one or more preceding and succeeding relations. In I-SCORE, all preceding relations of a box with an interaction point are flexible (interactive intervals). Otherwise, all are rigid (fixed intervals). In the first case, the box will start when one of its preceding relations has finished, and the interaction point will be enabled when they have reached their minimum duration (Figure 11b). In the second case, the box will start when all its preceding relations have finished (Figure 11a). As noted, we can merge a set of intervals into one that follows the behaviour described above. Next, we describe the processes to handle intervals.

```
1  let process run_h_box h_box =
2    let (ident, boxes, relations, interval) = h_box in
3    signal stop_box_h in
4    signal kill_m in
5    do (await immediate stop_f; emit stop_box_h) until
         kill_m done ||
6    (((do
7      run (run_intervals [interval]) ||
8      (run (wait_intervals [interval] ident);
9        begin
10         match interval with
11         | Fixed _ -> ()
12         | Interactive _ -> emit stop_box_h
13       end
14     ) ||
15     run (run_intervals (get_intervals ident relations
           From)) ||
16     run (wait_intervals (get_intervals ident relations
           To) ident)
17   until stop_box_h done); emit stop_box_h) ||
18   run (run_boxes_par boxes relations stop_box_h));
       emit kill_m
```

Figure 10. Process that executes a hierarchical box.



(a) Fixed intervals. (b) Interactive intervals.

Figure 11. Merging a set of intervals.

The process `run_intervals` (Figure 12) runs in parallel a list of intervals (line 3). Each interval emits a specific signal when it reaches its duration (line 7). In the case of an interactive interval, a different signal will be emitted when it reaches its minimum and maximum duration (line 12 and 14). Following the semantics described above, if a box has several preceding intervals, it will start when one of them finishes (**`do/until`** construction). We use the process iterator **`Rml_list`**`.par_iter` to execute in parallel a list of processes.

The process `wait_intervals` (Figure 13) waits for a set of intervals are satisfied. Then, in the case that all intervals are fixed, it waits until all intervals end (line 5 and 7). Otherwise, it first waits until all reach their minimum duration (line 5), and then it begins to listen the external events (line 13) until one interval reaches its maximum duration (**`do/until`** construction) or the event associated to the interaction point is triggered (line 14). The emission of the signal `max_s` also will stop all intervals.

```
let process run_intervals inter_l =
  (* .. *)
  run (Rml_list.par_iter
    (proc i ->
      match i with
        | Fixed (d,s) ->
          run (handle_duration d s)
        | Interactive (min,max,_) ->
          let (min_d,min_s) = min in
          let (max_d,max_s) = max in
          begin
            run (handle_duration min_d min_s);
            do
              run (handle_duration max_d max_s)
            until max_s done;
          end
    ) inter_l )
```

Figure 12. Process that runs a set of intervals.

```
let process wait_intervals inter_l id_box =
  (* .. *)
  if (List.length inter_l > 0) then
  begin
    run (sync_minimum inter_l);
    match (List.hd inter_l) with
    | Fixed (_) ->  ()
    | Interactive (_,max,ip) ->
      let (_,max_s) = max in
      begin
        do
          loop
            await input (ip_e) in
            (if (checkIP ip ip_e) then emit max_s);
            pause
          end
        until max_s done
      end
  end
```

Figure 13. Process that waits for a set of intervals.

3.3. Running an Example

In the following, we present two different executions of the scenario specified in Example 1 (Section 2.1) using our interpreter. In both executions the period of the clock was one second. We used PURE DATA to run the multimedia processes and trigger the interaction points by sending OSC messages.

In the first execution we only triggered at 23 seconds the interaction point at the start of the box 3. Comparing the log of execution (Figure 14) with the execution shown in Figure 4, we observe our implementation follows correctly the operational semantics of interactive scores. Recall that Figure 4 illustrates the execution of Example 1 under the same conditions.

On the other hand, in the second execution we started and stopped the box 3 at 10 and 17 seconds, respectively. We illustrate the execution of Example 1 under the above conditions in Figure 16. Note that the box 3 started early because the interaction point was triggered at 10 seconds. Furthermore, the children of the box 3 were stopped abruptly at 17 seconds because the parent was stopped by the interaction point. Comparing the log of execution (Figure 15) with the execution shown in Figure 16, we observe our implementation follows correctly the operational semantics of interactive scores.

```
Simulation log ...
==================

clock 0 -> (scenario started), (h_box 0 started).
clock 1 -> (p_box 2 started).
clock 2 -> .
clock 3 -> (p_box 1 started).
clock 4, 5 -> .
clock 6 -> (p_box 2 finished), (p_box 6 started).
clock 7 -> (p_box 1 finished).
clock 8 -> .
clock 9 -> (start listening ip 1).
clock 10, 11 -> .
clock 12 -> (stop listening ip 2), (h_box 3 started).
clock 13 -> .
clock 14 -> (p_box 4 started), (start listening ip 2).
clock 15 -> (p_box 5 started).
clock 16 -> (p_box 5 finished).
clock 17 -> (start listening ip 3).
clock 18, 19 -> .
clock 20 -> (stop listening ip 3), (p_box 4 finished).
clock 21 -> (p_box 6 finished).
clock 22 -> .
clock 23 -> (event ip 2 triggered), (stop listening ip
      2), (h_box 3 finished).
clock 24, 25 26 -> .
clock 27 -> (h_box 0 finished), (scenario finished).
```

Figure 14. Execution log of Example 1 where only the interaction point at the start of the box 3 was triggered.

```
Simulation log ...
==================

clock 0 -> (scenario started), (h_box 0 started).
clock 1 -> (p_box 2 started).
clock 2 -> .
clock 3 -> (p_box 1 started).
clock 4, 5 -> .
clock 6 -> (p_box 2 finished), (p_box 6 started).
clock 7 -> (p_box 1 finished).
clock 8 -> .
clock 9 -> (start listening ip 1).
clock 10 -> (event ip 1 triggered), (stop listening ip
      1), (h_box 3 started).
clock 11 -> .
clock 12 -> (p_box 4 started), (start listening ip 2).
clock 13 -> (p_box 5 started).
clock 14 -> (p_box 5 finished).
clock 15 -> (start listening ip 3).
clock 16 -> .
clock 17 -> (event ip 2 triggered), (stop listening ip
      2), (p_box 4 finished), (h_box 3 finished).
clock 18, 19, 20 -> .
clock 21 -> (p_box 6 finished), (h_box 0 finished), (
     scenario finished).
```

Figure 15. Execution log of Example 1 where both interaction points of the box 3 were triggered.

It is important to remark that unlike the Petri Net model presented in [2], our model allows to represent precisely the hierarchical behaviour of boxes. As can be seen in Figure 2, the hierarchical box represented by the transitions `start(3)` and `end(3)` only models the gathering of a set of boxes, but it does not model the forced stopping of its children due to an interaction point.

4. IMPROVING THE VISUALIZATION WITH INSCORE

Currently, the graphical interface of I-SCORE does not support a good feedback in real-time of the dynamic execu-

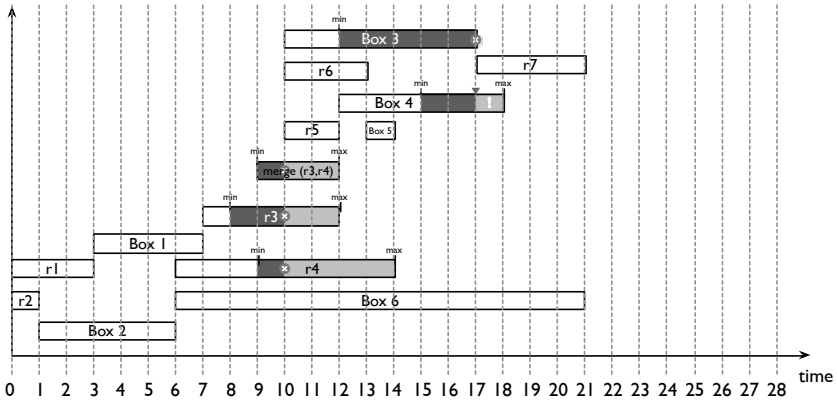


Figure 16. Execution of Example 1 where both interaction points of the box 3 are triggered.

tion of the scenario. In this section, we attempt to overcome this limitation by implementing a graphical interface that changes depending on the events emitted during the execution of a score.

Our approach consists in developing a visualization system in INSCORE [11] that behaves like a synchronous observer [13] of our interpreter (i.e., a process that listens the inputs and outputs of other process without altering its behaviour). INSCORE is a software for designing interactive and augmented scores. Here, scores are composed of heterogeneous graphic objects such as symbolic music notation, text, images, videos and files with a graphic and temporal dimension. Moreover, this tool integrates a message driven system that uses the OSC protocol in order to interact with any OSC application or device. Therefore, the graphical interface can dynamically transform depending on the messages.

Roughly, in our graphical interface (Figure 17) events can be triggered by clicking on the box. Single-click triggers the interaction point at the start of the box whereas double-click triggers the interaction point at the end. The performer knows that an interaction point can be triggered when the border of the box is either dashed (the interaction point at the start) or dotted (the interaction point at the end). A REACTIVEML process listens the events emitted by the interpreter and changes the organization and size of boxes in the graphical interface depending on them. Moreover, boxes change their colour when they are executing. The interface also shows the current time in the upper right of the scenario and indicates the current position of the execution with a vertical line.

4.1. Implementation

In the following we describe the implementation of the process `box`. This process listens the events emitted by the interpreter and dynamically transforms the graphical interface depending on them.

Roughly speaking, the process sends OSC messages to INSCORE according to both the events emitted by the interpreter and the current time of execution. For instance,

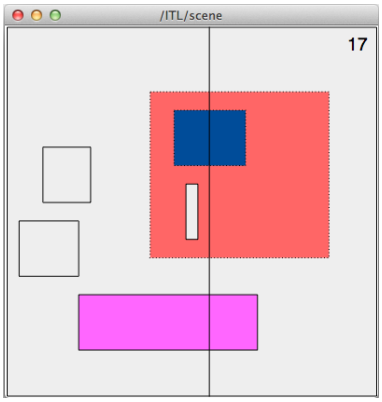


Figure 17. Graphical interface in INSCORE.

a box is moved to the right of the x-axis if the interpreter has not emitted its start event and its start date has elapsed. Additionally, we take advantage of the interaction capabilities of INSCORE to trigger the interaction points of the boxes directly from the graphical interface. Next, we describe in more detail the process `box` (Figure 18).

First, it draws the box in INSCORE by means of the function `draw_box` (line 2). This function also assigns INSCORE events to boxes in order to trigger their interaction points. Next, it verifies at each tick of clock if the box needs to move to the right of the x-axis (line 8) because it has not started and its start date has elapsed (i.e., the start date of the box is delayed). In the case of a hierarchical box, the children move along with the parent (line 10). Additionally, it checks if the interaction point at the start of the box is enabled. If that is true, the box changes its border to dashed (line 16).

Once the box starts (i.e., the interpreter emitted the start event of the box), it changes its colour (line 20) and returns to its original border (line 21). Then, the process verifies if the box started before its start date (line 22). In that case, the box is moved to the current position in the execution of the score (line 24). Next, the process tests at each tick of clock if the width of the box needs to be lengthened (line

33) because it has not stopped and its end date has elapsed (i.e., the end date of the box is delayed). In addition, it examines if the interaction point at the end of the box is enabled. If that is true, the box changes its border to dotted (line 39).

Once the box stops (i.e., the interpreter emitted the stop event of the box), it returns to its original colour (line 43) and border (line 44). Finally, the process verifies if the box stopped before its end date (line 46). In that case, the box is resized to the current position in the execution of the score (line 47). The box with identifier 0 is not resized because we do not want to resize the scenario.

```
1  let process box b =
2    draw_box id_b (!width *. dx) (get_x !pos_x) pos_y
         height ip_start ip_end;
3
4    (* waiting the start *)
5    do
6      loop
7        await immediate clock;
8        if (!pos_x < current) then begin
9          pos_x := !pos_x +. 1.0;
10         List.iter
11           (fun i -> move_box i dx; emit dx_s.(i) 1.0)
12           (id_b::children)
13       end;
14       pause
15     end ||
16     (await immediate start_ip.(id_b); change_border
           id_b "dash")
17   until start_s.(id_b) done;
18
19   (* box started *)
20   change_color id_b r g b ;
21   change_border id_b "solid";
22   if (!pos_x > current) then begin
23     let new_x = current -. !pos_x  in
24     List.iter
25       (fun i -> move_box i (new_x *. dx); emit dx_s.(i
             ) new_x)
26       (id_b::children)
27   end;
28
29   (* waiting the end *)
30   do
31     loop
32       await immediate clock;
33       if (current >= (!pos_x +. !width)) then begin
34         width := !width +. 1.0;
35         resize_box id_b (!width *. dx)
36       end;
37       pause
38     end ||
39     (await immediate start_ip.(id_b); change_border
           id_b "dot")
40   until end_s.(id_b) done;
41
42   (* box stopped *)
43   change_color id_b 238 238 238 ;
44   change_border id_b "solid";
45   let new_dy = (current -. !pos_x) in
46   width := if new_dy > 0.0 then new_dy else 0.0;
47   if (id_b <> 0) then  resize_box id_b (!width *. dx);
```

Figure 18. Process that encodes the behaviour of a box in the graphical interface.

Hence, each box shown in the graphical interface (i.e., INSCORE score) represents a process box that is running concurrently and interacting with other boxes of the graphical interface. Therefore, our visualization system allows performers to observe the current state of the execution of the score.

5. CONCLUDING REMARKS

In this work, we presented a synchronous interpreter of multimedia interactive scores. It was implemented in the synchronous programming language REACTIVEML [16]. We showed that the implementation is simple and small thanks to the synchronous model and high-order programming provided by REACTIVEML. Contrary to the Petri Net model presented in [17], our approach allows to model precisely the hierarchical behaviour of boxes.

We explored the use of INSCORE to develop a graphical interface that provides a real-time visualization of the execution of the score. In this sense, it improves the current graphical interface of I-SCORE. We took advantage of the OSC protocol to communicate our interpreter with external applications such as PURE DATA and INSCORE.

We believe that our implementation provides many advantages for the composition and execution of interactive scores. For instance, as shown in [5], we can prototype new features easily and execute living code using the toplevel of REACTIVEML [15]. Moreover, our approach would allow to execute dynamic processes unlike Petri Nets.

Related work. The work in [4, 5] embeds the ANTESFOCO language [9] and presents how to program mixed music in REACTIVEML. ANTESFOCO is a score following system that synchronizes in real-time electronic music scores with a live musician. The approach defines a synchronous semantics of the core language of ANTESFOCO, and then it is implemented in REACTIVEML. Therefore, composers can prototype new constructs and take advantage of the expressiveness of synchronous model and the power of functional programming. For example, recursion, high order programming, type induction, among others.

Future work. Multimedia interactive scores have a wide range of applications such as video games and museum installations. Therefore, in some cases, it is highly necessary to use conditions and loops in order to model the dynamics of the score easier and correctly. However, these constructions are not supported by the current model of I-SCORE. For this reason, we plan to take advantage of the features of REACTIVEML to prototype these new constructions.

Nowadays, composers have increasingly needed to manipulate streams in their multimedia scenarios. Then, we plan to examine the data-flow programming language LUCID SYNCHRONE [18] in order to handle streams in real-time. This programming language combines the synchronous model of LUSTRE [12] with the features of ML languages. In addition, we plan to perform tests to verify that the new solutions are more efficient than the current implementations.

We intend to increase the usability of our interpreter by developing a compiler that translates automatically scenarios designed in I-SCORE into the syntax of the interpreter. Additionally, we plan to improve the graphical interface in INSCORE in order to provide an environment

where the user can directly design and visualize the execution of a scenario.

Finally, we intend to verify properties of scenarios [14]. For instance, we are interested in knowing the maximum number of processes that can be executed in parallel during all possible executions of the scenario.

Acknowledgements. We thank the anonymous reviewers for their detailed comments that helped us to improve this paper. Also, we would like to thank Louis Mandel for his valuable remarks about the implementation. This work has been supported by the OSSIA (ANR-12-CORD-0024) project and SCRIME [6] .

6. REFERENCES

[1] Allen, J., « Maintaining knowledge about temporal intervals », *Communications of the ACM*, ACM, vol. 26 (11), New York, NY, USA, 1983, p. 832–843.

[2] Allombert, A., « Aspects temporels d'un système de partitions musicales interactives pour la composition et l'exécution », *Ph.D. Thesis*, Bordeaux, France, 2009.

[3] Allombert, A., Desainte-Catherine, M., and Assayag, G., « Iscore: A system for writing interaction », *Proceedings of the Third International Conference on Digital Interactive Media in Entertainment and Arts*, ACM, New York, NY, USA, 2008, p. 360–367.

[4] Baudart, G., Jacquemard, F., Mandel, L., and Pouzet, M., « A synchronous embedding of Antescofo, a domain-specific language for interactive mixed music », *Proceedings of the Thirteen International Conference on Embedded Software*, Montreal, Canada, 2013.

[5] Baudart, G., Mandel, L., and Pouzet, M., « Programming mixed music in ReactiveML », *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling*, Boston, USA, 2013.

[6] Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., and De Simone, R., « The synchronous languages 12 years later », *Proceedings of the IEEE*, IEEE, vol. 91 (1), 2003, p. 64–83.

[7] Berry, G., and Gonthier, G., « The Esterel synchronous programming language, design, semantics, implementation », *Science of Computer Programming*, Elsevier, vol. 19 (2), Amsterdam, The Netherlands, 1992, p. 87–152.

[8] Boussinot, F., and De Simone, R., « The SL synchronous language », *IEEE Transactions on Software Engineering*, IEEE Press, vol. 22 (4), Piscataway, USA, 1996, p. 256–266.

[9] Cont, A., « ANTESCOFO: Anticipatory synchronization and control of interactive parameters in computer music », *Proceedings of International Computer Music Conference*, Belfast, Ireland, 2008.

[10] Desainte-Catherine, M., Allombert, A., and Assayag, G., « Towards a hybrid temporal paradigm for musical composition and performance: The case of musical interpretation », *Computer Music Journal*, MIT Press, vol. 37 (2), Cambridge, MA, USA, 2013, p. 61–72.

[11] Fober, D., Orlarey, Y., and Letz, S., « An environment for the design of live music scores », *Proceedings of the Linux Audio Conference*, CCRMA, Stanford University, California, US, 2012, p. 47–54.

[12] Halbwachs, N., Caspi, P., Raymond, P., and Pilaud, D., « The synchronous dataflow programming language LUSTRE », *Proceedings of the IEEE*, IEEE, vol. 79 (9), 1991, p. 1305—1320.

[13] Halbwachs, N., Lagnier, F., and Raymond, P., « Synchronous observers and the verification of reactive systems », *Proceedings of the Third International Conference on Methodology and Software Technology*, Springer-Verlag, London, UK, 1994, p. 83–96.

[14] Halbwachs, N., and Raymond, P., « Validation of synchronous reactive systems: From formal verification to automatic testing », *Proceedings of the Fifth Asian Computing Science Conference on Advances in Computing Science*, Springer-Verlag, 1999, p. 1–12.

[15] Mandel, L., and Plateau, F., « Interactive programming of reactive systems », *Electronic Notes in Theoretical Computer Science*, Elsevier, vol. 238 (1), Amsterdam, The Netherlands, 2009, p. 21–36.

[16] Mandel, L., and Pouzet, M., « ReactiveML, a reactive extension to ML », *Proceedings of the Seventh ACM SIGPLAN International Symposium on Principles and Practice of Declarative Programming*, Lisbon, Portugal, 2005.

[17] Marczak, R., Desainte-Catherine, M., and Allombert, A., « Real-time temporal control of musical processes », *The Third International Conferences on Advances in Multimedia*, Budapest, Hungary, 2011, p. 12–17.

[18] Pouzet, M., « Lucid Synchrone, version 3. Tutorial and reference manual », *http://www.di.ens.fr/~pouzet/lucid-synchrone/lucid-synchrone-3.0-manual.pdf*.

[19] Sénac, P., De Saqui-Sannes, P., and Willrich, R., « Hierarchical Time Stream Petri Net: A model for hypermedia systems », *Proceedings of the Sixteenth International Conference on Application and Theory of Petri Nets*, Springer, Turin, Italy, 1995, p. 451–470.

[6] http://scrime.labri.fr

PARTITIONS RETROACTIVES AVEC IANNIX

Guillaume Jacquemin
Association IanniX
guillaume@iannix.org

Thierry Coduys
Association IanniX
thierry@iannix.org

RÉSUMÉ

Le logiciel IanniX offre la possibilité d'écrire et de composer des partitions exécutées en temps réel qui émettent des messages de contrôle — synthèse, effets, etc. — et reçoivent des messages de structure — ajout, altération d'objets —. La combinaison de ces entrées/sorties et des interfaces disponibles *(scripts, réseau, interface graphique)* permet de composer des partitions de contrôle de paramètres, des partitions réactives à des stimuli, des partitions stochastiques, génératives ou encore interactives où IanniX réalise des *mappings* complexes et temporels.

Les partitions rétroactives développées dans cet article représentent un nouveau champ de composition, où la sortie de la partition est bouclée sur son entrée. Elle entre parfois en résonance, parfois s'emballe mais reproduit globalement les phénomènes observés dans les systèmes bouclés.

L'article présente d'abord l'état des lieux de ce phénomène graphique sonifié par IanniX pour ensuite le proposer comme un nouveau processus génératif d'écriture musicale.

L'article se conclut sur une première réalisation, *Singularités*, développée avec ce mécanisme en utilisant un robot industriel ainsi que le synthétiseur Cosmos*f*.

1. IANNIX

1.1. Objets fondamentaux pour la composition

IanniX est un séquenceur graphique open source, inspiré des travaux de Iannis Xenakis et destiné à la création numérique [1]. Le logiciel propose une écriture polytemporelle d'évènements statiques (cues, déclencheurs, etc.) et dynamiques (variation de valeurs, automations, etc.) destinés à des environnement dédiés (Processing, PureData, SuperCollider, Max…).

Pour composer une partition graphique dans IanniX, une palette restreinte de trois objets fondamentaux et distincts est disponible :

- les *triggers* qui déclenchent des événements statiques (ex : déclenchement d'un fichier son) ;
- les *courbes*, évènements dynamiques qui sont des suites de points dans l'espace tridimensionnel (ex : changement d'un paramètre de synthèse) ;
- les *curseurs* qui évoluent sur des courbes et progressent en fonction du temps.

Ces objets émettent des messages (les *triggers* en émettent lorsqu'ils sont déclenchés, les *curseurs* en émettent périodiquement lorsqu'ils progressent sur une courbe) qui sont envoyés via des *interfaces* vers d'autres applications ou matériels.

1.2. Simplification des interfaces

L'écriture des partitions dans IanniX s'effectue via des *messages* envoyés dans des *interfaces* [2] parmi lesquelles figurent :

- **l'interface graphique utilisateur** (GUI),
- **le réseau** : Open Sound Control (OSC) [3] et UDP brut, pour les applications temps réel ; MIDI, pour les contextes de contrôle simples ou pour l'interfaçage sur des DAW ; RS232, pour le contrôle de hardware ; HTTP GET, pour le contrôle de pages Web ; WebSockets pour l'interfaçage en HTML5,
- **les scripts** JavaScript basés sur la norme ECMAScript (norme ECMA-262 [4]),
- **la boucle locale** qui permet aux partitions de s'envoyer des messages.

L'écriture de partitions repose donc sur l'échange de messages via ces protocoles *(y compris lorsque l'utilisateur manipule l'interface graphique, des messages OSC sont automatiquement générés en interne, de manière transparente pour l'utilisateur)*. Cependant, par souci de clarté, de documentation et aussi pour faciliter l'interfaçage avec des outils tiers, une nouvelle fenêtre, le *Helper*, présenté en figure 1, affiche les messages venant de l'interface graphique et permet de copier dans le presse-papier des snippets pour Processing et Max qui reproduisent l'opération réalisée manuellement dans l'interface graphique.

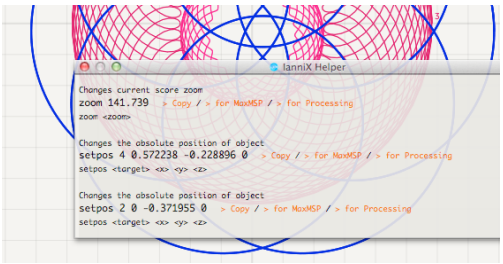


Figure 1. Capture d'écran de IanniX de la fenêtre *Helper* montrant les messages générés par

l'interface graphique utilisateur, ici un zoom, puis deux déplacements d'objets.

Enfin, l'architecture de IanniX 0.9 [5] a aussi été repensée et optimisée pour recevoir et envoyer massivement des messages.

1.3. Conséquences sur la sauvegarde des partitions

La multiplicité des interfaces de création et d'altération de partitions dans IanniX a soulevé le problème de la sauvegarde en fichier.

Prenons par exemple une partition créée dans l'interface JavaScript, sur laquelle le compositeur ajoute et modifie quelques éléments via l'interface graphique et finalement décide de rendre la partition interactive avec des capteurs [6]. Lorsque tous ces facteurs (du code, des modifications dans l'interface graphique et des données de capteurs) altèrent ou génèrent la partition, une simple sauvegarde de l'état de la partition est insuffisante. Par exemple, si le rôle d'un des capteurs est de supprimer une partie de la partition, la sauvegarde après la performance sera inéluctablement partielle…

IanniX propose donc une solution en unifiant le format des sauvegardes : tout document IanniX est désormais un script JavaScript, même si la partition est par exemple créée uniquement par l'interface graphique. Le code généré par IanniX et par l'utilisateur est ensuite ventilé dans quatre procédures en fonction de l'origine des modifications *(procédures appelées au chargement dans l'ordre ci-dessous)* :

- **makeWithScript()** : code JavaScript utilisateur ;
- **madeThroughGUI()** : section auto-générée par IanniX retranscrivant les ajouts et modifications réalisés dans l'interface graphique ;
- **madeThroughInterfaces()** : section auto-générée par IanniX retranscrivant les ajouts et modifications réalisés par des capteurs ou des interfaces réseau
- **alterateWithScript()** : code JavaScript utilisateur permettant d'altérer l'ensemble de la partition juste avant la fin du chargement.

Une rétrocompatibilité est évidemment maintenue pour l'ouverture de documents IanniX créés avant la version 0.9.

2. CLASSIFICATION DES PARTITIONS

Le champ des possibilités d'écriture ouvertes par les messages et protocoles de IanniX peut se classifier selon l'évolution de la partition au fil de son interprétation. Évidemment, le compositeur est amené à hybrider ces méthodes d'écriture dans la conception de son œuvre ; cette classification vise surtout à comprendre les grandeurs mises en jeu au niveau des entrées, des sorties et des interactions avec un environnement.

2.1. Partition de contrôle

Une **partition de contrôle** est une partition qui contrôle une ou plusieurs applications tierces. Elle **ne répond à aucun stimulus** externe. L'interprétation de la partition est autonome, reproductible et déterministe. Ce type de partition est très proche d'une surface de contrôle MIDI.

Ce type de partition permet par exemple d'écrire des courbes de contrôle d'effets, de synthèse (à la manière de l'UPIC) ou de spatialisation (figure 2).

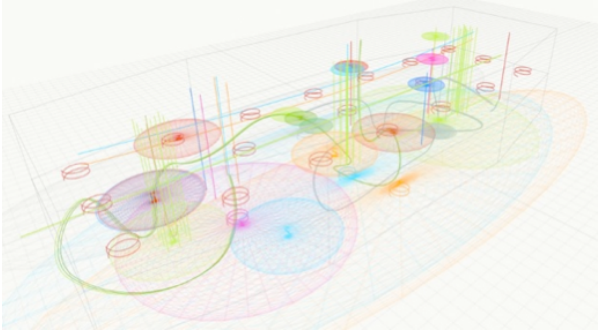


Figure 2. Capture d'écran de IanniX de la partition de contrôle de spatialisation pour *World Expo* de *Charles de Meaux ;* chaque courbe représente le déplacement d'une source sonore.

2.2. Partition réactive

Une **partition réactive** est une partition qui **réagit à des stimuli externes** mais qui ne produit aucun message de contrôle. Elle est interprétée par un humain qui la lit et/ou remplit une fonction esthétique et graphique.

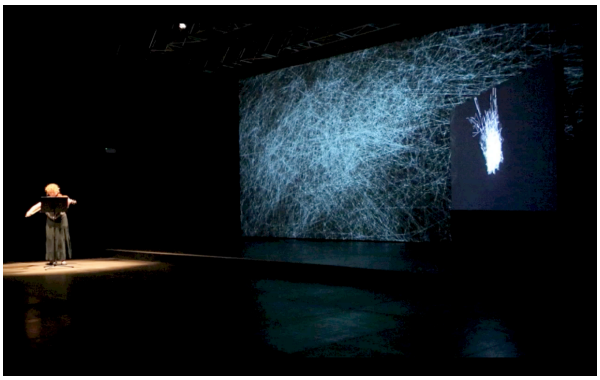


Figure 3. Capture d'écran de IanniX de la partition réactive esthétique pour *Influences* de *Davide Gagliardi* et *Victor Nebbiolo di Castri* ; les courbes sont générées en réponse à l'alto sur scène.

2.3. Partition stochastique

La **partition stochastique** est une partition dont le processus global est prévisible, même si les évènements qui la composent sont aléatoires.

L'usage du JavaScript dans IanniX permet d'écrire les règles de contrôle et de distribution stochastiques qui régissent les grandeurs aléatoires de la partition. Associés à des librairies JavaScript externes, les script IanniX permettent d'écrire rapidement des tirages aléatoires sous contraintes, des implémentations de la théorie des jeux ou des fonctions de distribution [7].

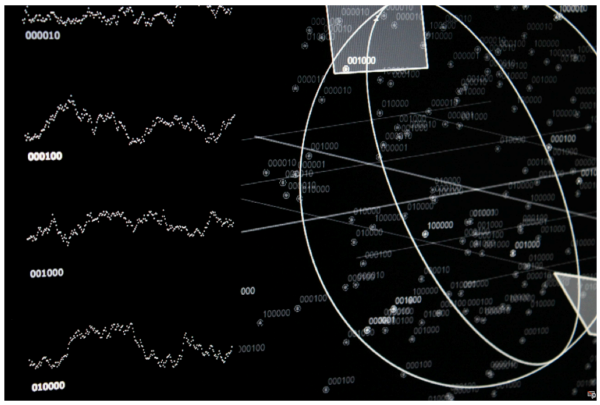


Figure 4. Capture d'écran de IanniX de la partition stochastique pour *Ultimarium* de *Thomas Bouaziz* ; la partition effectue des tirages aléatoires d'hexagrammes Yi King.

2.4. Partition générative

Une **partition générative** est une partition générée par des algorithmes, qui peut également évoluer d'elle-même de manière déterminée à l'avance ou non.

Tout comme les partitions stochastiques, le JavaScript offre une liberté d'écriture générative (figure 5) et permet aussi de reproduire des phénomènes biologiques grâce à des librairies externes dédiées (figure 6).

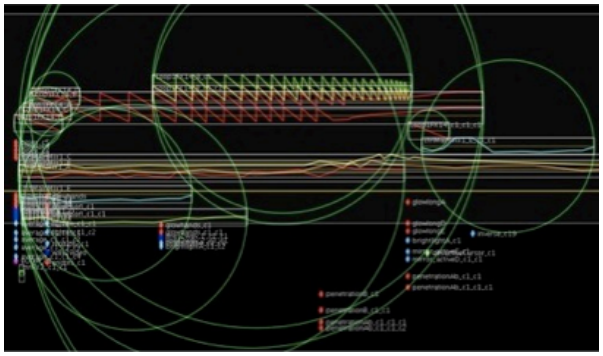


Figure 5. Capture d'écran de IanniX de la partition générative pour *Eros3* de *Joachim Montessuis* ; la partition n'évolue pas au fil du temps.

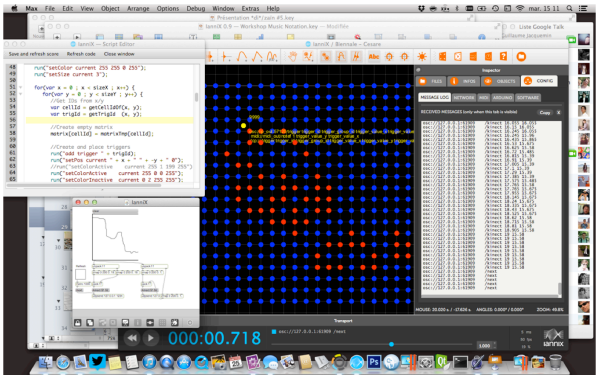


Figure 6. Capture d'écran de IanniX de la partition générative pour *Still Life* de *Cesare Saldicco* ; modèle génératif évolutif reproduisant la propagation des virus.

2.5. Partition interactive

Une **partition interactive** fait intervenir la coopération de IanniX avec plusieurs entités (hommes ou applications) qui agissent mutuellement en ajustant leur comportement.

IanniX est alors l'intermédiaire entre plusieurs entités et régule, par l'écriture et la composition, les interactions entre ces entités. IanniX agit comme un dispositif classique de *mapping* (transformation d'une grandeur vers une autre) mais introduit une dimension de composition et d'écriture du temps fondamentale.

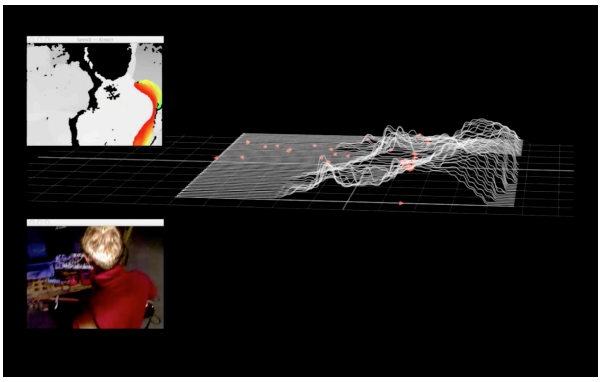


Figure 7. Capture d'écran de IanniX de la partition interactive pour *Fa Octothorp* de *Guillaume Jacquemin* et *Matthieu Ranc* ; une caméra Kinect capte une image 3D qui extrude un maillage de courbes dans IanniX sur lesquelles circulent des curseurs contrôlant la synthèse sonore.

2.6. Partition rétroactive

Une **partition rétroactive ou bouclée** est une partition dont l'interface de sortie est connectée directement à l'entrée, avec un ou plusieurs systèmes insérés dans la boucle.

Les valeurs émises par les objets de la partition à l'instant *t* contrôlent la structure et les objets de la

partition à l'instant *t+1*. Le délai de rétroaction entre deux itérations de calcul est initialement réglé à 5 ms dans IanniX mais reste modifiable par l'utilisateur (de 1 ms à 1 seconde).

Tout comme les systèmes bouclés, l'évolution de la partition *(répétition itérative en fonction du temps)* peut aboutir à plusieurs résultats.

2.6.1. Rudiments de code IanniX / JavaScript

Les partitions présentées dans la suite de l'article ont été écrites en JavaScript. Il convient alors de donner quelques rudiments de syntaxe.

IanniX respecte la norme JavaScript ECMA-262 ainsi que ses classes de base (Math, String…) auquel on ajoute une fonction spéciale permettant d'envoyer un message à IanniX : `run()`.

Les messages IanniX permettant de créer, modifier ou supprimer un élément de la partition respectent systématiquement la syntaxe : `<action> <ID objet> <paramètres>`.

Ainsi la commande `run("add curve 1")` permet créer une courbe qui aura l'ID #1 ; la commande `run("setPointAt 23 0 5 8");` permet de modifier le premier point (index n°0) de la courbe #23 et de le placer à la coordonnées 2D (5 ; 8).

2.6.2. Amplification continuelle ou extinction progressive

Dans le code suivant, le curseur fait évoluer en fonction de sa position, le point terminal de la courbe sur laquelle il progresse :

```
run("add curve 1");
run("setPointAt lastCurve 0 0 0");
run("setPointAt lastCurve 1 1 1");
run("add cursor 2");
run("setCurve current lastCurve");
run("setMessage current direct:// setPointAt lastCurve
      1 1 {cursor_yPos+1}");
```

La figure 8 montre que la partition s'étire verticalement au fil du temps tandis que la figure 9 (extraite à l'aide OSCulator) montre que l'agrandissement est linéaire.

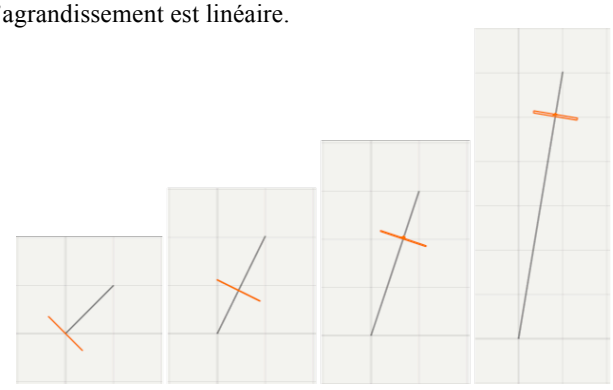


Figure 8. Captures d'écran de IanniX montrant le phénomène d'amplification continuelle à t = 0 sec., t = 1 sec., t = 2 sec. et t = 5 sec.

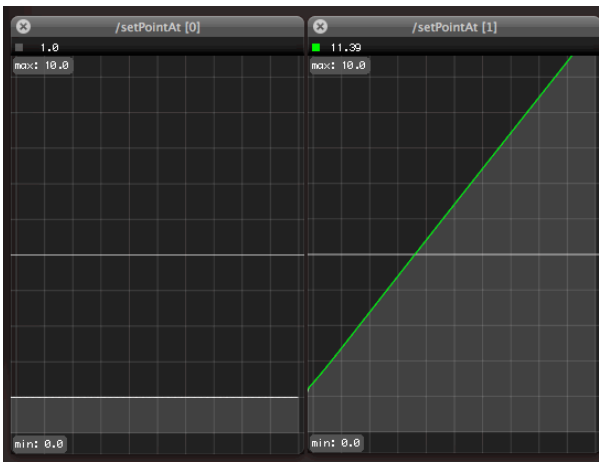


Figure 9. Progression linéaire de l'amplification (évolution des coordonnées *x* et *y* du point).

2.6.3. Emballement

Dans la partition suivante codée en JavaScript, le curseur fait évoluer en fonction de sa position, la taille de l'ellipse sur laquelle il progresse :

```
run("add curve 1");
run("setPointsEllipse lastCurve 1 1"); //Rayon = 1

run("add cursor 2");
run("setCurve current lastCurve");
run("setPattern current 0 0 1"); //Joue en boucle
run("setMessage current direct:// setResize lastCurve
      {cursor_xPos+1} {cursor_yPos+1}");
```

Un emballement va se produire car la figure va être redimensionnée dans l'espace des réels négatifs, et va donc se retourner (figure 10) ; de fait, le curseur ira également à rebours et la partition va entrer dans une résonnance non contrôlée (figure 11).

Figure 10. Captures d'écran de IanniX montrant les déformations de la figure dues à l'emballement du système

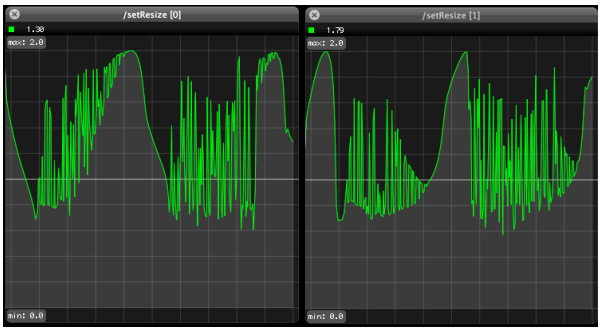


Figure 11. Emballement de la partition (visualisation du redimensionnement en *x* et en *y*).

2.6.4. *Régulation stable*

Pour cette dernière partition rétroactive, nous allons placer deux cercles avec deux curseurs. Le curseur du *cercle #1* contrôle le diamètre du *cercle #2* ; et le curseur du *cercle #2* contrôle le diamètre du *cercle #1*. Avec certaines conditions initiales *(c'est à dire le diamètre et la position initiale des cercles)*, les cercles entrent graphiquement en oscillation *(code fourni en annexe pour n cercles)*. La figure 12 propose quelques extraits graphiques de la partition tandis que la figure 13 montre l'évolution des rayons des cercles à l'aide d'OSCulator.

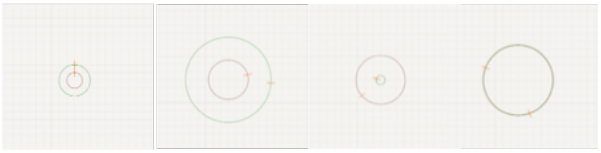


Figure 12. Régulation stable de la partition rétroactive, les cercles entre en oscillation / capture d'écran de IanniX

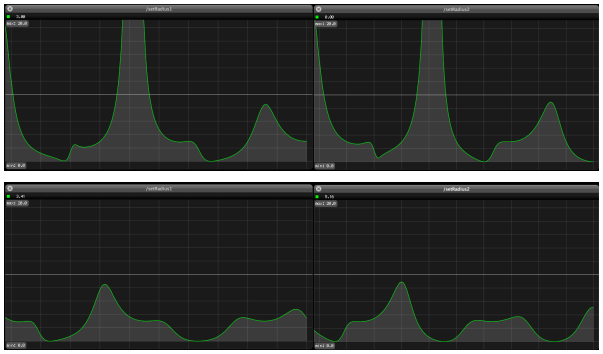


Figure 13. Régulation stable de la partition (visualisation des rayons à deux instants donnés).

2.6.5. *Du phénomène observé au processus d'écriture génératif*

Récurrences a été la première œuvre composée avec IanniX qui utilisait la rétroactivité. La pièce utilise six jeux de 8 courbes + curseurs qui contrôlent chacun un oscillateur indépendant. Chaque curseur d'un jeu de courbes contrôle l'amplitude et la durée de la courbe du jeu de courbes suivant.

La pièce a été jouée au centre DATABAZ d'Angoulême et la partition est distribuée librement avec l'application IanniX.

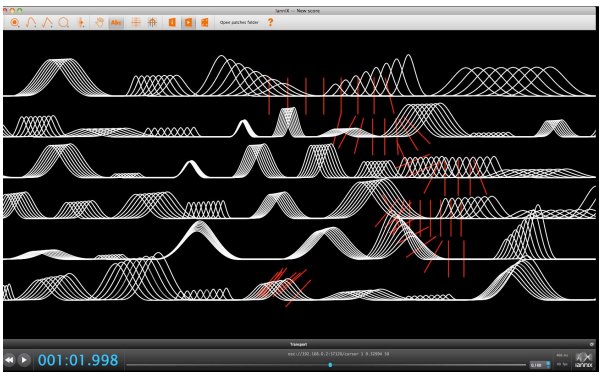


Figure 14. Capture d'écran de IanniX de la partition rétroactive *Récurrences* de *Thierry Coduys*.

3. SINGULARITES

3.1. Contexte

Les recherches menées sur les partitions rétroactives nous ont conduits naturellement aux systèmes bouclés et aux asservissements. Dans l'optique de créer une pièce dédiée à ces mécanismes, deux pistes nous ont paru intéressantes :

- l'asservissement dans le domaine de la robotique industrielle ;
- le feedback utilisé en synthèse sonore.

Après plusieurs séances de travail avec le compositeur *Sinan Bökesoy* (concepteur du synthétiseur stochastique inspiré des travaux de Iannis Xenakis, Cosmos*ƒ* [8] et auteur d'une publication sur l'utilisation des robots industriels en performance artistique [9]), une résidence autofinancée à *Büyükada* en Turquie s'est tenue du 29 juillet au 12 août 2013 afin d'esquisser les grands principes de la pièce et ses enjeux majeurs.

Figure 15. Capture d'écran du plugin Cosmos*ƒ*.

3.2. Intentions

La pièce *Singularités* amorcée lors de cette résidence va exploiter le principe de *singularité*.

En mathématiques, une **singularité** est un point, une valeur, dans lequel un objet mathématique n'est pas défini, par exemple une valeur où une fonction d'une variable réelle devient infinie.

En physique, une **singularité gravitationnelle** est un point spécial de l'espace-temps au voisinage duquel certaines quantités écrivant le champ gravitationnel deviennent infinies.

La **singularité technologique** est un point hypothétique de l'évolution technologique où il n'y a plus de progrès mais une explosion exponentielle de la science et des techniques.

En robotique, **les singularités** sont des points de l'espace que le robot ne peut atteindre. Contraint par ses moteurs et la rigidité de ses axes, le robot essaye d'atteindre les positions spatiales données par son opérateur tout en évitant les postures impossibles (les singularités). Ainsi, en donnant comme consigne au robot, des positions aux voisinages des singularités, il est contraint d'emprunter des itinéraires très complexes.

Ces singularités se retrouvant également dans les partitions rétroactives de IanniX (points, valeurs ou structures graphiques qui entraînent un emballement graphique de la partition), un dispositif très simple a été imaginé :

- IanniX prendra le rôle de l'opérateur et contrôlera les mouvements d'un robot (chorégraphie) ;
- le robot industriel ABB IRB120 sera équipé de capteurs embarqués ;
- Cosmos*f* sera utilisé pour la synthèse sonore en temps réel.

Figure 16. Robot industriel ABB-IRB120.

La rétroaction sera la suivante :

1. IanniX pilote le robot et donne des ordres de positions absolues,
2. le robot calcule les itinéraires pour atteindre ce point tout en évitant les singularités,
3. les capteurs placés sur le robot mesurent les rotations, accélérations, mouvements des axes (figure 17) et envoient ces informations vers Cosmos*f* qui va sonifier les mouvements du robot,
4. les mesures des capteurs du robot sont également renvoyées vers IanniX et altèrent la partition initiale, bouclant le système en rétroactivité.

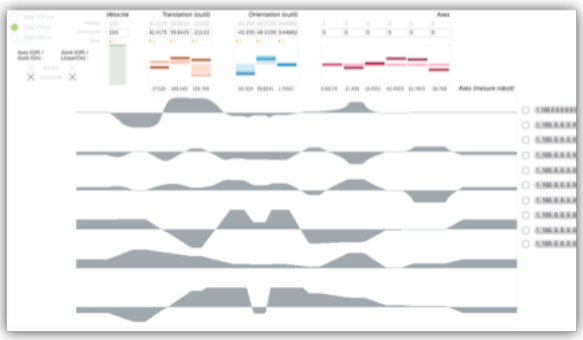


Figure 17. Mesures temps réel des capteurs sur le robot pour cinq consignes (déplacements) données au robot.

3.3. Prototypage

Dans le cadre du prototypage de la pièce, une caméra et un laser placés sur le robot permettent également d'obtenir un retour vidéo (caméra) et une projection du mouvement du robot sur un plan (laser).

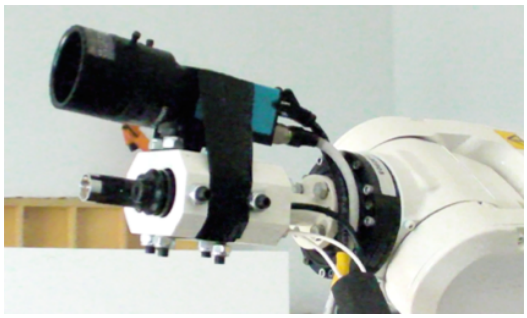


Figure 18. Caméra et laser sur la « main » du robot.

Enfin, le fonctionnement du robot étant extrêmement bruyant, l'outil RoKiSim [10] a permis afin de simuler et d'identifier les singularités, et ainsi d'écrire des partitions sans tester systématiquement avec le robot.

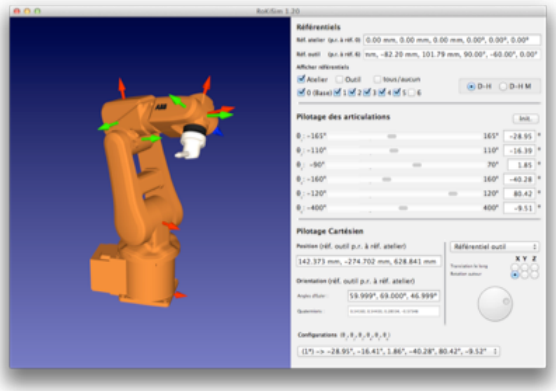


Figure 19. Logiciel *RoKiSim* permettant de simuler les comportements et singularités du robot.

4. CONCLUSION

Les partitions rétroactives ont été envisagées dans IanniX depuis quelques années. Elles permettent un auto-contrôle de la partition allant du simple événement rétroactif *(trigger qui arrête la partition ou saute à un timecode précis)* jusqu'à la mise en oscillation complexe des éléments constitutifs de la partition.

Au départ mal comprises, les partitions rétroactives constituent pour l'équipe de recherche IanniX une formidable opportunité de trouver de nouvelles approches dans la composition, même si l'écriture de telles partitions reste encore difficile à appréhender.

Au travers de l'amorce de la pièce *Singularités*, un premier socle prometteur [11] a été produit et la viabilité technique et esthétique est prouvée. L'environnement de travail étant économiquement lourd à cause du robot, des partenariats et des financements sont en cours.

5. REFERENCES

[1] Coduys, T. et Ferry G., "IanniX, aesthetical / symbolic visualisations for hypermedia composition", Sound and Music Computing, 2004

[2] Jacquemin, G., Coduys T. et Ranc M., "IanniX 0.8", Journées d'Informatique Musicale, Mons, 2012

[3] Wright, M. et Freed, A. "Open Sound Control: A New Protocol for Communicating with Sound Synthesizers" Proceedings of the International Computer Music Conference 1997, Thessaloniki, Hellas, pp. 101-104.

[4] Norme ECMA-262 http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[5] IanniX 0.9, http://www.iannix.org

[6] Jacquemin, G. et Coduys T., « Simone Beneventi + IanniX », 56ème Biennale de Venise, octobre 2012, http://www.labiennale.org/en/mediacenter/video/beneventi-int.html

[7] RandomJS, http://simjs.com/random.html

[8] Bökesoy S., "Synthesis of a macro sound structure within a self-organizing system", DAFX07 (Digital Audio FX Conference), Bordeaux 2007

[9] Bökesoy S., "1city 1001vibrations: development of a interactive sound installation with robotic instrument performance. ", NIME, Oslo 2011

[10] RoKiSim, http://parallemic.org/RoKiSim.html

[11] Vidéo du projet *Singularités*, http://vimeo.com/76981622

ANNEXE

Code de la partition rétroactive stable pour *n* cercles

```
function makeWithScript () {
   run("clear");

   var iMax = 2;
   for(var i = 0 ; i < iMax ; i++) {
       run("add curve " + (100+i));
       run("setPos current 3 3 0");
       run("setEquation current polar radius, TWO_PI*t,
              theta");
       run("setEquationParam current radius " + (1+i));
       run("setEquationParam current theta  0");
       run("setColorHue current " + map(i, 0, iMax, 0, 255)
              + " 255 128 255");

       run("add cursor " + i);
       run("setSpeed   current lock " + map(i, 0, iMax, 0.2,
              0.3));
       run("setCurve   current lastCurve");
       run("setPattern current 0 0 1");
       run("setBoundsSourceMode current 2");
       run("setMessage current 5, direct:// setEquationParam
              " + (101+i) + " radius cursor_xPos ");
   }
   run("setMessage current 5, direct:// setEquationParam " +
              100 + " radius cursor_yPos");
}
```

DÉPLOIEMENT DES SERVICES DU MOTEUR DE RENDU DE PARTITIONS GUIDO SUR INTERNET

M. Solomon, D. Fober, Y. Orlarey, S. Letz
Grame
Centre national de création musicale
mike@mikesolomon.org - {fober, orlarey, letz}@grame.fr

Résumé

La librairie Guido embarque un moteur de rendu de partitions musicales et fournit une API de haut niveau pour un ensemble de services liés au rendu de partition. Cette librairie est désormais embarquée dans un serveur HTTP qui permet d'accéder à son interface par le biais d'identifiants de ressource uniformes (URI). Après avoir décrit le style d'architecture *representational state transfer* (REST) sur lequel repose le serveur, l'article montre comment l'API C/C++ de la librairie est rendue accessible sous forme de requêtes HTTP.

1. L'ÉDITION MUSICALE SUR INTERNET

Pendant les dix dernières années, plusieurs outils de gravure musicale ont été développés selon un modèle client-serveur, à destination d'utilisateurs finaux et d'usagers du *cloud computing*. Le serveur GUIDO allie la gravure musicale sur Internet aux principes de l'architecture REST (élaborés dans la section 2) afin d'exposer l'API de la librairie Guido [8] [10] [4] par le biais d'URIs. Après un survol des outils d'édition musicale en ligne, nous dégagerons les thématiques sur lesquelles se basent les services actuellement disponibles et nous situerons le serveur Guido dans le paysage de la gravure musicale sur Internet.

1.1. Editeurs de musique sur Internet

Il y a actuellement trois outils majeurs d'édition musicale sur le web – Noteflight [1], Melodus [2] et Score [3]. Noteflight et Melodus fournissent un environment immersif d'édition musicale sur Internet comparable à Finale ou Sibelius. Scorio est un outil hybride qui utilise des algorithmes de mise en page élémentaires pour sa plate-forme mobile et propose le téléchargement des documents de haute qualité compilés avec GNU LilyPond.

1. http://www.noteflight.com
2. http://www.melod.us
3. https://scorio.com

1.2. Outils de partage de partitions sur Internet

Plusieurs outils musicaux dont Sibelius [4], MuseScore [1], Maestro [5] et Capriccio [6], proposent des services de partage sur Internet des partitions créées à partir de ces logiciels. Ces services permettent le téléchargement, la lecture et parfois le rendu sonore de ces partitions. Capriccio propose une application web écrite en Java qui reproduit les aspects essentiels de son éditeur principal alors que MuseScore, Sibelius et Maestro permettent la synchronisation automatique des partitions avec leurs représentations MIDI.

1.3. Services de compilation JIT sur Internet

WebLily [7], LilyBin [8] et OMET [9] proposent tous la compilation JIT de partitions LilyPond qui sont visualisées en SVG, PDF et/ou canvas HTML 5, selon l'outil. Le "*Guido Note Server*" [11] utilise le moteur Guido pour compiler des documents au format GMN (Guido Music Notation Format) [9] et calculer des images PNG.

1.4. Une alternative RESTful

Tous les outils exposés ci-dessous permettent la création et la visualisation de partitions par le biais de différentes méthodes d'entrée (représentation textuelle, MusicXML, etc.). En revanche, ils ne sont pas adaptés à des échanges d'information dynamiques entre un serveur et un client, du fait qu'ils ne proposent pas d'API publique et ne fournissent pas d'information au-delà d'une représentation graphique de la partition. Le serveur Guido résout ce problème en fournissant une API HTTP, qui agit comme une passerelle entre le client HTTP et l'API C/C++ de la librairie. Le serveur fournit donc aussi bien des représentations graphiques de la partition qu'un ensemble d'information liées à la partition : nombre pages, durée, répartition des éléments musicaux sur la page

4. http://www.sibelius.com
5. http://www.musicaleditor.com
6. http://cdefgabc.com
7. http://weblily.net
8. http://www.lilybin.com
9. http://www.omet.ca

et dans le temps, représentation MIDI... Le serveur GUIDO comble donc une lacune dans le paysage des outils de gravure musicale sur Internet en fournissnt à la fois des services de rendu de partition et en permettant le déploiement d'applications basées sur ces services.

2. REPRESENTATIONAL STATE TRANSFER

Le serveur GUIDO est basé sur une architecture de type REST (*REpresentational State Transfer*) [5]. Ce type d'architecture est largement utilisé pour l'élaboration de services Internet et repose sur un modèle client-serveur classique, « orienté resource », c'est-à-dire que le fonctionnement du serveur est optimisé pour la transmission d'informations relatives à des *ressources* [12]. Une des caractéristiques essentielles d'un serveur REST repose sur un fonctionnement *sans état* : toutes les informations requises pour traiter une requête se trouvent dans la requête elle même. Cela signifie, par exemple, qu'une ressource sur le serveur ne peut jamais être modifiée par un client – une modification équivaut au remplacement d'une ressource par une autre.

Plusieurs conventions sont proposés pour structurer les requêtes en forme d'URI afin qu'elles soient plus faciles à construire et décoder. Pour accélérer les interactions entre le serveur et le client, l'architecture REST permet la mise en cache de certains éléments, tant du côté du client que serveur. Le serveur doit proposer une interface uniforme en harmonisant sa syntaxe d'accès et en proposant les mêmes services à tous les clients qui l'utilisent.

Un serveur qui met en œuvre les recommandations REST s'appelle un serveur RESTful.

3. LE SERVEUR GUIDO HTTP

Le serveur GUIDO est un serveur RESTful qui compile des documents au format GMN (Guido Music Notation) et renvoie différentes représentations de la partition calculée à partir du code GMN. Il accepte des requêtes par le biais de deux méthodes du protocole HTTP : POST pour envoyer des partitions au format GMN au serveur, et GET pour interroger ces partitions ou pour en demander différentes représentations.

3.1. La méthode POST

L'implémentation de POST dans le serveur GUIDO est RESTful dans la mesure où il ne garde pas d'information par rapport aux clients et ne stocke que les documents GMN qu'on lui envoie. Supposons qu'un serveur GUIDO HTTP fonctionne sur le site `http://guido.grame.fr` sur le port 8000, une requête POST avec le code GMN `[a b c d]` pourrait être envoyée de la manière suivante :

```
curl -d"data=[a b c d]" http://guido.grame.fr:8000
```

Pour un code GMN valide, une réponse est renvoyée au format JSON [3], avec un identifiant unique qui correspond au code GMN.

```
{
  "ID": "07a21ccbfe7fe453462fee9a86bc806c8950423f"
}
```

Cet identifiant est généré via l'algorithme de hachage cryptographique SHA-1 [6] qui transforme des documents numérisés en clé de 160 bits. La probabilité de collision de deux clés est très faible ($\frac{1}{2^{63}}$), et l'on peut donc considérer que cette clé constitue un identificateur unique. Ce type d'identification est, par exemple, utilisé par Git [2], le système de gestion de versions distribué.

La clé SHA-1 est la représentation interne du code GMN qui est utilisée pour toutes les requêtes ultérieure. Dans l'exemple suivant, on accède à une partition en faisant référence à sa clé SHA-1. Cette clé sera désormais abrégée en `<key>` pour faciliter la lecture.

`http://guido.grame.fr:8000/<key>`

Sans autre information dans l'URL, l'objet de retour par défaut est illustré par la Figure 1.

Figure 1. Score with SHA-1 tag `<key>`.

Pour un fonctionnement totalement RESTful, le code GMN devrait être fournit avec toute requête le concernant. Dans la pratique, on peut considérer que la clé SHA-1 remplace ce code sans effet de bord. Son utilisation minimise la déviation du style RESTful tout en proposant un point d'accès unique à la partition.

3.2. La méthode GET

Une requête GET permet d'obtenir des informations sur une partition identifiée sur le serveur par une clé SHA-1 ou différentes représentations de cette partition. Selon la requête, les données renvoyées par le serveur sont de type :

- image (jpeg, png ou svg) pour une représentation graphique de la partition,
- MIDI pour une représentation MIDI de la partition,
- JSON pour toute autre demande d'information.

4. L'API HTTP DE LA LIBRAIRIE GUIDO

Le serveur GUIDO expose l'API de la librairie GUIDO en créant des équivalences une à une entre l'API HTTP et les fonctions C/C++ correspondantes. Les arguments des fonctions sont passés par le biais de paires *clé-valeur* dans la partie *query* de l'URI transmise au serveur. Une présentation complète de l'API se trouve dans la documentation du serveur [7].

Cette section décrit les conventions utilisées pour passer de l'API C/C++ à l'API HTTP. Elle est suivie de plusieurs exemples concrets d'interactions avec le serveur.

4.1. Clé SHA-1 comme représentation de partition

La section 3.1 a décrit comment la clé SHA-1 remplace du code GMN dans les URIs envoyés au serveur. Cette clé peut-être vue comme l'équivalent d'un pointeur sur une partition, en l'occurrence un *handler* sur une représentation abstraite (GAR) de la partition pour la librairie GUIDO.

4.2. Fonction comme segment d'URI

Un segment d'URI dans l'API HTTP correspond à une fonction dans l'API C/C++. Par exemple, la fonction `GuidoGetPageCount` dans l'API C/C++ correspond au segment `pagescount` dans une URI.

Les fonctions de l'API GUIDO peuvent être classées en deux catégories :

- fonctions qui fournissent de l'information sur la librairie.
- fonctions qui fournissent de l'information sur une partition.

Pour l'API C/C++ de la librairie, les fonctions qui s'adressent à une partition prennent comme argument une *handler* sur la partition, qui peut être vu comme un pointeur sur la représentation interne de la partition. Avec l'API HTTP, la clé SHA-1 remplace ce *handler* dans l'URI et permet de définir la partition cible de la requête :

- les requêtes adressées à une partition sont préfixées par la clé SHA-1,
- les requêtes adressées au moteur de rendu ne sont pas préfixées.

Par exemple,

```
http://guido.grame.fr:8000/version
```

renvoie la version de la librairie.
Par ailleurs, l'URI :

```
http://guido.grame.fr:8000/<key>/voicescount
      ou <key> est la clé SHA-1
```

expose la fonction `GuidoCountVoices` par le biais du segment `voicescount`, adressé à la partition identifiée par `<key>`, et renvoie le nombre de voix de la partition.

La table 1 présente une liste de fonctions de l'API GUIDO et les segments d'URI correspondants dans les requêtes au serveur.

4.3. Arguments comme paires clé-valeur

Pour les fonctions qui prennent des arguments, ceux-ci sont déclinés en paires clé-valeur dans l'URI. Par exemple, la fonction `GuidoGetStaffMap` prend un argument `staff` qui permet de préciser la *staff* (portée) cible de la requête.

C/C++ API	URI segment	cible
GuidoGetPageCount	pagescount	partition
GuidoGetVoiceCount	voicescount	partition
GuidoDuration	duration	partition
GuidoFindPageAt	pageat	partition
GuidoGetPageDate	pagedate	partition
GuidoGetPageMap	pagemap	partition
GuidoGetSystemMap	systemmap	partition
GuidoGetStaffMap	staffmap	partition
GuidoGetVoiceMap	voicemap	partition
GuidoGetTimeMap	timemap	partition
GuidoAR2MIDIFile	midi	partition
GuidoGetVersionStr	version	moteur
GuidoGetLineSpace	linespace	moteur

Table 1. Représentation des fonctions de l'API GUIDO comme segments d'URI.

```
http://guido.grame.fr:8000/<key>/staffmap?staff=1
```

Pour permettre aux URIs incomplets ou mal-formés d'être traités, tous les arguments passés aux fonctions ont des valeurs par défaut pour le serveur. Ces valeurs sont parfois indiquées dans l'API et sinon correspondent à des valeurs qui auraient du sens pour la plupart de partitions.

4.4. Options de mise en page et formatage

Le serveur GUIDO permet le contrôle de la mise en page et le formattage de la partition grâce à des paires clé-valeur indiqués de manière similaire aux arguments des fonctions décris ci-dessus. Ces paramètres sont utilisés de différentes manières selon leur fonction dans l'API C/C++ GUIDO . Certains paramètres, comme `topmargin`, font partie d'une structure `GuidoPageFormat` qui est utilisée pour contrôler la taille globale d'une page. D'autres, comme `resize`, déclenchent un appel à une fonction. Le paramètre `width` est utilisé plusieurs fois dans le processus de compilation selon le format de sortie choisi.

Toutes les options de mise en page et de formatage sont spécifiées dans l'URI, afin de fournir toutes les informations nécessaires au rendu de la partition sans gérer d'états et de satisfaire ainsi aux recommendations RESTful.

4.5. Valeurs de retour

Les paquets renvoyés par le serveur sont constitués d'une enveloppe et d'un contenu. L'enveloppe indique un code de retour et un type MIME pour le contenu associé. Les types renvoyés sont parmi :

- `image/[jpeg | png | svg+xml]` pour une représentation graphique de la partition au format jpeg, png ou svg,
- `audio/midi` pour une représentation MIDI de la partition,

– `application/json` pour des informations structurées sur la partition.

JSON est utilisé de manière consistante pour toutes les demandes informations. Pour les fonctions qui renvoient des structures de données, les données renvoyées par le serveur sont typées et structurées de manière équivalente en JSON. Par exemple, la structure `Time2GraphicMap` contient une liste de paires où chaque paire contient la durée d'un événement (`TimeSegment`) et les coordonnées du rectangle englobant l'événement dans la la page (`FloatRect`). `TimeSegment` et `FloatRect` sont elles mêmes des structures qui contiennent d'autres données. Ces structures peuvent être articulées comme une hiérarchie de dictionnaires JSON, comme dans l'exemple fourni dans la section 4.6.3 où `time` correspond à un `TimeSegment` et `graph` correspond à un `FloatRect`.

4.6. Exemples

4.6.1. voicescount

La requête `voicescount` demande le nombre de voix dans une partition. Elle correspond à la fonction `GuidoCountVoices` de l'API Guido . L'URI :

`http://guido.grame.fr:8000/<key>/voicescount`

produit la réponse suivante du serveur :

```
{
  "<key>": {
    "voicescount": 1
  }
}
```

où <key> est la clé SHA-1.

4.6.2. pageat

La requête `pageat` demande la page contenant la date passée en argument, où « date » correspond à une date de la partition. Elle correspond à la fonction `GuidoFindPageAt` de l'API Guido . L'URI :

`http://guido.grame.fr:8000/<key>/pageat?date=1/4`

produit la réponse suivante du serveur :

```
{
  "<key>": {
    "page": 1,
    "date": "1/4"
  }
}
```

4.6.3. staffmap

La requête `staffmap` demande la description des relations entre espace graphique et temporel de la partition. Elle renvoie une liste de paires associant un espace graphique décrit comme un rectangle, et un espace temporel décrit comme un intervalle borné par deux dates. Les dates sont indiquées par des rationels exprimant du temps musical (i.e. relatif à un tempo) où 1 représente la ronde. La requête correspond à la fonction `GuidoGetStaffMap` de l'API Guido.

L'URI :

`http://guido.grame.fr:8000/<key>/staffmap?staff=1`

produit la réponse suivante du serveur (abrégée pour des raison de commodité) :

```
{
  "<key>": {
    "staffmap": [
      {
        "graph": {
          "left": 916.18,
          "top": 497.803,
          "right": 1323.23,
          "bottom": 838.64
        },
        "time": {
          "start": "0/1",
          "end": "1/4"
        }
      },
      .
      .
      .
      {
        "graph": {
          "left": 2137.33,
          "top": 497.803,
          "right": 2595.51,
          "bottom": 838.64
        },
        "time": {
          "start": "3/4",
          "end": "1/1"
        }
      }
    ]
  }
}
```

5. CONCLUSION

Le serveur Guido repose sur une architecture architecture RESTful afin d'exposer l'API C/C++ de la librairie Guido à travers une API HTTP. La spécification cette API permet de traiter les requêtes sans gérer d'états successifs, dans la mesure où toutes les informations nécessaires à une requête sont contenues dans son URI. La disponibilité de ce service sur Internet ouvre de nouvelles perspectives dans le développement d'applications Web qui souhaitent incorporer la notation musicale symbolique, que ce soit pour visualiser des partitions ou encore pour exploiter des données sur ces partitions.

Le projet Guido est un projet *open source* hébergé sur sourceforge (`http://guidolib.sf.net`). Le serveur est actuellement accessible à l'adresse `http://guidoservice.grame.fr/`.

Références

[1] T. Bonte. MuseScore : Open source music notation and composition software. Technical report, Free and Open source Software Developers' European Meeting, 2009. http://www.slideshare.net/thomasbonte/musescore-at-fosdem-2009.

[2] Scott Chacon. *Pro Git*. Books for professionals by professionals. Apress, 2009.

[3] D. Crockford. The json data interchange format. Technical report, ECMA International, October 2013.

[4] C. Daudin, D. Fober, S. Letz, and Y. Orlarey. The Guido Engine - a toolbox for music scores rendering. In *Proceedings of the Linux Audio Conference 2009*, pages 105–111, 2009.

[5] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[6] P. Gallagher. Secure hash standard (shs). Technical report, National Institute of Standards and Technology, March 2012.

[7] Grame. *Guido Engine Web API Documentation v.0.50*, 2014.

[8] Grame. *GuidoLib v.1.52*, 2014.

[9] H. Hoos, K. Hamel, K. Renz, and J. Kilian. The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music. In *Proceedings of the International Computer Music Conference*, pages 451–454. ICMA, 1998.

[10] H. H. Hoos and K. A. Hamel. The GUIDO Music Notation Format Specification - version 1.0, part 1 : Basic GUIDO. Technical report TI 20/97, Technische Universitat Darmstadt, 1997.

[11] Renz K. and H. Hoos. A Web-based Approach to Music Notation Using GUIDO. In *Proceedings of the International Computer Music Conference*, pages 455–458. ICMA, 1998.

[12] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, 2008.

PROGRAMMER AVEC DES TUILES MUSICALES: LE T-CALCUL EN EUTERPEA

Paul Hudak
Department of Computer Science
Yale University
New Haven, CT 06520-8285
paul.hudak@yale.edu

David Janin
LaBRI, CNRS UMR 5800,
IPB, Université de Bordeaux,
F-33405 Talence
david.janin@labri.fr

Résumé

Euterpea est un langage de programmation dédié à la création et à la manipulation de contenus media temporisés – son, musique, animations, vidéo, etc.... Il est enchassé dans un langage de programmation fonctionnelle avec typage polymorphe : Haskell. Il hérite ainsi de toute la souplesse et la robustesse d'un langage de programmation moderne.

Le T-calcul est une proposition abstraite de modélisation temporelle qui, à travers une seule opération de composition : le produit tuilé, permet tout à la fois la composition séquentielle et la composition parallèle de contenus temporisés.

En présentant ici une intégration du T-calcul dans le language Euterpea, nous réalisons un outil qui devrait permettre d'évaluer la puissance métaphorique du tuilage temporel combinée avec la puissance programmatique du langage Euterpea.

1. INTRODUCTION

Programmer des pièces musicales.

De nos jours, il existe de nombreux langages de programmation dédiés à la composition musicale tels que, parmi d'autres, le langage Euterpea en Haskell [10] qui s'appuie sur la bibliothèque Haskore [13], le langage Elody [25] qui étend le λ-calcul, ou encore CLISP/ CLOS [4] pour OpenMusic [1].

Ces langages permettent de décrire des séquences de notes ou des agencements de flux audio. Ils offrent aussi, de plus en plus, des opérations abstraites de manipulation de séquences musicales entières, en intégrant des concepts de programmation classiques. La musique écrite dans ces langages peut ainsi résulter de l'*exécution* d'algorithmes de génération de flux musicaux.

Dans ces langages, bon nombre de structures de données et de contrôles usuelles des langages de programmation sont ainsi disponibles pour cette écriture algorithmique. Elles permettent de prendre de la hauteur en offrant aux compositeurs des métaphores de modélisation musicale plus adaptées à une pensée musicale abstraite. Paradoxalement, les structures musicales induites par ces langages de programmation peuvent aussi souffrir de limitations.

Les concepts de listes, d'arbres ou même de fonctions, sont bien entendu applicables à une écriture algorithmique de la musique. Les travaux autours de la linguistique appliquée à la musicologie [24] témoigne de leur pertinence. Cependant, de nombreuses constructions musicales ne s'y prêtent pas vraiment comme l'illustre, par exemple, des pièces polyrythmiques à la structure complexe telles que les arabesques de Debussy.

Programmer l'espace et le temps.

Une problématique majeure à laquelle ces langages doivent répondre est de rendre compte des caractéristiques spatiales et temporelles du langage musical.

Dans une approche classique, la *dimension temporelle* est modélisée par le *produit séquentiel* : la concaténation de deux listes, et la *dimension spatiale* est modélisée par le *produit parallèle*. Cette approche est formalisée dans [9]. Elle conduit à définir l'algèbre des flux média polymorphes. Applicable à la musique comme dans le *Domain Specific Language* Euterpea[10] basé sur la librairie Haskore [13] réalisé en Haskell, elle est aussi applicable à tout flux media temporisé tels que les flux vidéo, les animations ou même les flux de contrôle-commande [8].

Une étude récente des structures rythmiques [15] montre cependant que la distinction faite entre composition séquentielle et parallèle n'est pas compatible avec une description hiérarchique, multi-échelle, des structures musicales. Par exemple, un départ en anacrouse peut empiéter sur le flux musical qui le précède. Il induit un parallélisme local. Pourtant, au niveau de l'intention musicale, plus abstraite, il pourra apparaitre lors de la composition séquentielle de deux mélodies.

Bien sur, ces superpositions locales pourraient être traitées comme des exceptions à cette distinction séquentielle/parallèle. Notre proposition consiste, au contraire, à les expliciter. Ce faisant, la composition n'est plus séquentielle ou parallèle : elle est tuilée.

travail partiellement soutenu par ANR-12-CORD-0009

La modélisation par tuilage spatio-temporel.

Dans la continuité des propositions existantes pour la programmation musicale [1, 25, 10], comme dans la continuité des approches en musicologie formelle [24], la *modélisation par tuilage* permet d'unifier les notions de compositions séquentielles et parallèles.

Plus précisément, en nous appuyant sur le concept de *flux media temporisés* [8, 9] enrichi par des *marqueurs de synchronisation* [3], nous obtenons une notion de *flux media temporels tuilés*. Le produit associé, appelé *produit tuilé*, apparaît tout à la fois comme un opérateur séquentiel et un opérateur parallèle.

En pratique, la composition tuilée s'inspire de travaux déjà anciens. Le produit de tuilage ne fait qu'offrir une alternative à la notion de barre de mesure en musique. Il permet par exemple de modéliser la notion musicale d'anacrouse [15]. Le produit tuilé apparaît déjà implicitement dans le langage LOCO [6], une adaptation du langage *Logo* à la manipulation de flux musicaux.

La formalisation du produit tuilé, nous a conduit à proposer une algèbre dédiée à la synchronisation des flux musicaux [3]. Deux outils de manipulations de flux audio tuilés, la librairie *libTuiles* et l'interface *liveTuiles* [22], dérivent de cette approche. In fine, une proposition abstraite : le T-Calcul [21], propose d'intégrer ces concepts à un langage de programmation.

En théorie, le produit de tuilage apparaît aussi dans les monoïdes inversifs [23]. Son usage comme outil de modélisation pour les systèmes informatisés semble prometteur [20]. Le produit tuilé, tel que nous proposons de le manipuler en vue d'applications musicales, est donc une construction particulièrement bien fondée au cœur d'une théorie mathématique dont la robustesse n'est plus à démontrer.

2. TUILER LES FLUX TEMPORISÉS

Nous décrivons ici comment les flux media tuilés peuvent être construit à partir des flux media en les enrichissant simplement de deux marqueurs de synchronisation. Cette construction peut être vue comme une abstraction des notions de synchronisation et de mixage telle qu'elle sont couramment mise en œuvre dans les langages de programmation musicale comme dans les studios de montage audio.

2.1. Flux media temporisés

Un *flux média temporisé* [8] est une structure abstraite représentant des données qui sont positionnées dans le temps relativement au début du flux media. Cette notion est illustrée Figure 1 sur un axe temporel, implicite, allant du passé, à gauche, vers le futur, à droite. Parfois produites par des programmes, ces structures peuvent être de durée infinie.

Figure 1. Un flux média temporisé fini m_1 et un flux média temporisé infini m_2.

Les opérations naturellement associées à ces flux média temporisés sont : la *composition séquentielle*

$$m_1 :+: m_2$$

qui modélise le lancement de deux flux média temporisés l'un après l'autre, le premier étant nécessairement fini, et, la *composition parallèle*

$$m_1 :=: m_2$$

qui modélise le lancement de deux flux média temporisés en même temps. Ces notions sont illustrés Figure 2.

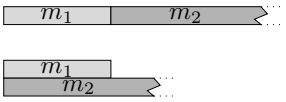


Figure 2. Composition séquentielle $m_1 :+: m_2$ et composition parallèle $m_1 :=: m_2$ des flux m_1 et m_2.

Il apparait que ces deux opérations sont des cas particuliers d'une opération plus générale : le *produit synchronisé* ou *produit tuilé*.

2.2. Flux media temporisés tuilés

Un flux media temporisé tuilé est définit comme un flux media temporisé m enrichi de deux marqueurs de synchronisation *pre* et *post* qui sont définis par la distance, mesurée en temps, qui les sépare du début du flux. Autrement dit, un flux media tuilé t peut simplement être codé comme un triplet

$$t = (pre, post, m)$$

avec *pre* et *post* définit comme deux rationnels positifs et m un flux media. Un tel flux tuilé est illustré Figure 3.

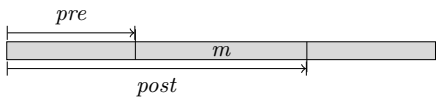


Figure 3. Un flux media temporel.

Remarque. Cette définition des flux tuilés reste valide avec des flux infinis, la position de référence pour positionner *pre* et *post* sur un flux étant toujours le début de ce flux.

2.3. Produit tuilé

Le *produit tuilé* $t_1 \,\%\, t_2$ de deux flux tuilés de la forme $t_1 = (pre_1, post_1, m_1)$ et $t_2 = (pre_2, post_2, m_2)$,

se définit alors en deux étapes successives. La *synchronisation*, qui consiste à positionner les deux flux t_1 et t_2 l'un par rapport à l'autre, de telle sorte que le marqueur de sortie $post_1$ du premier flux tuilé t_1 coïncide avec le marqueur d'entrée pre_2 du second flux tuilé t_2. La *fusion*, qui consiste alors à fusionner [1] les deux flux sous-jacents ainsi positionnés, en conservant pre_1 comme marqueur d'entrée et $post_2$ comme marqueur de sortie résultant de ce produit.

Cette construction est illustrée Figure 4. Dans cette

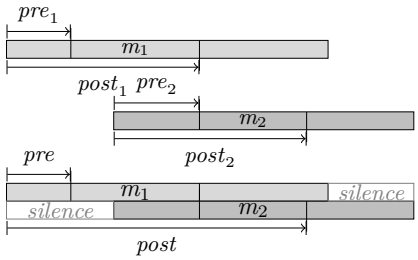


Figure 4. Une instance de produit synchronisé

figure le début du flux musical résultant provient de la première tuile. C'est là un cas particulier comme on peut le vérifier sur un autre example, décrit Figure 5.

3. IMPLÉMENTATION EN HASKELL

L'implémentation en *Haskell/Euterpea* des flux media tuilés et du produit tuilé ne fait que mettre en œuvre les schémas ci-dessus. Les opérateurs qui peuvent dériver de ces structures sont décrits par la suite, à la fois par leur implémentation en *Haskell* et par le schéma correspondant.

3.1. Le type tuile

Le type de donnée *Tile a* est codé de la façon suivante à partir du type *Music a*, qui est une instance particulière du type *Temporal a* :

```
data Tile a = Tile { preT  :: Dur,
                     postT :: Dur,
                     musT  :: Music a }
```

avec **type** *Dur* = *Rational*. On utilise ici le type *Rational* afin d'éviter les erreurs d'approximation qui pourraient résulter, par exemple, de l'utilisation du type *Float*.

Une fonction *durT* permet de calculer la *durée de synchronisation* d'un flux tuilé, c'est à dire, le temps relatif entre la marque de synchronisation *pre* et la marque de synchronisation *post*.

1 . Fusion qui dépendra du media considéré : mixage pour de l'audio, union pour de la musique, superposition pour de la vidéo, etc.

```
durT :: Tile a → Dur
durT (Tile pr po m) = po − pr
```

Dans le cas où le marqueur *pre* se trouve *avant* le marqueur *post* on dit que le flux tuilé est *positif*. Dans le cas contraire, lorsque le marqueur *pre* est situé *après* le marqueur *post*, on dit que le flux tuilé est *négatif*. Dans tous les cas, le flux media est inchangé. Il sera toujours produit du passé (représenté à gauche) vers le futur (représenté à droite).

3.2. Le produit tuilé

Le produit synchronisé, noté % est alors codé par :

```
(%) :: Tile a → Tile a → Tile a
Tile pr1 po1 m1 % Tile pr2 po2 m2 =
  let d = po1 − pr2
  in Tile (max pr1 (pr1 − d))
          (max po2 (po2 + d))
          (if d > 0 then m1 :=: mDelay d m2
           else mDelay (−d) m1 :=: m2)
```

où *mDelay* est la fonction définit par :

```
mDelay d m = case signum d of
   1 → rest d :+: m
   0 → m
 − 1 → m :+: rest (−d)
```

Dans ce codage, la fonction *mDelay* assure la phase de synchronisation, le produit parallèle :=: assure la phase de fusion. Implicitement, du silence est inséré de part et d'autre des flux musicaux sous-jaçent afin de permettre cette composition parallèle.

Selon la position des marqueurs de synchronisation, le début de la musique peut provenir de la première tuile, comme dans le cas de la Figure 4 ou bien de la seconde tuile, comme dans le cas de la Figure 5.

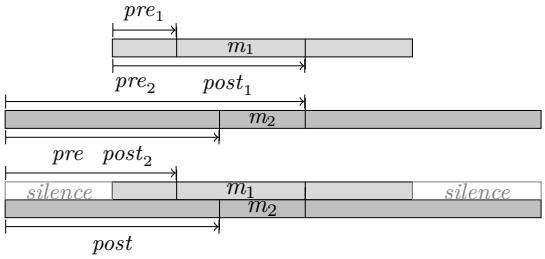


Figure 5. Une autre instance de produit synchronisé

3.3. Reset, co-reset et inverses

Trois fonctions sur les tuiles : le *reset*, le *co-reset* et l'*inverse*, découlent de ces définitions. Elles sont,

comme on le verra, liées les unes aux autres.

```
re, co, inv :: Tile a → Tile a
re  (Tile pre post m) = Tile pre pre m
co  (Tile pre post m) = Tile post post m
inv (Tile pre post m) = Tile post pre m
```

Leur effet sur une tuile t est décrit Figure 6.

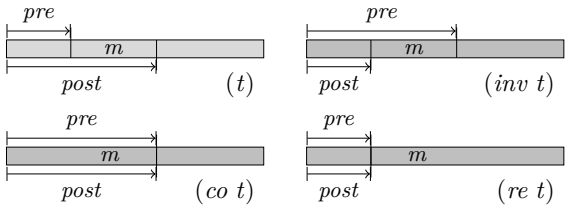


Figure 6. Reset, co-reset and inverse

3.4. Codage des tuiles musicales

Chaque note, de type *Music Pitch* en Euterpea, est décrite comme un triplet (n, o, d) avec un nom [2] n, une octave [3] o et une durée [4] d. Ces notes peuvent être converties en notes tuilées à l'aide de la fonction t définie par :

```
t :: (Octave → Dur → Music Pitch)
          → Octave → Dur → Tile Pitch
t n o d = if d < 0 then Tile d 0 (n o (−d))
          else Tile 0 d (n o d)
```

De même pour les silences, avec la fonction r définie par :

```
r :: Dur → Tile a
r d = if d < 0 then Tile d 0 (rest (−d))
      else Tile 0 d (rest d)
```

Pour jouer une tuile, on ne convient de ne garder que la partie musicale situé entre les marqueurs *pre* et *post*. Cette *projection* est réalisée par la fonction *tToM* décrite ci-dessous.

```
tToM :: Tile a → Music a
tToM (Tile pr po m) = takeM (po − pr)
                            (dropM pr m)
```

où les fonctions Euterpea *takeM* et *dropM* sur les objets de type *Music* permettent de supprimer ou extraire, comme sur les listes, des sous-séquences musicales selon le paramètre de durée passé en argument.

La musique codée dans une tuile peut alors être jouée par composition :

```
playT = play ∘ tToM
```

où la fonction *play* est la fonction en Euterpea permettant de jouer des séquences musicales de type *Music*.

3.5. Premier exemple

On peut ainsi proposer un premier exemple :

```
fj1 = t c 4 en % t d 4 en % t e 4 en % t c 4 en
fj2 = t e 4 en % t f 4 en % t g 4 qn
fj3 = t g 4 sn % t a 4 sn % t g 4 sn % t f 4 sn
    % t e 4 en % t c 4 en
fj4 = t c 4 en % t g 3 en % t c 4 qn
```

qui code le canon *Frère Jacques*. Il pourra être joué par la commande
```
test1 = playT (fjr % r 4)
```

Une fonction *tempoT* permet aussi de changer le tempo des tuiles. Elle est codée de la façon suivante, héritant en quelque sorte de la fonction *tempo* sur les objets musicaux sous-jacents :

```
tempoT :: Dur → Tile a → Tile a
tempoT r (Tile pr po m) =
  assert (r > 0) (Tile (pr/r) (po/r) (tempo r m))
```

On utilise ici la fonction *Control.Exception.assert* qui permet de vérifier que le coefficient r de changement de tempo est strictement positif.

Plus généralement, toute fonction agissant sur les objets musicaux peut être appliquée aux tuiles, lorsqu'on ne souhaite pas modifier les paramètres de synchronisation, grâce à la fonction d'ordre supérieur *liftT* définie par :

```
liftT :: (Music a → Music b) → (Tile a → Tile b)
liftT f (Tile pr po m) = Tile pr po (f m)
```

Ainsi, l'exemple suivant nous permettra de jouer *Frère Jacques* sur un piano Rhodes deux fois plus vite.
```
fjR = litfT (intrument rhodesPiano) fjr
test2 = playT (tempoT 2 (fjR % r 4))
```

4. ÉQUIVALENCE OBSERVATIONNELLE

Un concept important en Euterpea est l'équivalence observationnelle. Elle se généralise sur les flux musicaux tuilés nous permettant de donner un sens à l'affirmation « les flux tuilés t_1 et t_2 se comportent de la même façon ».

2 . En notation anglaise, de *a* pour *la* à *g* pour *sol*, avec *c* pour *do*, *cf* (*c flat*) pour *do* bémol, *cs* (*c flat*) pour *do* dièse, etc.

3 . C'est-à-dire un numéro d'octave, le do à la clé correspondant à *c* 4 ; il est précédé de *b* 3 et suivie de *d* 4, un clavier de piano allant typiquement de *a* 0, le *la* le plus grave, à *c* 8, le *do* le plus aigu.

4 . En valeur symbolique valant fraction de ronde, avec des constantes prédéfinies, en notation anglaise, telle que *wn* (*whole note*) pour la ronde, valant 1, ou bien *qn* (*quarter note*) pour la noire, valant 1/4, ou encore *en* (*eight note*) pour la croche, calant 1/8, etc.

4.1. Equivalence de flux

Deux flux musicaux m_1 et m_2 sont observationellement équivalent, ce qu'on note

$$m_1 \text{ `}equiv\text{` } m_2$$

lorsqu'ils produisent le même effet à l'execution, c'est à dire, lorsque *play* m_1 et *play* m_2 produisent le même effet, où *play* désigne la fonction *jouant* les flux musicaux.

Les détail sur cette équivalence *equiv* peuvent être trouvé dans [8, 9]. Pour ce qui nous interesse, il suffit de savoir qu'elle satisfait les axiomes suivants :

$$\begin{array}{rcl}
(m_1 :+: m_2) :+: m_3 & \text{`}equiv\text{`} & m_1 :+: (m_2 :+: m_3) \\
(m_1 :=: m_2) :=: m_3 & \text{`}equiv\text{`} & m_1 :=: (m_2 :=: m_3) \\
m_1 :=: m_2 & \text{`}equiv\text{`} & m_2 :=: m_1 \\
rest\ 0 :+: m & \text{`}equiv\text{`} & m \\
m :+: rest\ 0 & \text{`}equiv\text{`} & m
\end{array}$$

et, lorsque $dur\ m_1 = dur\ m_3$

$$\begin{array}{l}
(m_1 :+: m_2) :=: (m_3 :+: m_4) \\
\qquad \text{`}equiv\text{`}\ (m_1 :=: m_3) :+: (m_2 :=: m_4),
\end{array}$$

On suppose en outre que l'équation suivante

$$m \text{ `}equiv\text{` } (m :=: m)$$

est satisfaite pour tout flux tuilé m.

Remarque. Bien que raisonnable, cette dernière équivalence necessite une réelle mise en oeuvre. En effet, dans de nombreux logiciels, jouer en parallèle deux fois le même flux musical s'accompagne, en général, d'une augmentation du volume. A défaut d'une telle implementation, cette équivalence devra donc être comprise *modulo* la balance des volumes des pistes.

4.2. Equivalence de flux tuilés

Cette *équivalence observationnelle* se généralise aux flux tuilés en disant que deux flux tuilés t_1 et t_2 sont observationnellement équivalents lorsqu'ils produisent le même effet à l'exécution quel que soit le contexte d'exécution dans lequel on les joue. Cette mise en contexte nous permet de rendre compte des paramètres de synchronisation.

Formellement, en notant $playT$ la fonction qui permet de *jouer* le flux media d'une tuile *entre* les points de synchronisation *pre* et *post*, deux flux tuilés t_1 et t_2 seront équivalents, ce qui sera noté $t_1 \equiv t_2$ lorsque

$$\begin{array}{l}
Tile\ pr_1\ po_1\ m_1 \equiv Tile\ pr_2\ po_2\ m_2 = \\
\quad pr_1 - po_1 == pr_2 - po_2 \wedge \\
\quad \textbf{let}\ d\ = dur\ m_1 - dur\ m_2 \\
\qquad p\ = pr_1 - pr_2 \\
\qquad n_1 = \textbf{if}\ d < 0 \\
\qquad\quad \textbf{then}\ mDelay\ d\ m_1\ \textbf{else}\ m_1 \\
\qquad n_2 = \textbf{if}\ d > 0 \\
\qquad\quad \textbf{then}\ mDelay\ (-d)\ m_2\ \textbf{else}\ m_2 \\
\quad \textbf{in}\ \ \textbf{if}\ p > 0\ \textbf{then}\ n_1\ \text{`}equiv\text{`}\ mDelay\ p\ n_2 \\
\qquad \textbf{else}\ mDelay\ (-p)\ n_1\ \text{`}equiv\text{`}\ n_2
\end{array}$$

On vérifie que deux flux tuilés t_1 et t_2 sont équivalents si et seulement si, pour toute tuiles de silence r_1 et r_2 on a bien

$$tToM\ (r_1 \,\%\, t_1 \,\%\, r_2)\ \texttt{'equiv'}\ tToM\ (r_1 \,\%\, t_2 \,\%\, r_2)$$

4.3. Structure algébrique induite

On vérifie aussi que pour tous flux tuilés t, t_1, t_2 et t_3, les équivalences suivantes sont satisfaites.

$$\begin{array}{rcl}
t_1 \,\%\, (t_2 \,\%\, t_3) & \equiv & (t_1 \,\%\, t_2) \,\%\, t_3 \\
t \,\%\, r\ 0 & \equiv & t \\
r\ 0 \,\%\, t & \equiv & t
\end{array}$$

La première équivalence indique que l'ordre dans lequel on évalue les composants d'un produit tuilé n'a pas d'importance. C'est l'équivalence d'associativité. Les deux suivantes indiquent que la tuile silence $(r\ 0)$ de durée de synchronisation nulle agit comme un élement neutre. Autrement dit, l'équivalence observationnelle induit une structure de *monoïde* sur les tuiles.

Plus encore, on constate aussi que pour tout flux tuilé t on a :

$$\begin{array}{rcl}
t & \equiv & inv(invt) \\
t & \equiv & (t \,\%\, (inv\ t) \,\%\, t) \\
(re\ t) & \equiv & (t \,\%\, (inv\ t)) \\
(co\ t) & \equiv & ((inv\ t) \,\%\, t)
\end{array}$$

La première équivalence indique que l'opération d'inversion est une involution. La seconde, que l'inverse d'un flux est bien un inverse au sens de la théorie des semigroupes [23]. Les quatrième et cinquième équivalences montrent le reset et le co-reset sont en fait des opérations dérivées du produit tuilé et de l'inversion.

On peut poursuivre plus en détail l'étude de cette équivalence est découvrir, comme en théorie des semigroupes inversifs, que les flux tuilés de durée de synchronisation nulle sont les seuls qui satisfont n'importe laquelle des équivalences suivantes :

$$\begin{array}{rcl}
t & \equiv & t \,\%\, t \\
t & \equiv & inv\ t
\end{array}$$

et que, de plus, ils commutent, c'est-à-dire que
$| \ t_1 \,\%\, t_2 \equiv t_2 \,\%\, t_1 \ |$
lorsque t_1 et t_2 sont deux tuiles de durée de synchronisation nulle. La théorie des monoides inversifs [23] nous assurent alors que, modulo équivalence observationnelle, tout flux tuilé t admet un unique inverse $(inv\ t)$. La structure induite est un *monoïde inversif.*

4.4. Codage inverse

Dans ce langage de manipulation de tuiles, nous retrouvons aussi les produits séquentiels et parallèles de Euterpea.

En effet, dans le cas flux musicaux finis, on définit la fonction $mToT$ qui transforme tout flux musical fini m en un flux tuilé par :

$$\begin{aligned} &mToT :: Music\ a \to Tile\ a \\ &mToT\ m = \textbf{let}\ d = dur\ (m) \\ &\qquad\qquad \textbf{in}\ Tile\ 0\ d\ m \end{aligned}$$

On constate que ce codage est injectif vis à vis de l'équivalence observationnelle. On constate de surcroit que ce codage est fonctoriel vis à vis de la composition séquentielle. En effet, pour tout flux musical fini m_1 et m_2 on a bien :
$mToT\ (m_1 :+: m_2) \equiv (mToT\ m_1)\ \%\ (mToT\ m_2)$
Autrement dit, le produit de synchronisation code, via la fonction $mToT$, le produit séquentiel.

Dans le cas de flux (potentiellement) infinis, l'encodage décrit ci-dessous échoue. En effet, le produit séquentiel de deux flux infinis ne peut être définit de façon satisfaisante puisque le second flux se voit en quelque sorte repoussé à l'infini. Il ne pourra être joué.

On retrouve ici une problématique abordée et résolue par la théorie des ω-semigroupes [27] en distinguant les structures finies (les *strings*) et les structures infinies (les *streams*). Pourtant, il apparait que la manipulation conjointe de ces deux types d'objets peut être faites dès lors qu'ils sont tuilés (voir[7] pour plus de détails).

En pratique, on définit la fonction $sToT$ qui transforme tout flux musical (potentiellement) infini m en un flux tuilé par :

$$\begin{aligned} &sToT :: Music\ a \to Tile\ a \\ &sToT\ m = Tile\ 0\ 0\ m \end{aligned}$$

On vérifie que ce codage est injectif sur les flux musicaux infinis. De plus, il est fonctoriel vis à vis de la composition parallèle. En effet, pour tout flux musical (potentiellement) infini m_1 et m_2 on a bien :
$sToT\ (m_1 :=: m_2) \equiv (sToT\ m_1)\ \%\ (sToT\ m_2)$
Autrement dit, le produit tuilé encode, via la fonction $sToT$, le produit parallèle.

Bien entendu, ces deux exemples peuvent sembler artificiels puisque le produit tuilé, lui-même, est définit à partir des produits séquentiels et parallèles. Mais cette objection ne tient pas puisque qu'elle s'appuie sur une implémentation particulière. Tout autre implémentation du produit tuilé satisfera les propriétés énoncées ci-dessus.

5. RESYNCHRONISATION

Un premier jeu de fonctions définit sur les flux tuilés permet de manipuler la position des points de synchronisation positionnés sur un flux musical tuilé.

5.1. Fonctions de resynchronisation

La fonction *resync* permet de déplacer le point de sortie *post* de la synchronisation. De façon duale, la fonction *coresync* permet de déplacer le point d'entrée *pre* de la synchronisation.

$$\begin{aligned} &resync :: Dur \to Tile\ a \to Tile\ a \\ &resync\ s\ (Tile\ pre\ post\ m) = \\ &\quad \textbf{let}\ npost = post + s \\ &\quad \textbf{in if}\ npost < 0 \\ &\qquad \textbf{then}\ Tile\ (pre - npost)\ 0\ (mDelay\ (-npost)\ m) \\ &\qquad \textbf{else}\ Tile\ \ pre\ npost\ m \\ &coresync :: Dur \to Tile\ a \to Tile\ a \\ &coresync\ s\ (Tile\ pre\ post\ m) = \\ &\quad \textbf{let}\ npre = pre + s \\ &\quad \textbf{in if}\ npre < 0 \\ &\qquad \textbf{then}\ Tile\ 0\ (post - npre)\ (mDelay\ (-npre)\ m) \\ &\qquad \textbf{else}\ Tile\ npre\ post\ m \end{aligned}$$

Le comportement de ces fonctions est illustré Figure 7 pour un offset $s > 0$.

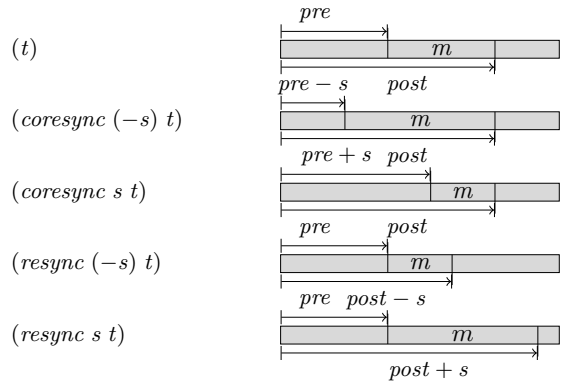


Figure 7. Resynchronisation et co-resynchronisation

5.2. Propriétés de la resynchronisation

Ces fonctions généralisent les fonctions reset *re* et co-reset *co* au sens où on a :

$$\begin{aligned} re\ (Tile\ pr\ po\ m) &= resync\ (pr - po) \\ &\quad\ (Tile\ pr\ po\ m) \\ co\ (Tile\ pr\ po\ m) &= coresync\ (po - pr) \\ &\quad\ (Tile\ pr\ po\ m) \end{aligned}$$

On remarque par ailleurs que ces fonctions induisent des actions du groupe des nombres rationnels muni de l'addition sur l'ensemble des flux tuilés. En pratique, pour tout flux musical tuilé t et tout offset rationnel

a et b on a en effet :

$$\begin{aligned} resync\ 0\ t &= t \\ resync\ a\ (resync\ b\ t) &= resync\ (a+b)\ t \\ coresync\ 0\ t &= t \\ coresync\ a\ (coresync\ b\ t) &= coresync\ (a+b)\ t \end{aligned}$$

En particulier, pour toute tuile t et toute durée s, on constate qu'on a :

$$\begin{aligned} resync\ s\ t &\equiv t \,\%\, r\ s \\ coresync\ s\ t &\equiv r\ s \,\%\, t \end{aligned}$$

Elles sont aussi duales l'une de l'autre au sens des équivalences suivantes :

$$\begin{aligned} resync\ a\ (inv\ t) &\equiv (inv\ (coresync\ a\ t)) \\ coresync\ a\ (inv\ t) &\equiv (inv\ (resync\ a\ t)) \end{aligned}$$

Enfin, leur comportement vis à vis du produit tuilé est décrit par les équivalences observationnelles suivantes. Pour tout flux musical tuilé t_1 et t_2 et tout offset de durée a, on a aussi :

$$\begin{aligned} resync\ a\ (t_1 \,\%\, t_2) &\equiv (resync\ a\ t_1) \,\%\, t_2 \\ coresync\ a\ (t_1 \,\%\, t_2) &\equiv t_1 \,\%\, (coresync\ t_2) \end{aligned}$$

5.3. Fonctions dérivées

Des exemples de fonctions dérivées des fonctions de resynchronisation sont les fonctions insertions de tuiles. Elles sont de deux sortes.

La premiè, fait un *fork* parallèle d'une tuile t_2 dans une tuile t_1 à la position d depuis l'entrée de synchronisation *pre*.

$$\begin{aligned} &insertT :: Dur \to Tile\ a \to Tile\ a \to Tile\ a \\ &insertT\ d\ t_1\ t_2 = coresync\ (-d)\ (re\ t_2 \,\%\, coresync\ d\ t_1) \end{aligned}$$

De façon duale, la seconde, $co-insertion$ fait un *join*.

$$\begin{aligned} &coinsertT :: Dur \to Tile\ a \to Tile\ a \to Tile\ a \\ &coinsertT\ d\ t_1\ t_2 = resync\ (-d)\ (resync\ d\ t_1 \,\%\, co\ t_2) \end{aligned}$$

Le comportement de ces fonctions est décrit Figure 8.

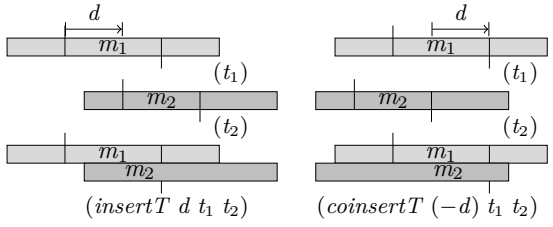


Figure 8. Insertions par Fork et Join.

Remarque. Bien qu'on ait proposé un codage directe de cette fonction, remarquons qu'elle peut aussi se coder simplement à l'aide du produit tuilé, des reset et co-reset, et des tuiles silencieuses. En effet, pour tout flux tuilés t_1 et t_2, et pour tout offset d, on a :

$$\begin{aligned} insertT\ d\ t_1\ t_2 &\equiv r\ d \,\%\, re\ t_2 \,\%\, r\ (-d) \,\%\, t_1 \\ coinsertT\ d\ t_1\ t_2 &\equiv t_1 \,\%\, r\ d \,\%\, co\ t_2 \,\%\, r\ (-d) \end{aligned}$$

5.4. Exemple : croisements temporels

Pour illustrer la puissance de la métaphore du tuilage au sein d'un langage de programmation complet tel que Haskell, on propose ci-dessous le codage d'une fonction *crossing* qui simule, par répétitions et décalages successifs, le *croisement* de deux flux musicaux tuilés.

```
crossing :: Dur → Tile a → Tile a → Tile a
crossing o t1 t2 =
  if ((centerT (t1) > 0)
     ∧ (centerT (t2) > 0)) then
    let v1 = (resync o t1)
        v2 = (resync (−o) t2)
    in (t1 % t2 % (crossing o v1 v2))
  else (t1 % t2)
```

Cette fonction est alors mis en oeuvre dans l'exemple musicale suivant.

```
train1 = liftT (instrument RhodesPiano)
         (r en % t a 3 en % t c 4 en % t e 4 en
              % t d 4 en % r (3 ∗ en))
train2 = liftT (instrument RhodesPiano)
         (t e 4 en % t g 3 en % t a 3 en
                % t a 3 en % r (4 ∗ en))
crossL = crossing (−en) train1 train2
```

L'audition de l'ensemble se fait en executant la commande *playT crossL*. Une ligne de percussion régulière *percL* telle que décrite ci-dessous lancée en parallèle par la commande $playT\ ((re\ percL) \,\%\, crossL)$, permettra d'entendre les phénomènes de décalage de la ligne *crossL*.

6. CONTRACTIONS ET EXPANSIONS

Lors d'une resynchronisation, l'invariant et le flux musicale sous-jacent au flux tuilés. Un jeu de fonctions sensiblement différent est obtenu en préservant la taille de la synchronisation tout en contractant ou étirant le flux musicale sous-jacent. Ce jeu de fonctions est présenté ici.

6.1. Les fonctions de contraction/expansion

La fonction *strech* permet d'étirer et de contracter le flux musical en maintenant la position du point d'entrée de synchronisation *pre* sur le flux musical. de façon duale, la fonction *costretch* étire ou contracte

le flux musical en maintenant le point de sortie de synchronisation *post* sur le flux musical.

La fonction *stretch* est définit par :

$$
\begin{aligned}
&stretch :: Dur \to Tile\ a \to Tile\ a\\
&stretch\ r\ (Tile\ pre\ post\ m) =\\
&\quad assert\ (r > 0)\ (Tile\ \ (pre * r)\ (pre * (r-1) + post)\\
&\qquad\qquad (tempo\ (1/r)\ m))
\end{aligned}
$$

La fonction *costretch* est, quant à elle, définit par :

$$
\begin{aligned}
&costretch :: Dur \to Tile\ a \to Tile\ a\\
&costretch\ r\ (Tile\ pre\ post\ m) =\\
&\quad assert\ (r > 0)\ (Tile\ (post * (r-1) + pre)\ (post * r)\\
&\qquad\qquad (tempo\ (1/r)\ m))
\end{aligned}
$$

Le comportement de ces fonctions est décrit Figure 9 avec un facteur $r > 1$ et en notant (abusivement) $m{*}r$ le flux *tempo* $(1/r)$ m, et m/r le flux *tempo* r m.

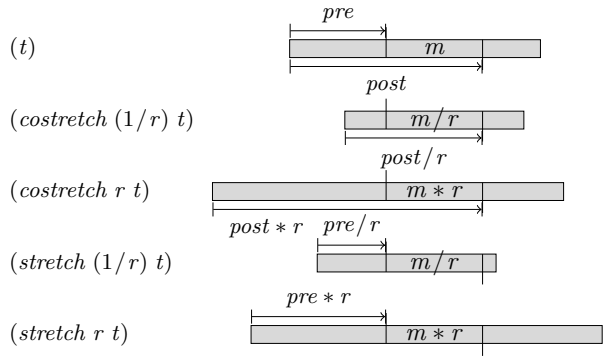


Figure 9. Stretch et co-stretch

6.2. Propriétés des contractions/expansions

De façon analogue aux fonctions de resynchronisation, ces fonctions induisent une action du groupe des nombres rationnels strictement positifs muni de la multiplication sur l'ensemble des flux tuilés. Pour tout flux musical tuilé t et tout offsets a et b, on a :

$$
\begin{aligned}
stretch\ 1\ t &= t\\
stretch\ a\ (stretch\ b\ t) &= stretch\ (a * b)\ t\\
costretch\ 1\ t &= t\\
costretch\ a\ (costretch\ b\ t) &= costretch\ (a * b)\ t
\end{aligned}
$$

Ces deux fonctions sont duales l'une de l'autre au sens des équivalences suivantes :

$$
\begin{aligned}
stretch\ a\ (inv\ t) &\equiv (inv\ (costretch\ a\ t))\\
costretch\ a\ (inv\ t) &\equiv (inv\ (stretch\ a\ t))
\end{aligned}
$$

Enfin, le comportement de ces fonctions vis à vis du produit synchronisé est décrit par les equations suivantes. Pour tout flux musical tuilé t_1 et t_2 et tout facteur $a > 0$, on a :

$$
\begin{aligned}
&stretch\ a\ (t_1\ \%\ t_2) \equiv\\
&\quad (tempoT\ (1/a)\ t_1)\ \%\ (stretch\ a\ t_2)
\end{aligned}
$$

$$
\begin{aligned}
&costretch\ a\ (t_1\ \%\ t_2) \equiv\\
&\quad (costretch\ a\ t_1)\ \%\ (tempoT\ (1/a)\ t_2)
\end{aligned}
$$

6.3. Exemples : génération rythmique

L'utilisation de *stretch* et *costretch* peut être illustré par les transformations rythmiques déjà évoquées dans [15].

$$
\begin{aligned}
&march = t\ c\ 4\ qn\ \%\ r\ qn\ \%\ t\ g\ 4\ qn\ \%\ r\ qn\\
&waltz\ \ = costretch\ (2/3)\ march\\
&tumb = costretch\ (5/4)\ march
\end{aligned}
$$

Le rythme initial *march*, dans une métrique à deux temps, consiste à jouer un *do* sur le premier temps et un *sol* sur le second temps.

Dans le rythme *waltz*, dans une métrique à trois temps, les notes se retrouvent jouées sur le deuxième et le troisième temps de la durée de synchronisation. C'est donc effectivement une ligne de basse pour une valse.

Dans le rythme *tumb*, dans une métrique à quatre temps, le *do* est joué sur le 4ème temps de la mesure qui précède, et le *sol* sur la levée du 2ème temps. C'est là une ligne de basse typique de la salsa cubaine : le *tumbao*. Ces trois exemples sont décrits Figure 10.

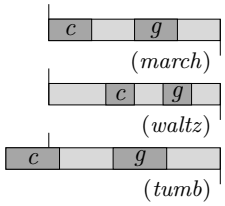


Figure 10. Cellules rythmiques engendrées par contraction/expansion

Joués en parallèle à une structure rythmique répétitive marquant le début de chaque mesure, c'est à dire le point *pre*, on peut écouter ces cellules à l'aide du code suivant :

$$
\begin{aligned}
bass &= liftT\ (instrument\ Percussion)\\
&\quad (Tile\ 0\ wn\ (perc\ AcousticBassDrum\ wn))\\
hiHat &= liftT\ (instrument\ Percussion)\\
&\quad (Tile\ 0\ (1/8)\ (perc\ ClosedHiHat\ (1/8)))\\
bassL &= bass\ \%\backslash\ re\ bassL\\
hiHatL &= repeatT\ 4\ hiHat\ \%\backslash\ re\ hiHatL\\
percL &= re\ bassL\ \%\ hiHatL\\
testW &= playT\ (re\ bassL\ \%\ tempoT\ (3/4)\\
&\quad\qquad (re\ hiHatL)\ \%\ repeatT\ 4\ waltz)\\
testS &= playt\ (re\ bassL\ \%\ re\ hiHatL\\
&\quad\qquad \%\ repeatT\ 4\ tumb)
\end{aligned}
$$

Remarque. Ces exemples sont construits à l'aide d'un opérateur de produit $\%\backslash$ qui, en limitant les superpositions vers le passé, permet de définir facilement

des tuiles récursives. On trouvera ci-dessous, une première discussion sur les définitions récursives tuilés. Une discussion plus poussé sur cette problèmatique délicate pourra être trouvé dans [12].

7. TUILES RÉCURSIVES

Dans les exemples ci-dessus, nous avons vu comment les structures de contrôle classique des langages de programmation permettent de définir *algorithmiquement* des flux tuilés finis complexes.

Le langage Haskell, reposant sur un principe d'évaluation paresseuse, permet aussi de définir des objets potentiellement infinis qui sont évalués à la demande. Nous souhaiterions pouvoir utiliser cette caractéristique pour définir des tuiles potentiellement infinies.

Par exemple, étant donné une tuile finie t, on souhaiterait pouvoir définir une tuile x de support musical infini, via une equation de la forme

$$x = t \,\%\, (re\ x)$$

Dans une telle equation, l'appel récursif se fait sur le *reset* de la variable x afin de produire une tuile dont la distance de synchronisation serait celle de la tuile t. Pourtant, malgré cette précaution, l'évaluation de cette équation en Haskell boucle !

En effet, le modèle des flux tuilés est ainsi fait que les superpositions en amont du point de synchronisation *pre* peuvent provenir de tuiles situées arbitrairement en aval dans une suite de produit synchronisé. Notre implémentation boucle donc sur une telle équation car elle doit dérouler toute les répétitions de x pour en connaitre les anticipations : le typage de la tuile x ainsi définit par équation échoue à être calculé par Haskell.

Dans [21], des conditions nécessaires et suffisantes, calculables, sont décrites pour résoudre cette récursion. Néanmoins, elles sont proposées dans une définition de T-calcul pure, particulièrement limitée. Il ne contient pas de structure de contrôle telle que les conditionnelles. Dans l'implémentation du T-calcul proposée ici, qui hérite de toute l'expressivité d'Haskell, le calcul du type d'un flux tuilé définit par équation est, en toute généralité, indécidable.

Pour remédier à cela, nous proposons un nouveau produit synchrone, partiel, qui résout ce calcul de type potentiellement cyclique en « coupant » les anticipations qui y conduisent. Plus précisément, on définit le produit restreint $\%\backslash$ suivant :

$$(\%\backslash) :: Tile\ a \to Tile\ a \to Tile\ a$$
$$Tile\ pr_1\ po_1\ m_1\ \ \%\backslash\ \sim(Tile\ pr_2\ po_2\ m_2) =$$
$$\quad Tile\ pr_1\ po_1\ (m_1 :=: mDelay\ po_1\ (dropM\ pr_2\ m_2))$$

Remarque. Le $\sim$ dans cette définition est une technicité qui permet de gouverner l'évaluation paresseuse.

On vérifie facilement qu'une équation de la forme

$$x = t \,\%\backslash\, (re\ x)$$

peut maintenant être typée par Haskell. Le type de la tuile x est bien le type de la tuile t. En effet, l'anticipation induite par $(re\ x)$ a été supprimée dans le produit restreint et sa durée de synchronisation est nulle grace au reset. Les marqueurs d'entrée *pre* et de sortie *post* de toute solution sont donc uniquement déterminés par ceux de la tuile t. Une généralisation de ce codage, plus souple, est proposée dans [12].

8. CONCLUSION

Le T-calcul en Euterpea apparait clairement, via les nombreuses propriétés algébriques qu'il satisfait, comme un formalisme particulièrement robuste pour une description hiérarchique de la structure temporelle des flux media temporisés. Son expérimentation pour la modélisation musicale est en cours.

Bien entendu, l'implémentation proposée ici ne traite que des flux musicaux symboliques. Une intégration des flux audio tuilés reste à mettre en oeuvre. Elle pourrait s'appuyer sur la *libTuiles* déjà réalisée [2, 22].

Une telle extension offrirait sans doute un gain de productivité important pour la création et la production de musique électroacoustique. Une bonne partie du mixage, parfois fastidieux et répétitif, peut en effet être factorisée grace à la métaphore du tuilage : chaque tuile audio intègre une fois pour toute, dans ses points de synchronisation, toute l'information necessaire pour positionner, *avant* ou bien *après*, les autres tuiles audio.

Remarquons aussi que le langage proposé ici n'est destiné qu'à une manipulation « hors temps » des structures musicales. Bien sur, en toute généralité, on ne peut décider de la terminaison de ces programmes. Remarquons cependant que, dès lors qu'un flux tuilés est typé, il est de durée de synchronisation fini. La fonction $playT$ de lecture de ce flux terminera donc puisqu'elle ne parcourt les flux qu'entre les marqueurs de synchronisation *Pre* et *Post*.

On peut mentioner enfin qu'une théorie des langages adaptée à la description et à la manipulation de sous-ensembles de monoïdes inversifs est actuellement en cours de développement. Elle offre des outils traditionnels de manipulation des langages, qu'ils soient logiques [17], algébriques [14, 16] ou qu'ils s'appuient sur la théorie des automates [19, 18, 16]. Autrement dit, en s'appuyant sur cette théorie, il sera possible de développer les outils de spécification, d'analyse et de validations qui pourront accompagner efficacement les outils de modélisations par tuilage.

9. REFERENCES

[1] C. Agon, J. Bresson, and G.Assayag. *The OM composer's Book, Vol.1 & Vol.2.* Collection Musique/Sciences. Ircam/Delatour, 2006.

[2] F. Berthaut, D. Janin, and M. DeSainteCatherine. libTuile : un moteur d'exécution multi-échelle de pro-

cessus musicaux hiérarchisés. In *Actes des Journées d'informatique Musicale (JIM)*, 2013.

[3] F. Berthaut, D. Janin, and B. Martin. Advanced synchronization of audio or symbolic musical patterns : an algebraic approach. *International Journal of Semantic Computing*, 6(4) :409–427, 2012.

[4] J. Bresson, C. Agon, and G. Assayag. Visual Lisp / CLOS programming in OpenMusic. *Higher-Order and Symbolic Computation*, 22(1), 2009.

[5] P. Desain and H. Honing. LOCO : a composition microworld in Logo. *Computer Music Journal*, 12(3) :30–42, 1988.

[6] A. Dicky and D. Janin. Embedding finite and infinite words into overlapping tiles. Research report RR-1475-13, LaBRI, Université de Bordeaux, 2013.

[7] P. Hudak. An algebraic theory of polymorphic temporal media. In *Proceedings of PADL'04 : 6th International Workshop on Practical Aspects of Declarative Languages*, pages 1–15. Springer Verlag LNCS 3057, June 2004.

[8] P. Hudak. A sound and complete axiomatization of polymorphic temporal media. Technical Report RR-1259, Department of Computer Science, Yale University, 2008.

[9] P. Hudak. *The Haskell School of Music : From signals to Symphonies*. Yale University, Department of Computer Science, 2013.

[10] P. Hudak and D. Janin. Tiled polymorphic temporal media. Research report RR-1478-14, LaBRI, Université de Bordeaux, 2014.

[11] P. Hudak, T. Makucevich, S. Gadde, and B. Whong. Haskore music notation – an algebra of music. *Journal of Functional Programming*, 6(3) :465–483, May 1996.

[12] D. Janin. Quasi-recognizable vs MSO definable languages of one-dimensional overlapping tiles. In *Mathematical Found. of Comp. Science (MFCS)*, volume 7464 of *LNCS*, pages 516–528, 2012.

[13] D. Janin. Vers une modélisation combinatoire des structures rythmiques simples de la musique. *Revue Francophone d'Informatique Musicale (RFIM)*, 2, 2012.

[14] D. Janin. Algebras, automata and logic for languages of labeled birooted trees. In *Int. Col. on Aut., Lang. and Programming (ICALP)*, volume 7966 of *LNCS*, pages 318–329. Springer, 2013.

[15] D. Janin. On languages of one-dimensional overlapping tiles. In *Int. Conf. on Current Trends in Theo. and Prac. of Comp. Science (SOFSEM)*, volume 7741 of *LNCS*, pages 244–256. Springer, 2013.

[16] D. Janin. Overlaping tile automata. In *8th International Computer Science Symposium in Russia (CSR)*, volume 7913 of *LNCS*, pages 431–443. Springer, 2013.

[17] D. Janin. Walking automata in the free inverse monoid. Research report RR-1464-12, LaBRI, Université de Bordeaux, 2013.

[18] D. Janin. Towards a higher dimensional string theory for the modeling of computerized systems. In *Int. Conf. on Current Trends in Theo. and Prac. of Comp. Science (SOFSEM)*, volume 8327 of *LNCS*, pages 7–20. Springer, 2014.

[19] D. Janin, F. Berthaut, M. DeSainte-Catherine, Y. Orlarey, and S. Salvati. The T-calculus : towards a structured programming of (musical) time and space. In *ACM Workshop on Functional Art, Music, Modeling and Design (FARM)*, pages 23–34. ACM Press, 2013.

[20] D. Janin, F. Berthaut, and M. DeSainteCatherine. Multi-scale design of interactive music systems : the libTuiles experiment. In *Sound and Music Computing (SMC)*, 2013.

[21] M. V. Lawson. *Inverse Semigroups : The theory of partial symmetries*. World Scientific, 1998.

[22] F. Lerdahl and R. Jackendoff. *A generative theory of tonal music*. MIT Press series on cognitive theory and mental representation. MIT Press, 1983.

[23] S. Letz, Y. Orlarey, and D. Fober. Real-time composition in Elody. In *Proceedings of the International Computer Music Conference*, pages 336–339. ICMA, 2000.

[24] D. Perrin and J.-E. Pin. *Infinite Words : Automata, Semigroups, Logic and Games*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

FAUSTLIVE
UN COMPILATEUR À LA VOLÉE POUR FAUST
... ET BIEN PLUS ENCORE

Sarah Denoux
GRAME
sdenoux@grame.fr

Stéphane Letz
GRAME
letz@grame.fr

Yann Orlarey
GRAME
orlarey@grame.fr

RÉSUMÉ

FaustLive est une application qui, grâce à son compilateur Faust embarqué, se propose de réunir le confort d'un langage interprété avec l'efficacité d'un langage compilé. Basée sur *libfaust*, une librairie qui offre une chaîne de compilation complète en mémoire, FaustLive ne requiert aucun outil externe (compilateur, éditeur de lien, ...) pour traduire du code FAUST en code machine exécutable. Par l'intermédiaire de cette technologie, FaustLive offre de multiples fonctionnalités. Par exemple, il est possible de glisser un nouveau fichier DSP sur une application FAUST en fonctionnement pour remplacer son comportement et ce, sans interruption du son. Il est aussi possible de transférer une application qui fonctionne en local, sur une autre machine, même si celle-ci utilise un système d'exploitation différent.

1. INTRODUCTION

FAUST [Functional Audio Stream] [5] est un langage fonctionnel, dédié spécifiquement à la synthèse et au traitement du signal en temps réel. Comparativement à autres langages existants comme Max, PD, Supercollider, Csound, Chuck etc, l'originalité de FAUST est de créer des programmes non pas interprétés mais entièrement compilés [1] . Il offre ainsi une alternative au C/C++ pour l'implémentation d'algorithmes DSP travaillant efficacement à l'échantillon près.

Alors que les compilateurs ont l'avantage de l'efficacité, ils présentent certains désavantages par rapport à des interpréteurs. D'abord, les compilateurs traditionnels ont besoin d'une chaine complète d'outils (compilateurs, éditeurs de liens, librairies de développement, ...). Pour des "non-programmeurs", l'installation de ces outils peut être assez complexe. De plus, le cycle de développement, depuis l'édition du code source jusqu'à l'application exécutée, est bien plus long pour un compilateur que pour un interpréteur. Pour un artiste en situation de création, l'expérimentation rapide est indispensable. Un long cycle de compilation peut donc être un frein. D'autre part, le code binaire n'est pas compatible entre différentes plateformes et systèmes d'exploitation.

FaustLive essaie de réunir le confort d'un langage interpreté autonome avec l'efficacité d'un langage compilé. Grâce à *libfaust*, une librairie qui offre une chaîne de compilation complète en mémoire, FaustLive ne requiert aucun outil externe pour traduire du code FAUST en code machine, et l'exécuter. Grâce à son cycle de développement très court, FaustLive se comporte sur de nombreux points comme un interpréteur FAUST (se rapprochant des environnements modernes LISP compilés, ou de l'approche présentée par A. Graef avec Pure dans [1]).

Basé sur ce court cycle de développement, FaustLive présente des fonctionnalités avancées. Il est, par exemple, possible d'éditer le code source pendant qu'une application FAUST tourne. Le code est alors recompilé dynamiquement, sans interruption du son. Si cette application utilise JACK comme driver, toutes les connexions sont maintenues. FaustLive offre donc une certaine flexibilité pour le prototypage des applications FAUST. D'autre part, lorsque FaustLive fonctionne sur un réseau, il est possible de transférer les calculs audio d'une application sur une machine distante, même si celle-ci utilise un système d'exploitation différent.

Les applications FAUST peuvent aussi être contrôlées à distance, en utilisant les protocoles HTTP ou OSC. Enfin, FaustLive peut se connecter à FaustWeb, un service de compilation distant pour exporter une application en un binaire traditionnel pour l'un des systèmes d'exploitation et une des architectures supportées par la distribution FAUST.

2. LE COMPILATEUR FAUST

Le compilateur FAUST traduit un programme FAUST en son équivalent dans un langage impératif (C, C++, Java, etc), s'occupant de générer du code efficace. La distribution FAUST inclut de nombreux fichiers d'architecture, faisant la liaison entre le code généré et le monde extérieur (drivers audio, interfaces de contrôle, ...).

La version officielle du compilateur FAUST transforme

1 . Les langages interprétés ont souvent recours à une phase de traduction, parfois appelée «compilation», permettant de traduire le programme en une représentation interne, par exemple du bytecode, plus efficace pour l'interprétation. Dans le domaine audio, les langages interprétés ont également recours à une autre technique, le groupage des échantillons par vecteur, de façon à amortir d'avantage le coût de l'interprétation. Néanmoins ces techniques ne permettent pas d'atteindre l'efficacité d'un programme véritablement compilé en code machine, comme c'est le cas pour les programmes FAUST.

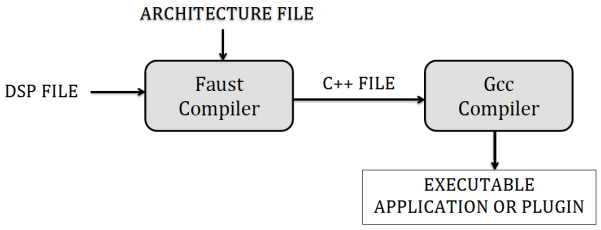


Figure 1. Étapes de la chaine de compilation FAUST

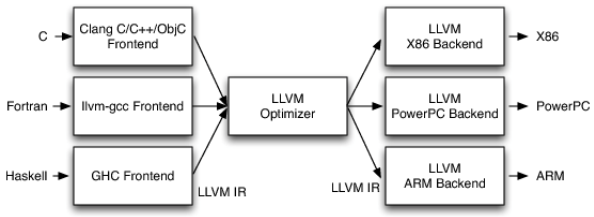


Figure 3. LLVM Compiler Structure

le code DSP en une classe C++, insérée ensuite dans une architecture. Le fichier C++ résultant est finalement compilé avec un compilateur standard pour produire une application ou un plugin (Figure 1).

L'application finale est alors structurée de la manière suivante : le DSP est représenté sous forme d'un module de calcul alors que l'architecture joue le rôle de lien vers le driver audio et vers l'interface graphique (Figure 2).

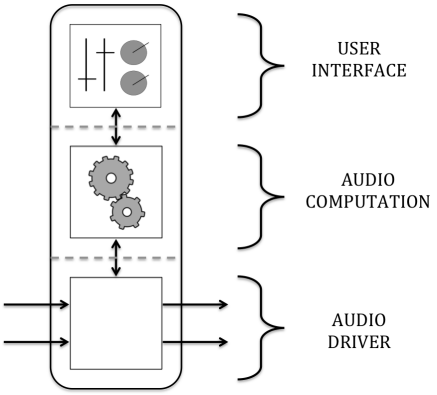


Figure 2. Structure d'une application FAUST

2.1. LLVM

LLVM (anciennement "Low Level Virtual Machine") est une infrastructure de compilateur. Elle est conçue pour compiler, éditer les liens et optimiser des programmes écrits dans un langage arbitraire. Le code exécutable est produit dynamiquement par un compilateur à la volée, à partir d'une représentation spécifique appelée LLVM IR (Représentation Intermédiaire). Clang, le compilateur natif LLVM pour le C/C++/Objective-C est un front-end pour le compilateur LLVM. Il peut, par exemple, convertir un fichier C en code LLVM IR (Figure 3).

Les langages spécialisés, comme FAUST, peuvent facilement produire du code LLVM IR. Un back-end spécifique FAUST vers LLVM IR a été ajouté au compilateur FAUST pour générer ce code, [4].

2.2. Chaîne de Compilation Dynamique

La chaîne de compilation complète démarre donc au code source DSP, compilé en LLVM IR en utilisant le back-end FAUST-LLVM, pour finalement produire du code exécutable grâce au compilateur LLVM-JIT. Toutes les étapes sont réalisées en mémoire. Des pointeurs sur les fonctions exécutables sont ensuite accessibles dans le module LLVM résultant, et leurs codes pourront être appelés avec les paramètres appropriés.

Dans la branche de développement *faust2*, le compilateur FAUST a été compilé sous forme d'une librairie embarquable, *libfaust*, publiée avec une API associée, [2]. Cette API imite le concept des langages orientés objet, comme le C++. L'étape de compilation, habituellement réalisée par GCC, est accessible au travers de la fonction createDSPFactory. À partir du code FAUST sous forme de string ou de fichier, la chaine de compilation embarquée (FAUST + LLVM-JIT) génère le "prototype" de la classe, appelée llvm-dsp-factory. Puis, la fonction *createDSPInstance*, correspondant à un "new nomClasse" du langage C++, permet d'instancier un llvm-dsp. L'instance peut ensuite être utilisée comme n'importe quel objet, fonctionner et être contrôlée à travers son/ses interfaces.

3. FAUSTLIVE - UTILISATION

FaustLive est un logiciel basé sur QT [2], qui permet de lancer des applications FAUST à partir de leur code source, sans avoir à les précompiler. La figure 4 présente donc la transformation des fichiers source FAUST baa.dsp et foo.dsp en applications JACK-QT, une fois déposés dans FaustLive.

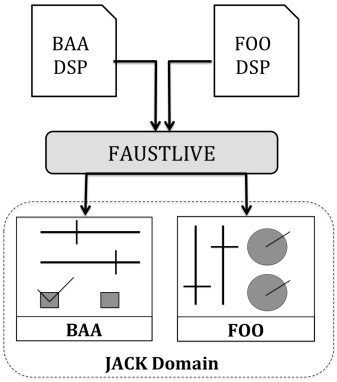


Figure 4. Principe de FaustLive

2 . framework pour la conception d'interfaces graphiques

FaustLive exploite la compilation dynamique, associée à différents systèmes d'interfaçage et drivers audio pour moduler la structure des applications FAUST.
Pour donner une idée du potentiel de FaustLive, la section qui suit présente ses diverses fonctionnalités, montrant pour chacune d'entre elles les altérations correspondantes dans la structure des applications.

Le point de départ de FaustLive est le glisser/déposer. Un DSP FAUST peut être ouvert dans une nouvelle fenêtre, ou bien il peut être glissé sur une application en fonctionnement. Il apparaît alors un état intermédiaire pendant lequel les deux applications co-existent. L'application entrante copie les connexions audio établies, puis les deux comportements audio sont interpolés, via un cross-fade (Figure 5). Pour finir, l'application entrante remplace durablement la précédente. Ce système permet de modifier indéfiniment l'application courante dans une fenêtre, en évitant les interruptions audio.

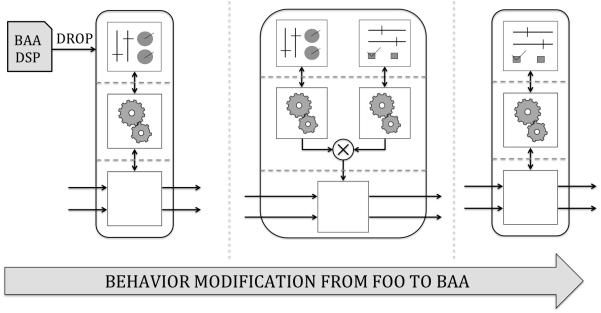


Figure 5. Interpolation de comportements

Ce mécanisme permet aussi l'édition du code source. Lorsqu'un utilisateur choisit d'éditer son code source, il est ouvert dans l'éditeur de texte par défaut. Puis, au moment où les modifications sont enregistrées, l'application est mise à jour, en utilisant l'interpolation de comportements. Cette fonctionnalité est centrale. Elle simplifie grandement le processus de prototypage : un utilisateur peut modifier son code à loisir et entendre/voir le résultat instantanément.

Originairement, JACK a été choisi comme driver audio, puisqu'il permet à l'utilisateur de connecter ses applications FAUST entre elles. Les drivers NetJack, CoreAudio et PortAudio ont ensuite été intégrés à FaustLive donnant à l'utilisateur la possibilité de changer dynamiquement de driver. L'application n'a pas besoin d'être quittée. La migration est réalisée pendant l'exécution de FaustLive et s'applique à toutes les applications FAUST en fonctionnement (Figure 6).

Afin d'offrir une application modulaire, FaustLive offre plusieurs choix d'interfaces de contrôle. Le protocole OSC [3]

3 . Open Sound Control

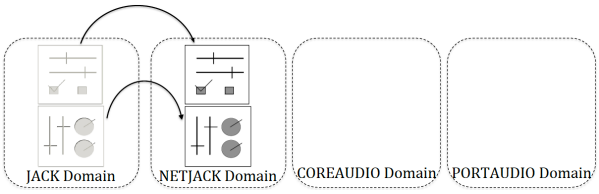


Figure 6. Migration des applications vers un nouveau driver audio

est intégré à FaustLive pour fournir plus d'interopérabilité. De nombreux environnements et matériels audio implémentent ce protocole, grâce auquel FaustLive pourra communiquer avec eux. Un smartphone peut alors ouvrir une interface OSC, contrôlant l'application à distance (Figure 7).

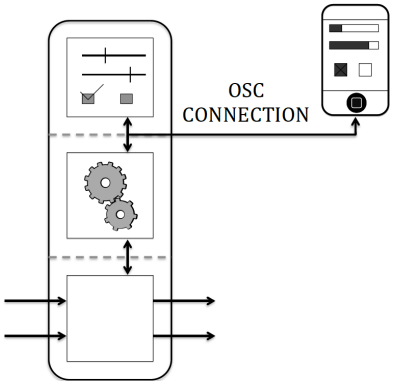


Figure 7. Interface OSC

De même, une interface HTML est accessible à partir d'un QrCode [4] . Une fois scanné, par exemple par une tablette, l'interface est ouverte dans un navigateur. Dans le cas d'OSC comme dans celui d'HTTP, l'interface est dupliquée et une synchronisation est établie entre l'interface locale et celle déportée.

L'interface HTML présente un intérêt additionnel : elle est paramétrée pour accepter le glisser/déposer. L'utilisateur contrôlant l'interface à distance peut donc modifier le comportement de l'application en glissant son DSP sur l'interface. Le code source est envoyé à l'application locale, où elle remplace l'application en fonctionnement par le processus d'interpolation. Enfin, l'interface distante est remplacée (Figure 8).

Dans le cas où de nombreuses applications coûteuses en calculs sont ouvertes, la charge CPU peut saturer le processeur. Le déplacement du calcul sur une machine distante peut donc permettre d'alléger cette charge (Figure 9).

4 . *QR code* (abbréviation de "Quick Response Code") est un type de code barre à deux dimensions

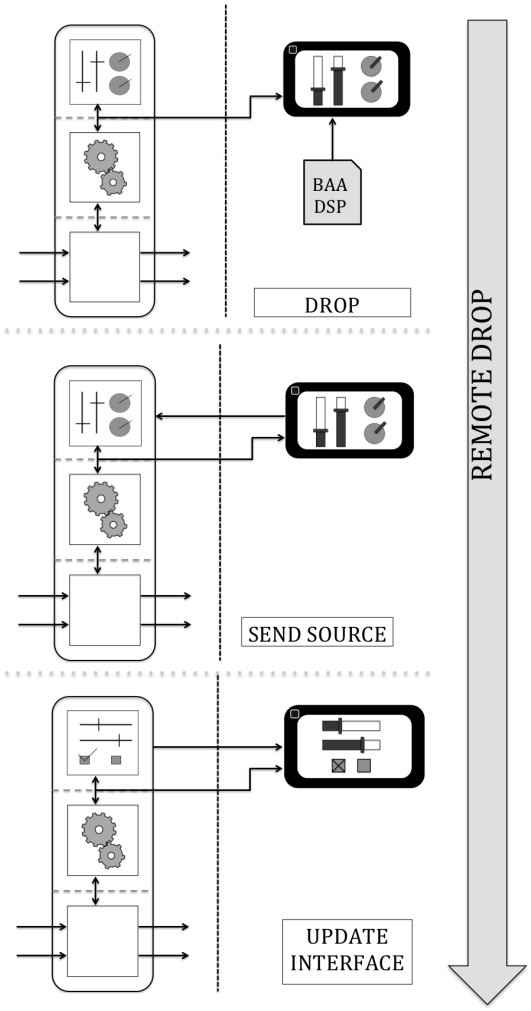


Figure 8. Glisser/Déposer sur une interface distante

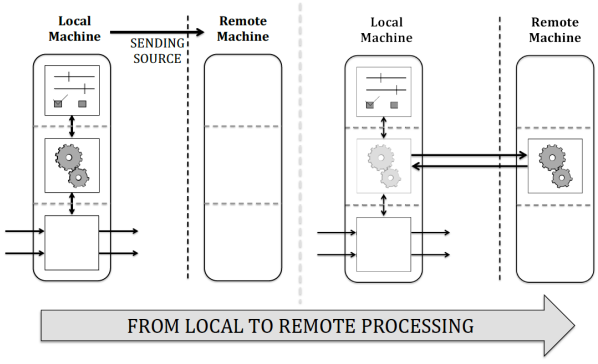


Figure 9. Calcul distribué

Un utilisateur peut vouloir utiliser son application FAUST dans un environnement différent (Max/MSP, SuperCollider, ...). Dans ce but, un lien au service web de compilation, FaustWeb, est intégré dans FaustLive. L'utilisateur doit seulement choisir la plateforme et l'environnement qu'il désire cibler. En retour, il reçoit l'application binaire ou le plugin demandé.

Lorsque FaustLive est quitté, la dernière configuration est sauvegardée et sera restaurée à la prochaine exécution. D'autre part, l'utilisateur peut choisir d'enregistrer l'état de l'application à n'importe quel moment. Il pourra ensuite recharger son "snapshot" en l'important dans l'état courant ou en le rappelant (Figure 10).

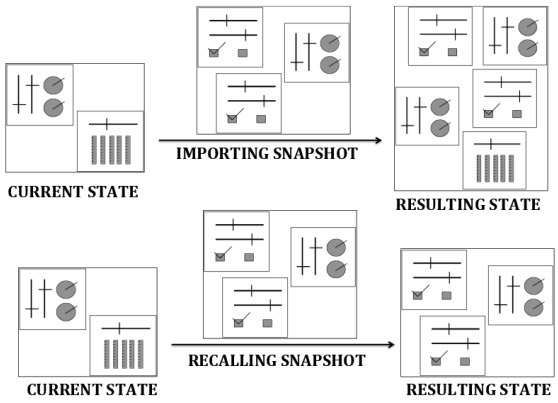


Figure 10. Reloading Snapshot

4. FAUSTLIVE - ASPECTS TECHNIQUES

4.1. Fonctionnalités basiques

Le but principal de FaustLive est de créer un environnement dynamique pour le prototypage d'applications FAUST en embarquant *libfaust*. Les fenêtres QT, une fois paramètrées pour accepter le glisser/déposer, permettent à l'utilisateur de glisser son code FAUST sous forme de fichier, de string ou d'url. Le code est immédiatement donné au compilateur embarqué par le biais de la fonction *createDSPFactory* de l'API libfaust (cf 2.2). L'avantage de la chaîne de compilation résultante (Figure 11) est d'accélérer le processus de compilation, en retournant quasiment instantanément les pointeurs sur les fonctions exécutables. Le nouveau processeur audio remplace alors l'application courante par le biais d'un crossfade, évitant une coupure brutale du son.

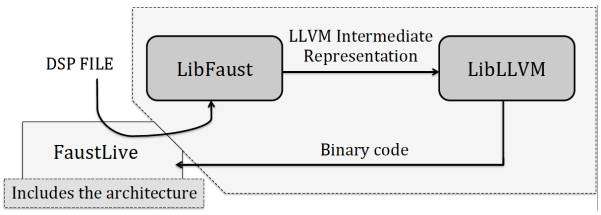


Figure 11. Compilation chain in FaustLive

Un avantage important de FaustLive est la coexistence de plusieurs applications FAUST, en opposition avec l'ar-

chitecture QT-JACK de la distribution FAUST "statique" où chaque programme doit être compilé séparément pour produire une application propre. L'environnement résultant est présenté Figure 12.

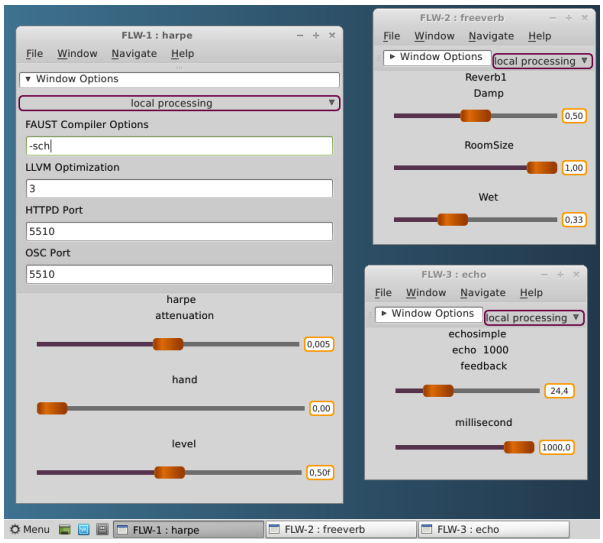


Figure 12. FaustLive's environment

4.2. Audio Drivers

FaustLive a intégré JACK, CoreAudio, NetJack et PortAudio. Selon la plateforme, les librairies sont ou non compilées dans FaustLive [5] . Grâce à cette diversité de drivers, il est possible de basculer de l'un à l'autre ou de modifier les paramètres (taille de buffer, fréquence d'échantillonnage, ...) durant l'exécution de FaustLive. Tous les clients audio sont d'abord arrêtés, puis les applications sont transférées dans le nouveau domaine pour être redémarrées.

4.2.1. JACK

JACK est un système qui gère en temps réel de l'audio/midi basse-latences. Il fonctionne sous GNU/Linux, Solaris, FreeBSD, OSX et Windows. Il peut connecter plusieurs applications à un dispositif audio. De plus, il permet de partager des flux audio entre applications.

Une contrainte intéressante amenée par l'utilisation de JACK a donc été la question des connexions audio. Lorsque des connexions ont été établies, l'objectif est de les conserver, quels que soient les changements d'application FAUST dans une fenêtre. Lorsqu'une nouvelle application entre dans la fenêtre, le graphe des connexions JACK est sauvé sous forme de fichier. Ce dernier est parsé pour remplacer le nom du client sortant par le nom du nouveau processeur audio. Le nouveau client JACK peut alors être connecté à la manière de son prédecesseur. Si la nouvelle application a plus de ports que la précédente, l'utilisateur devra faire les connexions lui-même.

5 . JACK, CoreAudio et NetJack sont utilisés sur OSX, JACK et NetJack sur Linux, PortAudio, JACK et NetJack sur Windows.

4.2.2. NetJack

NetJack est un système de transport audio en temps réel à travers un réseau IP. Il est intégré à JACK. NetJack synchronise ses clients sur une unique carte son pour qu'il n'y ait pas de rééchantillonnage ou de click. Le "master" impose sa fréquence d'échantillonnage, ainsi que sa taille de buffer, en liaison avec le dispositif audio qu'il utilise. Grâce à NetJack, plusieurs utilisateurs peuvent envoyer leurs sorties audio sur une unique carte son. Une utilisation typique de ce système est une classe où seule la machine du professeur est connectée au système audio et les élèves envoient leurs sorties audio à travers NetJack.

4.2.3. CoreAudio et PortAudio

Sous OSX, l'application FaustLive en version JACK n'est pas autonome, elle nécessite que l'utilisateur installe JACK par ailleurs. Pour faciliter l'installation de FaustLive aux utilisateurs débutants, une version *CoreAudio* [6] , ainsi qu'une version *PortAudio* [7] ont été créées. Incluses par défaut dans les systèmes ou facilement distribuées sous la forme de DLL, ces architectures n'ajoutent pas de contrainte d'installation supplémentaire pour l'utilisateur.

4.3. Interfaces de contrôle

Pour pouvoir contrôler l'interface utilisateur à distance, un port UDP doit être ouvert pour le protocole OSC et un port TCP pour la connexion HTTP. Les deux protocoles utilisent par défaut le port 5510 et sont configurables dans la barre d'options de la fenêtre. Lorsque le port est utilisé, le système utilise automatiquement le prochain port disponible.

4.3.1. Interface HTML

Lorsqu'une interface HTML est construite, un serveur HTTP est démarré et s'occupe de délivrer la page HTML (Figure 13). Ce serveur est géré par la librairie *libmicrohttpd*.

Pour simplifier l'accès à cette interface, un QrCode est construit à partir de l'adresse HTTP, grâce à la librairie *libqrencode*. La plupart des smartphones et des équipements portables sont équipés de décodeurs de QrCode. En scannant le code, un navigateur se connecte à la page de l'interface.

4.3.2. Glisser/déposer à distance

Comme le reste de la distribution FAUST, l'interface HTML a un comportement "statique". Pour dynamiser son comportement à l'image des interfaces locales de FaustLive, une zone de glisser/déposer a été ajoutée à l'interface HTML. Ce service HTTP est indépendant et spéci-

6 . *CoreAudio* est l'infrastructure audio de iOS et OS X. Elle offre un framework pour manipuler de l'audio dans les applications.

7 . *PortAudio* est une librairie audio écrite en C/C++. Elle est libre et cross-platformes. Son intention est de promouvoir le portage d'application audio entre différentes plateformes.

fique à FaustLive. Le serveur, démarré par FaustLive, est capable de créer une page HTML qui encapsule les interfaces de contrôle. Les interactions de cette page sont gérées avec Javascript. Le service résultant (interface de contrôle + glisser/déposer de DSP) a l'adresse suivante : http ://adresseIP :portServiceGlisserDéposer/portInterface-Controle (Figure 13).

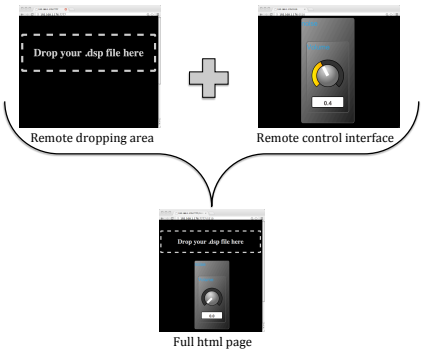


Figure 13. Interface HTML avec le service de glisser/déposer

Le port du service de glisser/déposer est configuré dans les préférences générales de FaustLive et il est commun à toutes les applications FAUST. Les interfaces de contrôle ont, elles, chacune un port différent, modifiable dans les options de la fenêtre.

La réaction à un glisser/déposer sur l'interface distante suit le même modèle que sur l'interface locale. Le code DSP est d'abord envoyé à FaustLive par une requête HTTP, POST. Le code est alors compilé et l'application est remplacée après le mécanisme de crossfade. Enfin, l'interface distante est remplacée.

4.4. Calcul distribué

Pour élagir ses bénéfices, FaustLive permet de faire du calcul à distance. La compilation ainsi que le calcul audio sont redirigés sur la machine distante. La charge du processeur local peut ainsi être réduite.

Sur la machine distante, une application démarre un serveur HTTP, offrant le service de calcul distribué. Ce serveur attend des requêtes de compilation/calcul.

Du côté du client (FaustLive), une API "proxy" rend transparent la création d'un remote-dsp plutôt qu'un llvm-dsp local (c.f 2.2). Cette API, *libfaustremote*, prend soin d'établir la connexion avec le serveur.

La première étape (la compilation) est exécutée par la fonction *createRemoteDSPFactory*. Le code est envoyé au serveur, qui le compile et crée la "réelle" llvm-dsp-factory. La remote-dsp-factory est, pour l'utilisateur, une image de la "véritable" factory. Avant d'envoyer le code FAUST, une étape de compilation FAUST-FAUST résout les dépendances du code à ses librairies. De cette manière, le code envoyé est indépendant et pourra être correctement compilé du côté serveur. La remote-dsp-factory peut ensuite être instanciée pour créer des remote-dsp instances, qui peuvent être lancées dans l'architecture audio et visuelle choisie, ici FaustLive (Figure 14).

Pour être capable de créer l'interface localement, le serveur renvoie une interface encodée en JSON [8] . De cette manière, la fonction *buildUserInterface* peut être recréée pour donner l'impression qu'un remote-dsp fonctionne de la même manière qu'un dsp local.

De plus, les calculs audio sont redirigés à travers une connexion NetJack. Les données audio sont envoyées à la machine distante qui traite les données avant de renvoyer ses résultats. Outre le flux audio standard, un port midi est utilisé pour transférer les valeurs des contrôleurs. L'intérêt de cette solution est de transmettre les buffers audio et les contrôleurs, synchronisés dans la même connexion. D'autre part, les échantillons audio peuvent être encodés en employant les différents types de données audio : float, integer et audio compressé [9] .

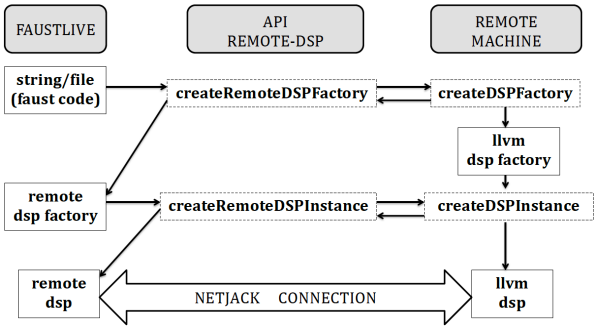


Figure 14. Compilation et Calcul distribués

libfaustremote utilise la librairie *libcurl* pour envoyer des requêtes HTTP au serveur de calcul, lui même manié avec *libmicrohttpd*.

Sur chaque fenêtre de FaustLive, l'accès au service de calcul à distance est simple. Le protocole ZeroConf est utilisé pour scanner les machines distantes présentant ce service. Une liste est construite dynamiquement à partir des machines disponibles. En naviguant dans cette liste, l'utilisateur peut ensuite faire basculer son application FAUST d'une machine à l'autre très facilement.

4.5. FaustWeb

Pour simplifier l'accès à la compilation FAUST, un service de compilation web a été conçu. Il reçoit du code FAUST et renvoie une application ou un plugin pour l'architecture et la plateforme choisie. Dès lors, l'installation de la distribution complète FAUST n'est plus nécessaire. N'importe qui peut décrire un programme FAUST, l'envoyer au serveur puis utiliser le plugin qu'il aura reçu.

8 . *JavaScript Object Notation* est un format de données textuelles, dérivé de la notation des objets du langage JavaScript.

9 . à l'aide du codec OPUS : http ://www.opus-codec.org

Ce service est accessible depuis un navigateur mais nécessite plusieurs requêtes. À travers FaustLive, l'export des applications est facilité. Un menu est construit dynamiquement avec les plateformes et les architectures disponibles, encodées sous forme de JSON . Tandis que l'utilisateur fait le choix d'exporter son application, le code FAUST correspondant est envoyé au serveur. Ce dernier vérifie la syntaxe FAUST. Une clé SHA1 unique correspondant à ce DSP est générée pour faire ensuite les requêtes de différents binaires. La seconde étape est la compilation, utilisant la chaîne de compilation "statique" réalisée coté serveur, pour renvoyer enfin l'application choisie par l'utilisateur (Figure 15). Pour réaliser les requêtes au service FaustWeb, FaustLive utilise la librairie NetWork du framework QT.

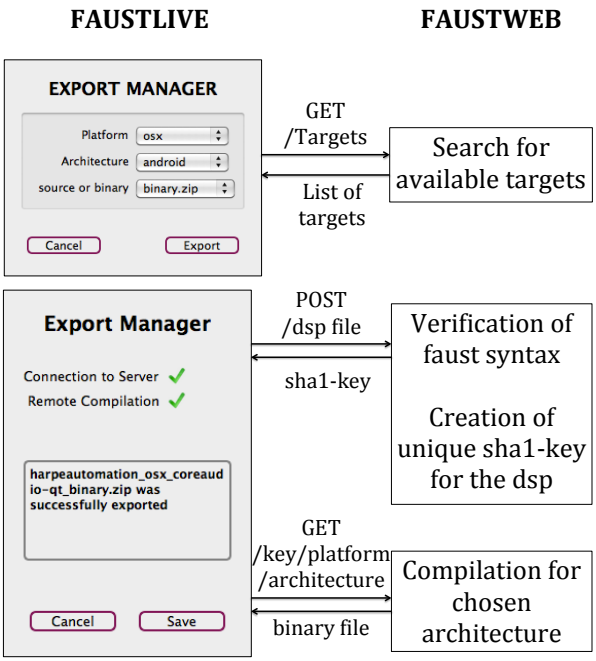


Figure 15. Les étapes de la chaîne de compilation à travers FaustLive

4.6. Concept de session

Un concept de session est conservé pour préserver l'état de l'application (valeur des paramètres, position sur l'écran, connexions audio, options de compilation, ...). Quand l'application est quittée, la session est enregistrée par défaut pour être restaurée à la prochaine exécution. L'utilisateur peut aussi sauvegarder l'état de l'application complet dans un "snapshot", enregistré dans un dossier, xxx.tar.

Toutes les ressources locales nécessaires (fichiers DSP, ...) sont copiées dans le dossier, xxx.tar, rendant indépendants les "snapshots" de FaustLive. Les chemins d'accès vers les ressources originelles sont utilisés prioritairement. Cependant, si un fichier source est supprimé ou si un "snapshot" est déplacé sur une autre machine, les copies doivent être utilisées.

Pour réduire le temps de compilation d'un DSP, la sortie du compilateur FAUST (le code LLVM IR optimisé) est enregistrée. Lorsqu'une application est rechargée, la phase de compilation FAUST du code DSP au code LLVM IR est alors économisée, ainsi que la phase d'optimisation du code LLVM IR. Pour des programmes couteux, le gain est notable. On peut passer de quelques secondes de compilation à une exécution instantanée.

4.7. Limites

Certains aspects techniques limitent l'ouverture d'un nombre indéfini de fenêtres FaustLive.

D'abord, pour pouvoir synchroniser la recompilation avec l'enregistrement du fichier source, un moniteur est placé sur le fichier, ce qui consomme des ressources systèmes. Selon la plateforme utilisée, cette limite est plus ou moins grande. De plus, il existe une limitation du nombre de liaisons OSC parallèles. Enfin, le nombre de clients JACK est limité.

Pour toutes ces raisons, un maximum de soixante applications FAUST peuvent être ouvertes parallèlement.

5. CONCLUSION

FaustLive réunit le confort d'un langage interprété autonome et l'efficacité d'un langage compilé. FaustLive offre actuellement le cycle de développement le plus rapide pour des applications FAUST, fournissant un outil modulaire pour le prototypage d'applications FAUST. Elle intègre aussi des fonctionnalités avancées de calcul distribué et de contrôle à distance pour des applications audio temps-réel. De plus, FaustLive met à disposition, à travers sa fonction d'export, un front-end confortable pour FaustWeb, le service de compilation web.
Ce projet est open-source et disponible sur SourceForge [3]. Il fonctionne sur Linux, OS X et Windows.

6. REMERCIEMENTS

Ces développements ont été menés dans le cadre des projets INEDIT [ANR-12-CORD-0009] et FEVER [ANR-13-BS02-0008], soutenus par l'Agence Nationale pour la Recherche.

7. REFERENCES

[1] A. Graef. Functional signal processing with pure and faust using the llvm toolkit. 2011.

[2] faust2 repository. http ://sourceforge.net/p/faudiostream /code/ci/faust2/tree/.

[3] faustlive repository. http ://sourceforge.net/p/faudiostream /faustlive/ci/master/tree.

[4] S. Letz, Y. Orlarey, and D Fober. Dynamic compilation of parallel audio applications. 2013.

[5] Y. Orlarey, S. Letz, and D. Fober. Faust : an efficient functional approach to dsp programming. 2009.

CECILIA 5, LA BOÎTE À OUTILS DU TRAITEMENT AUDIO-NUMÉRIQUE

Olivier Bélanger
iACT (institut Arts Cultures et Technologies)
Université de Montréal, Faculté de Musique

RÉSUMÉ

Le logiciel de traitement sonore Cecilia est une boîte à outils pour la manipulation du son, utilisée autant en pédagogie qu'en production musicale. Grâce à sa panoplie de modules de traitement originaux et à son interface graphique modulaire, Cecilia représente un environnement idéal pour l'exploration et la création sonore. La version 5 [2], dont le développement a débuté en octobre 2011, constitue une réécriture complète de l'infrastructure du logiciel. Une modernisation qui aura permis l'ajout de plusieurs nouvelles fonctionnalités au logiciel. Cet article se divise en deux sections. Dans un premier temps, seront illustrés les mécanismes que met en place le logiciel afin de favoriser une expérience de travail efficace et créative. Nous y retrouverons des outils tels que le grapheur, la lecture à boucle variable de fichiers son, la communication MIDI [1] et OSC [2] ainsi que la génération algorithmique de trajectoires de contrôle. Ensuite, sera détaillé l'API [3] de Cecilia, qui offre une interface simplifiée à l'utilisateur désirant développer de nouveaux modules, tout en bénéficiant de la flexibilité de l'interface graphique fournie par l'environnement. Cecilia 5 est un logiciel gratuit, de sources libres et fonctionnant sur toutes les plates-formes majeures.

1. INTRODUCTION

Cecilia fut initialement développé, de 1995 à 1998, par Jean Piché et Alexandre Burton [4] à la faculté de musique de l'Université de Montréal. L'objectif initial du projet était de pourvoir les studios de composition d'une plateforme numérique unifiée comme environnement de travail. La structure modulaire du logiciel permettait, à l'intérieur d'un environnement unique, d'effectuer une multitude de tâches reliées à la création de musiques concrètes. La version initiale de Cecilia fut écrite en TCL/TK, une combinaison constituée d'un langage de programmation interprété (TCL) et d'une bibliothèque graphique (TK), permettant le développement rapide d'applications multiplateformes. Le moteur audio utilisé pour effectuer les tâches relatives au traitement de signal était Csound [6], un langage de programmation pour la création sonore développé par Barry Vercoe au MIT. Jean Piché a poursuivi le développement de Cecilia, en conservant la structure originale, jusqu'en 2008. La version officielle était alors Cecilia 2.5. En 2009, afin de permettre l'ajout d'un certain nombre de fonctionnalités et une maintenance plus aisée du logiciel, un changement d'architecture s'opère et Cecilia 4 est reconstruit à neuf en remplaçant TCL par Python [5], un autre langage de programmation interprété, plus moderne, orienté-objet et soutenu par une communauté très active. L'interface de Cecilia est complètement redessinée avec la librairie graphique WxPython [3] et Csound demeure le langage assurant les services audio de l'application. Cette version sera maintenue et développée, par Olivier Bélanger et Dominic Thibault, jusqu'en 2011, année où Csound fût remplacé par pyo [1], un module python, créé par Olivier Bélanger, dédié au traitement de signal audio. L'utilisation d'un module python comme moteur audio, plutôt qu'un langage de programmation externe tel que Csound, offre certains avantages au niveau du design du logiciel. D'une part, l'intégration du moteur audio à l'interface de contrôle est grandement simplifiée en ce que les deux existent dans le même environnement de programmation. Pour communiquer avec Csound, les versions antérieures de Cecilia devaient avoir recours soit à une traduction pour python de l'API de Csound, soit au protocole de communication OSC [7]. Ces deux méthodes imposaient plusieurs contraintes en ce qui concerne le développement de nouveaux paradigmes de contrôle du processus audio. Avec un module audio dédié au langage python, le transfert des données de l'interface graphique vers les paramètres du processus sonore s'effectue sans intermédiaire, de façon plus directe que lorsque deux logiciels indépendants tentent de communiquer entre eux. Enfin, un langage de programmation orienté-objet tel que python offre beaucoup plus de flexibilité concernant l'écriture d'algorithmes audio et de contrôle, la gestion de la polyphonie ainsi que la manipulation en temps-réel des paramètres du son. Ce changement majeur dans l'architecture du logiciel a donné naissance à Cecilia 5, la version actuelle du logiciel.

Dans les versions précédentes, Cecilia était composé de deux éléments d'interface distincts. Une première fenêtre présentait un éditeur de texte, optimisé pour l'écriture de programmes Csound, permettant de développer des modules de traitement à même le logiciel. Ensuite, une seconde fenêtre contenant l'interface graphique de jeu, c'est-à-dire le grapheur, les potentiomètres et plusieurs autres

1 . *Musical Instrument Digital Interface*
2 . *Open Sound Control*
3 . *Application Programming Interface*

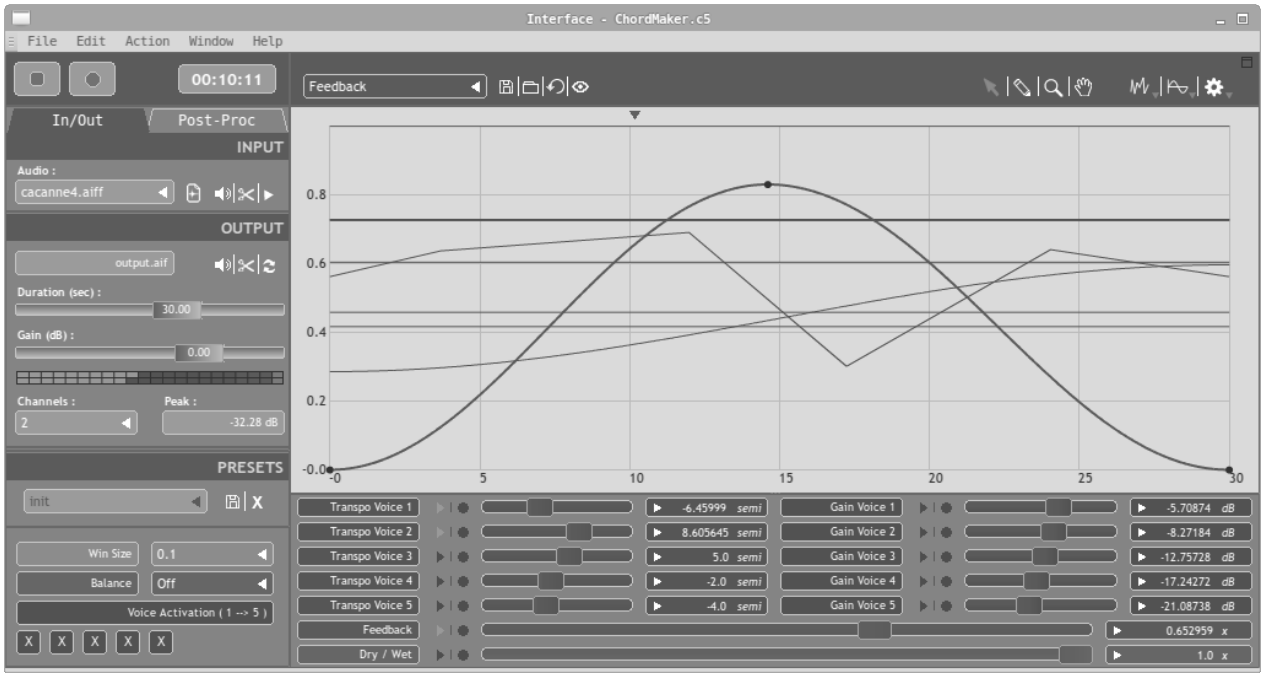


Figure 1. Fenêtre principale de l'interface graphique de Cecilia 5. On y retrouve différentes sections. En haut à gauche : le panneau de transport. Au centre à gauche : le panneau de contrôle avec les commandes d'entrée, de sortie et de sauvegarde des pré-réglages. En bas à gauche : la section des menus et boutons. Au centre de l'interface : le grapheur où sont éditées les automations de paramètres. Sous le grapheur : la section dédiée aux potentiomètres.

outils de contrôle, permettait de manipuler le processus audio en temps réel. Les modules de traitement audio de la version 5 étant écrit avec le langage python, il a été décidé d'alléger la structure du logiciel en retirant l'éditeur de texte. Il existe déjà une grande quantité d'éditeurs de texte gratuits et multi-plateformes, où le langage python est automatiquement reconnu, pouvant servir à l'écriture des modules. Olivier Bélanger est maintenant le développeur principal de Cecilia 5, alors que Jean Piché demeure le père spirituel du projet. Jean-Michel Dumas a grandement contribué à la traduction des modules Csound (de la version 4) en modules pyo pour la version 5.

2. UTILISATION DU LOGICIEL

L'objectif de base de l'application consiste à fournir un environnement simple et efficace afin d'explorer les effets audio-numériques sur le son. Une bibliothèque de modules, couvrant un large éventail de traitements, est fournie avec le logiciel. L'utilisateur, après avoir chargé un son dans le module choisi, peut manipuler et automatiser les différents paramètres du processus à l'aide des éléments d'interface graphique. Différentes fonctions d'enregistrement permettent de sauvegarder le résultat sonore sur le disque dur.

Dans cette section seront décrits les principaux éléments de l'interface graphique (Figure 1) et les possibilités de création qu'ils offrent à l'utilisateur. Nous y retrouverons les différents types de sources sonores à traiter, les modes d'enregistrement du rendu sur le disque dur, les techniques proposées pour moduler les paramètres au cours du temps et quelques utilitaires favorisant une expérience créative avec Cecilia.

2.1. Source sonore

La porte d'entrée du logiciel est la source sonore à modifier. Traditionnellement, Cecilia était conçu pour traiter un fichier son lu en boucle par l'application. Au fil des versions, la lecture en boucle du son a été grandement bonifiée, au point de devenir un élément important en tant que traitement audio dans le module. Divers modes de jeu, permettant notamment l'accès aux signaux provenant de la carte de son, ont aussi été ajoutés. Ces fonctionnalités, accessibles en cliquant sur l'icône à la droite du menu, font de Cecilia un environnement de traitement en temps réel simple d'approche et versatile. La configuration du mode de jeu et le choix de la source sonore s'effectue à l'aide de la section « Input » du panneau de contrôle.

Figure 2. Panneau de configuration de la source sonore.

2.1.1. Lecteur de son en boucle

Lorsqu'un son est chargé en mémoire dans le lecteur, la fenêtre de contrôle de la boucle peut être affichée en cliquant sur le petit triangle horizontal, dernier icône de la boîte d'outils à droite du menu (Figure 2).

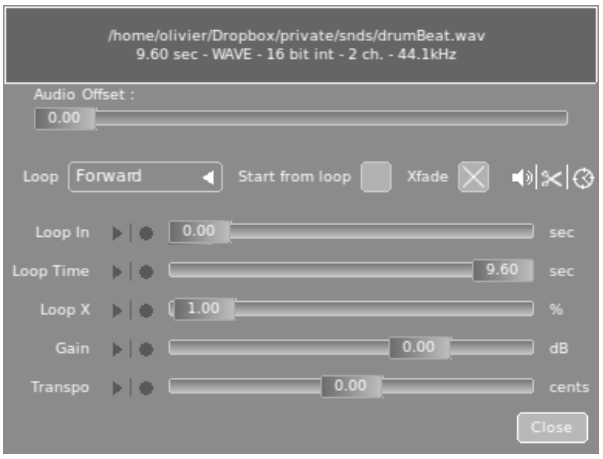


Figure 3. Fenêtre de contrôle du lecteur en boucle.

Cette fenêtre permet de spécifier les paramètres de la lecture, c'est-à-dire le point de départ de la boucle, sa durée, le temps de fondu-enchaîné aux points de bouclage, le volume du signal ainsi qu'un facteur de transposition en demi-tons. Tous ces paramètres peuvent être asservis à des contrôleurs externes, via le protocole MIDI ou le protocole OSC, et enregistrés dans le grapheur en tant qu'automations.

2.1.2. mode 1 : Fichier son

Le mode de jeu principal du logiciel (icône de fichier sonore) consiste à charger un fichier son, à l'aide du menu de la section « Input », dans la mémoire du lecteur. La lecture de ce son servira de signal source au traitement mis en place par le module choisi. Pour ouvrir un fichier son, il suffit de cliquer sur le menu, de naviguer dans la hiérarchie de dossiers à l'aide du dialogue présenté, puis de sélectionner le fichier désiré. En plus du fichier sélectionné, tous les fichiers sons contenus dans le même dossier viendront peupler le menu, offrant ainsi un accès rapide à une bibliothèque de sources sonores. On affiche le contenu en cliquant sur la petite flèche à droite du menu.

2.1.3. mode 2 : Entrée micro

Le deuxième mode de jeu (icône de micro) permet d'utiliser le signal en provenance des entrées de la carte de son comme source sonore à modifier. Configuré ainsi, Cecilia devient un environnement de traitement en temps réel à la fois simple d'utilisation et possédant des contrôles sophistiqués.

2.1.4. mode 3 : Entrée micro et lecteur en boucle

Le troisième mode de jeu (icône de micro accompagné du chiffre 1) permet à l'utilisateur d'enregistrer le signal d'entrée dans la mémoire du lecteur en boucle. La durée de la mémoire est spécifiée dans la fenêtre de contrôle de la boucle. Ce mode est utile pour manipuler un signal réel, qui n'a pas été préalablement enregistré sur le disque dur, tout en bénéficiant des fonctionnalités du lecteur en boucle.

2.1.5. mode 4 : Entrée micro continue et lecteur en boucle

Le dernier mode (icône de micro entouré de flèches circulaires) est une variante du mode précédent, où la mémoire du lecteur en boucle est constamment renouvelée par le signal réel en entrée de la carte de son. Pour éviter les artefacts causés par le bris de la forme d'onde au croisement des pointeurs de lecture et d'écriture, deux espaces-mémoire sont utilisés. La lecture et l'écriture alternent, hors phase, entre les deux mémoires. Le lecteur en boucle manipule donc un matériau en constante évolution.

2.2. Sortie audio

La configuration du signal de sortie s'effectue via la section « Output » du panneau de contrôle (Figure 4). On y retrouve, entre autres, le nom du fichier son à enregistrer, la durée totale de la performance (directement reliée aux automations du grapheur), un contrôle de volume global ainsi qu'un menu permettant de spécifier le nombre de canaux audio de sortie. Cette section est toujours présente, peu importe le module choisi pour traiter le signal. Les caractéristiques générales du signal, telles que la fréquence d'échantillonnage, la quantification, le choix des interfaces audio et MIDI ou le format audio désiré peuvent être modifiées par le biais de la fenêtre de préférences du logiciel.

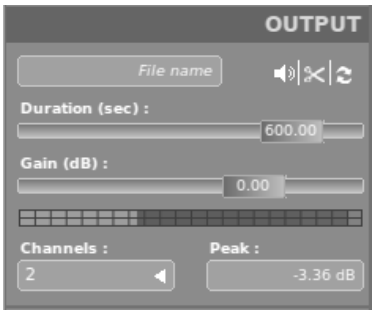


Figure 4. Panneau de configuration de la sortie audio.

Divers modes d'écoute et de sauvegarde sont disponibles pour rendre le travail dans l'environnement efficace.

2.2.1. Panneau de transport

Le panneau de transport (Figure 5) est situé dans le coin supérieur gauche de l'interface. Il permet d'activer la per-

formance, à l'aide du bouton *play*, pour la durée totale spécifiée dans la section « Ouptut ». Le bouton d'enregistrement, en plus d'activer la performance, enregistre en temps réel le signal de sortie sur le disque dur. Lorsque la performance est lancée en temps réel, le curseur situé au-dessus du grapheur entreprendra sa course et toutes les manipulations effectuées dans l'interface (menus, boutons, potentiomètres, lignes de graphe) auront des répercutions immédiates sur le signal de sortie.

Figure 5. Panneau de transport.

2.2.2. *Écriture sur le disque*

Pour les performances de longues durées, où l'évolution des paramètres est entièrement configurée à même le grapheur, il y a possibilité de sauvegarder le résultat sonore, en temps différé, dans un fichier sur le disque dur. Cecilia calculera alors le signal le plus vite possible, en fonction de la puissance du processeur. Cette fonction est accessible sous le menu *Action* → *Bounce to Disk*.

2.2.3. *Exportation par lot*

Plusieurs fichiers sonores peuvent être créés automatiquement avec les fonctions d'exportation par lot. Deux méthodes d'exportation sont disponibles. La première, accessible sous le menu *Action* → *Batch Processing on Preset Sequence*, permet d'appliquer tous les pré-réglages enregistrés (voir section 2.4.2), un par fichier généré, sur le fichier sonore courant. Cette fonction est très utile pour créer une séquence de sonorités ayant une source commune. La seconde méthode, activée via *Action* → *Batch Processing on Sound Folder*, permet d'appliquer le processus courant sur tous les sons chargés dans le menu, avec option d'ajuster automatiquement la durée des sons générés sur la durée des sons originaux. Cette fonction est particulièrement utile lorsque plusieurs sons doivent être traités de façon similaire.

2.3. Modulation des paramètres au cours du temps

Une des grandes forces de Cecilia réside dans les différents paradigmes offerts pour la génération de trajectoires de paramètres. Cette section exposera les principaux outils permettant de créer des traitements audio originaux et dynamiques.

2.3.1. *Le grapheur*

Le grapheur (Figure 6) est l'élément central de l'application. Tous les paramètres continus (potentiomètres inclus) y sont représentés sous la forme d'une ligne de couleur différente. Lorsque la performance est lancée, chacun des paramètres dont l'automation est activée se verra soumis à la trajectoire qui lui est associée dans le grapheur.

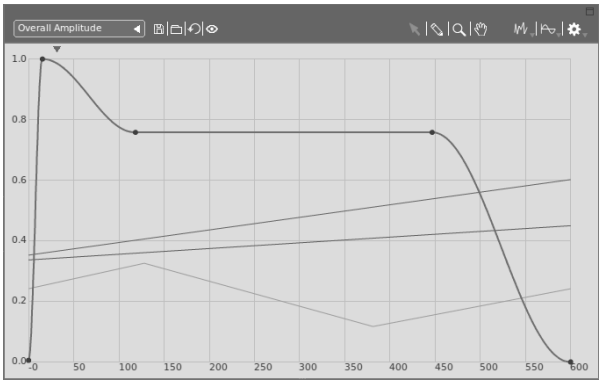


Figure 6. Gestion de l'évolution des paramètres au cours du temps.

La barre d'outils situé au-dessus du grapheur est divisée en trois sections :

1. La section de gauche (Figure 7) offre les outils pour la gestion des trajectoires dans le grapheur. On y retrouve le menu de sélection, la sauvegarde et l'ouverture de fichier, un bouton de ré-initialisation ainsi qu'un commutateur de visibilité.

Figure 7. Gestion des trajectoires du grapheur.

2. La section du centre (Figure 8) permet la sélection des outils de manipulation graphique. Le pointeur permet de déplacer, sélectionner et créer des points sur une trajectoire. Le crayon permet de dessiner des courbes en mode « main libre ». La loupe permet d'agrandir une section du grapheur afin d'effectuer des manipulations détaillées sur une portion de trajectoire. Enfin, en mode agrandi, la main permet de déplacer la région visible du grapheur.

Figure 8. Outils de manipulation du grapheur.

3. La section de droite (Figure 9) donne accès à divers algorithmes de génération ou de manipulation de trajectoire. On y retrouve des algorithmes de génération aléatoire (voir section 2.3.2), des générateurs de formes d'onde diverses ainsi qu'une série d'algorithmes de manipulation de trajectoire. Ces algorithmes permettent notamment de compresser (ou de dilater) un groupe de points, d'interpoler entre deux points ou d'ajouter des déviations aléatoires.

Figure 9. Générateurs et modificateurs de trajectoire.

2.3.2. Les générateurs algorithmiques

Les générateurs algorithmiques, dont les contrôles sont illustrés sur la figure 10, offre à l'utilisateur une méthode simple et puissante pour créer des trajectoires aléatoires aux polarités influencées par l'algorithme choisi. Un choix de courbes, telles que uniforme, gaussienne, weibull et beta, ainsi que des algorithmes de génération musicale, comme la marche aléatoire, les répétitions et les segments bouclés, sont disponibles via le premier menu de la fenêtre. Deux types d'interpolation sont offerts dans le second menu. Il y a l'interpolation linéaire, où chacun des points est relié au suivant par une ligne droite, et le *sample and hold*, où la valeur d'un point est tenue jusqu'au point suivant. Viennent ensuite une série de potentiomètres permettant de spécifier la quantité de points à générer, les balises minimum et maximum à l'intérieur desquelles la génération sera limitée ainsi que les paramètres de contrôle liés à l'algorithme sélectionné. Les trajectoires générées constituent un excellent point de départ pour une paramétrisation dynamique d'un processus musical.

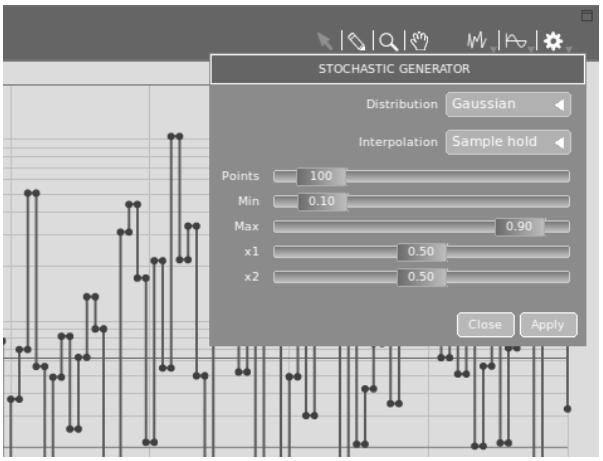


Figure 10. Génération algorithmique de trajectoires.

2.3.3. Automation de paramètres

Tous les paramètres continus, qu'ils appartiennent au module lui-même, au lecteur de son en boucle ou aux effets post-traitement (voir section 2.4.1), peuvent être asservis à un contrôleur externe via le protocole MIDI ou le protocole OSC. La figure 11 illustre l'assignation de canaux OSC aux fréquences de coupure de deux filtres ainsi que l'assignation d'un contrôleur MIDI sur le paramètre de mixage des filtres. Les connexions sont indiquées sur le fond du potentiomètre et sauvegardées à même le module lorsque celui-ci est enregistré. On double-clique sur l'étiquette du potentiomètre pour ouvrir la fenêtre d'assignation de canaux OSC ou on clique avec le bouton droit pour lancer la détection automatisée de contrôleur MIDI (fonction *midi learn*).

Figure 11. Assignation de contrôleurs externes aux potentiomètres.

2.4. Les utilitaires

Parmi les fonctionnalités présentes dans le logiciel, il y en a deux qui permettent à l'utilisateur de personnaliser les processus appliqués au signal source. D'abord, l'onglet « Post-Processing » du panneau de contrôle expose une interface où des effets supplémentaires peuvent être appliqués sur le signal audio produit par le module. Ensuite, un système de pré-réglages permet de sauvegarder indépendamment plusieurs états du module et de les rappeler à volonté.

2.4.1. Post-traitement

Le panneau de post-traitement met à la disposition de l'utilisateur un certain nombre d'effets, aux contrôles simplifiés, à la manière des *plugins* dans un séquenceur. Ces effets, entièrement automatisables dans le grapheur, sont appliqués en série sur le signal produit par le module courant. Ce système permet notamment de personnaliser un module en lui ajoutant une ou plusieurs couches de traitements supplémentaires. On y retrouve un éventail d'effets classiques, tels que la réverbération, le filtrage, la distorsion, l'harmonisation, la compression, etc.

Figure 12. Panneau des traitements post-module.

2.4.2. Pré-réglages

La section « Presets » (Figure 13) permet de sauvegarder l'état du module en plusieurs moments différents afin de pouvoir naviguer rapidement d'un état à l'autre. Ces multiples pré-réglages sont sauvegardés à même le module (c'est-à-dire le fichier portant l'extension « c5 », où est définit la chaîne de traitement de signal ainsi que l'interface graphique correspondante) lorsque ce dernier est enregistré. On navigue d'un pré-réglage à l'autre en les sélectionnant dans le menu déroulant. Ce système fait en sorte qu'un seul module Cecilia peut devenir un environnement de travail créatif autonome et complet. Un processus sonore peut alors être développé en plusieurs étapes, chacune de ces étapes ayant un rôle spécifique à jouer dans une construction musicale donnée. Tout ce processus est sauvegardé dans un seul fichier texte, entièrement portable d'un système d'exploitation à un autre.

Figure 13. Panneau de gestion des pré-réglages.

3. ÉCRITURE DE MODULES CECILIA

Depuis la version 5 de Cecilia, un module n'est rien d'autre qu'un simple script python, contenant une classe où est décrite la chaîne de traitement de signal et une liste d'éléments d'interface graphique à afficher. En plus de la librairie de modules fournis avec l'application, des modules originaux peuvent être développés avec un simple éditeur de texte et ouverts directement par Cecilia. L'API de Cecilia est accessible à même le logiciel, via le menu *Help → Show API Documentation*. Il contient principalement deux sections. La première concerne la classe audio où est implémenté le processus sonore. La seconde section décrit les différents outils graphiques disponibles pour la construction de l'interface.

3.1. La classe de traitement audio

Un module Cecilia doit absolument contenir une classe nommée « Module », dans laquelle sera développée la chaîne de traitements sonores. Cette classe, pour fonctionner correctement dans l'environnement Cecilia, doit hériter de la classe « BaseModule », définie à même le code source de Cecilia. C'est la classe « BaseModule » qui se charge de créer, à l'interne, les liens entre l'interface graphique, les contrôleurs externes et le processus audio lui-même.

3.1.1. Initialisation

La classe audio d'un module doit impérativement être déclarée comme ceci :

```
class Module(BaseModule):
    def __init__(self):
        BaseModule.__init__(self)
        # traitement sur le signal...
```

Comme le fichier sera exécuté à l'intérieur de l'environnement Cecilia, il n'est pas nécessaire d'importer *pyo* ou la classe « BaseModule », à laquelle on fait référence en tant que classe parente. Ces composantes étant déjà importées par Cecilia, elles seront disponibles à l'exécution du fichier.

3.1.2. Sortie audio du module

Afin d'acheminer le signal audio du processus à l'application, « self.out » doit absolument être le nom de variable de l'objet à la toute fin de la chaîne de traitement. Cecilia récupère cette variable pour faire suivre le signal audio vers la section post-traitement et ultimement, à la sortie audio. Un example typique de déclaration de la variable « self.out » ressemblerait à ceci :

```
self.out = Mix(self.boo, self.nchnls, self.env)
```

Le signal modifié, « self.boo », est remixé en fonction du nombre de canaux spécifié par l'interface de Cecilia (« self.nchnls »), puis pondéré en amplitude par la ligne de graphe « self.env ».

3.1.3. Attributs et méthodes publiques de la classe « BaseModule »

Certaines variables publiques, définies à l'initialisation de la classe « BaseModule », contiennent des informations importantes sur la configuration courante de Cecilia et peuvent être utilisées à tout moment dans la gestion du processus audio. De même, la classe « BaseModule » définit un certain nombre de méthodes utilitaires qui peuvent être utilisées dans la composition de la chaîne de traitements du module.

Principaux attributs publics :

- **self.sr** : Fréquence d'échantillonnage de Cecilia.
- **self.nchnls** : Nombre de canaux audio en sortie.
- **self.totalTime** : Durée totale de la performance.

Principales méthodes publiques :

- **self.addFilein(name)** : Charge un son dans un espace mémoire.
- **self.addSampler(name, pitch, amp)** : Génère un lecteur de son en boucle.
- **self.getSamplerDur(name)** : Retourne la durée, en secondes, du son chargé dans un lecteur.

Le lecteur est invité à consulter la documentation de Cecilia pour obtenir la liste complète des attributs et méthodes disponibles.

3.1.4. Documentation du module

Les informations pertinentes à une bonne compréhension du comportement d'un module peuvent être données dans le « __doc__ string » de la classe « Module ». L'utilisateur de Cecilia peut, à tout moment, consulter la documentation du module via la commande menu *Help* → *Show module info*.

3.2. Déclaration des éléments d'interface

Le fichier Cecilia (portant l'extension « c5 »), en plus de la classe audio, doit spécifier la liste des contrôles graphiques nécessaires au bon fonctionnement du module. Cette liste doit impérativement porter le nom « Interface » et être constituée d'appels de fonctions propres à l'environnement Cecilia. Voici un exemple où est déclaré un lecteur de son, une ligne de graphe, trois potentiomètres et un contrôle de la polyphonie :

```
Interface = [
    csampler(name="snd"),
    cgraph(name="env", label="Overall Amplitude",
           func=[(0,1),(1,1)], col="blue1"),
    cslider(name="drv", label="Drive", min=0, max=1,
            init=0.5, col="purple1"),
    cslider(name="lp", label="LP F", min=20, max=15000,
        rel="log", init=5000, unit="Hz", col="green1"),
    cslider(name="q", label="LP Q", min=0.5, max=5,
            init=1, col="green2"),
    cpoly()
]
```

Ces quelques lignes de code suffisent à produire l'interface graphique suivante :

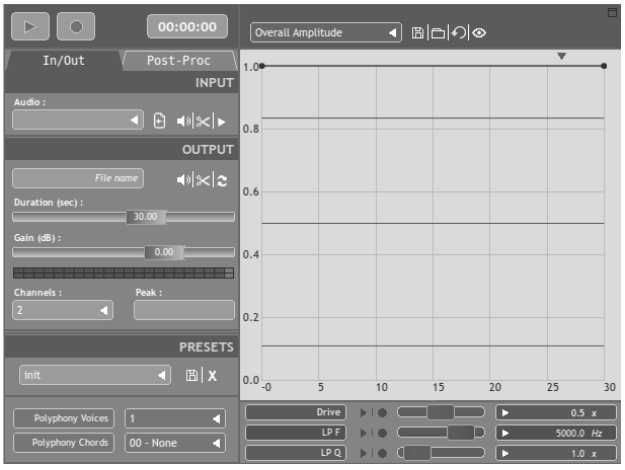


Figure 14. Interface graphique (les outils de manipulation du grapheur sont cachés sur cette figure).

La documentation de Cecilia fournit la liste complète des fonctions graphiques disponibles et de l'élément d'interface qu'elles permettent de créer.

3.3. Exemple

Le code suivant, basé sur l'interface graphique générée à la section précédente, illustre la création d'un module de distorsion du signal agrémenté d'un filtre passe-bas résonant.

```
class Module(BaseModule):
    """
    Module's documentation

    """
    def __init__(self):
        BaseModule.__init__(self)
        self.snd = self.addSampler("snd")
        self.pre = Sig(self.drv, mul=10, add=1)
        self.clp = Clip(self.snd*self.pre, -.99, .99)
        self.flt = Biquad(self.clp, self.lp, self.q)
        self.boo = self.flt * 0.2
        self.out = Mix(self.boo, self.nchnls, self.env)


Interface = [
    csampler(name="snd"),
    cgraph(name="env", label="Overall Amplitude",
           func=[(0,1),(1,1)], col="blue1"),
    cslider(name="drv", label="Drive", min=0, max=1,
            init=0.5, col="purple1"),
    cslider(name="lp", label="LP F", min=20, max=15000,
        rel="log", init=5000, unit="Hz", col="green1"),
    cslider(name="q", label="LP Q", min=0.5, max=5,
            init=1, col="green2"),
    cpoly()
]
```

4. CONCLUSION

Grâce à son large éventail de modules de traitement de signal et à son interface graphique simple et versatile, Cecilia constitue un environnement de travail idéal pour l'exploration et la production sonore. Les nouveaux modes de jeu, permettant d'injecter dans le module le signal en provenance de la carte de son, alliés au contrôle des paramètres via des interfaces externes (MIDI ou OSC), font de Cecilia un environnement de traitement en temps réel aux multiples possibilités. La simplicité de l'API de Cecilia en fait un outil pédagogique motivant pour l'apprentissage de la programmation musicale et devrait favoriser, dans un futur rapproché, la création de modules d'effets audio-numérique originaux.

5. REFERENCES

[1] Bélanger, O., « La programmation audio avec Python », *GNU Linux Magazine France*, éditions diamond, vol. 157, France, 2013.

[2] Cecilia5, « Projet Cecilia5 sur googlecode », *http ://code.google.com/p/cecilia5/*, (2011-2014).

[3] Dunn, R., Rappin, N., « wxPython in action », Manning publications, Greenwich, CT, 2006.

[4] Piché, J., Burton, A., « Cecilia : A Production Interface to Csound », *Computer Music Journal*, MIT Press, 1998, Vol 22, numéro 2, p. 52–55.

[5] Python S. F., « Python programming language – official website », *http ://python.org/*, (1990-2013).

[6] Vercoe, B., Ellis, D., « Real-time csound : Software synthesis with sensing and control », *International computer music conference*, 1990, p. 209–211, Glasgow.

[7] Wright, M., et al., « OpenSound Control : State of the Art 2003 », *Conference on New Interfaces for Musical Expression*, 2003, Montréal.

UNE NOUVELLE APPROCHE DES OBJETS GRAPHIQUES ET INTERFACES UTILISATEURS DANS PURE DATA

Pierre Guillot
CICM - EA1572,
Université
Paris 8, MSH Paris Nord,
Labex Arts H2H
Guillotpierre6@gmail.com

RÉSUMÉ

Cet article présente le CICM Wrapper, une interface de programmation facilitant la création d'objets graphiques et d'interfaces utilisateurs pour le logiciel Pure Data ainsi que la bibliothèque d'objets qui en découle. Nous revenons sur les problèmes de programmation liés aux mises en œuvre d'interfaces graphiques utilisateurs rencontrés dans Pure Data afin de préciser les choix qui ont conduit à la création d'une interface de programmation ainsi que les solutions que nous proposons. Enfin nous présentons en détail les objets de la bibliothèque et leurs nouvelles fonctionnalités pour mettre en évidence les améliorations offertes par notre approche.

1. INTRODUCTION

Depuis 2012, le CICM[1] a développé une bibliothèque de mise en espace du son grâce aux techniques ambisoniques dans le cadre des projets du Labex Arts H2H de l'Université Paris 8[2]. Ces projets, orientés vers une approche artistique, ont pour objectif d'explorer les possibilités musicales offertes par ces techniques et de rendre accessible aux musiciens et compositeurs les concepts acoustiques et mathématiques sous-jacents pour faire émerger de nouvelles applications musicales. La bibliothèque HOA[3], qui en résulte, offre un ensemble de classes C++ desquelles découlent des mises en œuvre pour des logiciels d'édition musicale et de synthèse sonore largement usités par les musiciens, notamment sous forme d'objets pour les logiciels Max[4] et Pure Data[5] [1][2].

L'un des éléments essentiels de ces mises en œuvre a été la création d'interfaces graphiques utilisateurs permettant de répondre aux enjeux didactiques de la bibliothèque et de faciliter la prise en main des opérations dans le domaine des harmoniques circulaires[6]. Grâce à l'objet hoa.scope~, par exemple, il est possible de visualiser graphiquement les harmoniques circulaires et ainsi de faciliter la compréhension du modèle ambisonique (Figure 1). Ces interfaces sont relativement complexes à élaborer de par la grande variété des interactions proposées à laquelle se rajoutent des représentations qu'il n'est aisément possible de réaliser qu'en utilisant des systèmes de calques[7] et des opérations comme des rotations et translations matricielles. En résumé, la création de ces interfaces nécessite que les plateformes logicielles auxquelles elles sont destinées disposent au sein de leurs kits de développement logiciel de nombreux outils dédiés à la mise en œuvre d'interfaces graphiques utilisateurs.

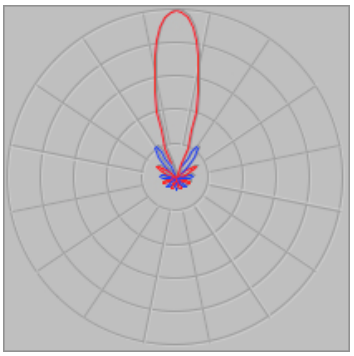


Figure 1. L'objet hoa.scope~ représente le champ sonore sous la forme d'une somme pondérée d'harmoniques circulaire.

Alors que la mise en œuvre des interfaces pour le logiciel Max a été relativement aisée à réaliser, grâce à un SDK[8] très bien adapté aux besoins de la bibliothèque HOA - tant pour la mise en place de traitements pour le multicanal que pour la création d'interfaces graphiques, de nombreuses complications sont apparues lors des mises en œuvre pour le logiciel Pure Data. Ces difficultés nécessitent de repenser la structure et le fonctionnement général des objets graphiques et des objets de traitement du signal dans Pure Data [3]. Notons néanmoins qu'il était déjà possible d'utiliser certains traitements de la bibliothèque HOA dans Pure Data grâce à la version FAUST qui permet de compiler des

[1] Centre de recherche Informatique et Création Musicale.

[2] Les projets sont "*La spatialisation du son pour le musicien, par le musicien*" et "*Des interfaces pour la mise en espace du son*" et "*HOA, 3D*".

[3] http://www.mshparisnord.fr/hoalibrary/.

[4] http://cycling74.com/.

[5] http://msp.ucsd.edu/software.html/.

[6] Les harmoniques circulaires sont des fonctions périodiques utilisées en ambisonie pour représenter l'espace.

[7] Système de calques que nous pouvons trouver dans les logiciels de dessin.

[8] *Software development kit* ou kit de développement logiciel.

traitements pour de multiples plateformes logicielles, cependant cette version est plus adaptée à une utilisation fixe qu'aux expérimentations[9] que nous souhaitons réaliser avec la version HOA spécifique à Pure Data. Afin de répondre à ces contraintes et de faciliter la mise en œuvre des objets graphiques, nous avons réalisé une bibliothèque en C et Tcl/Tk[10] sous la forme d'une API[11] : le CICM Wrapper[12]. Cet ensemble de codes permet de proposer de nouvelles fonctionnalités et de faciliter la mise en œuvre d'objets. Loin d'être restreinte à notre contexte d'interfaces pour la spatialisation ambisonique et à nos problématiques initiales, ce travail s'est ouvert à de nombreuses utilisations et a permis notamment de réaliser un ensemble d'interfaces graphiques dans Pure Data au fort potentiel ergonomique.

Dans cet article, nous revenons, dans un premier temps, sur les problèmes que nous avons rencontrés lors de la mise en œuvre d'objets dans Pure Data et présentons par la suite les solutions que nous proposons via le CICM Wrapper. Il s'agit de permettre aux utilisateurs de comprendre les modifications opérées par notre approche et les changements par rapports aux précédentes interfaces graphiques. Nous espérons ainsi offrir aux développeurs la possibilité de prendre en main cette API, afin qu'ils puissent offrir de nouveaux outils qui nous l'espérons répondront aux besoins de la communauté d'utilisateurs de Pure Data [4].

2. LES DIFFICULTES DE PROGRAMMATION D'OBJETS EXTERNES

Lors de la mise en œuvre des interfaces graphiques utilisateurs dans Pure Data, nous avons rencontré de nombreuses difficultés. Il nous est impossible d'énumérer l'ensemble des fonctionnalités et approches qui selon nous posent problème; de ce fait nous présentons celles qui nous semblent les plus contraignantes.

Tcl et Tk sont dans Pure Data le langage et les outils qui gèrent les interfaces graphiques et les interactions avec l'utilisateur par envoi d'instructions sous forme de scripts. Ainsi, ces scripts doivent être réécrits dans chaque objet pour chaque nouvelle instruction, comme le dessin d'un rectangle ou la récupération de la position du curseur de la souris. Étant un langage interprété, cette pratique est très sujette aux erreurs de programmation car celles-ci ne sont pas révélées lors de la compilation des objets. De plus, la syntaxe expansive peut très rapidement encombrer les codes et rendre difficile leur lecture. Bien qu'il nous paraisse préférable de ne pas se servir de ce langage au sein du code d'un objet, son utilisation semble inévitable, du moins dans une première approche.

L'un des éléments gênants de ce langage dans notre contexte est le système de *tag*. Il est, en effet, possible d'attribuer des noms de référence aux éléments dessinés permettant par la suite de pouvoir les modifier ou les supprimer, opérations nécessaires dans toute mise en œuvre. Ainsi, cette propriété du langage se rapproche des systèmes de calques, que nous avons suggéré précédemment. Cependant, la formulation n'est ici pas des plus adaptées. Il est souvent nécessaire de définir des noms spécifiques à certains éléments, des noms généraux à des groupes d'éléments et un nom global à l'ensemble des éléments d'un même objet ce qui complexifie le code. Notons encore qu'étant donné que ces noms ont un champ d'action pour l'ensemble d'un *canvas*, il faut que les noms contiennent un élément unique relatif à chaque instance d'un objet afin que les changements ne s'appliquent qu'à un objet spécifique et non à l'ensemble des objets d'une même classe qui se trouvent dans ce *canvas*.

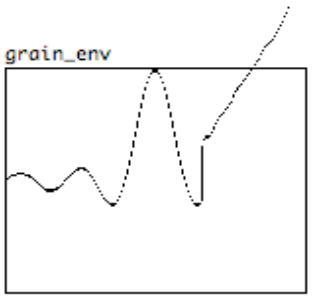


Figure 2. Exemple concret où le dessin peut sortir des limites d'un objet dans Pure Data avec l'objet array.

Une autre contrainte rencontrée réside dans la particularité de Pure Data de devoir dessiner directement dans le *canvas*. Bien que le programmeur définisse des limites à son objet (permettant notamment d'être notifié lorsque l'utilisateur clique ou se déplace dans l'objet), il lui est possible de dessiner en dehors de celles-ci (Figure 2). Ce problème n'est que minime pour des interfaces graphiques simples tels qu'un objet bang ou un objet toggle mais dès lors que l'interface se complexifie, il peut devenir nécessaire de tronquer les figures dessinées dans les limites de l'objet.

Dans l'objet hoa.map, par exemple, il est possible de déplacer des cercles, représentant des sources sonores, au delà des bordures de l'objet parce que l'interface ne représente qu'une partie focalisée de l'espace d'interaction. Dans un tel cas, il est nécessaire dans le code de vérifier si les cercles sont dans les limites ou non de la boîte afin de les dessiner ou de les effacer (Figure 3). Cette opération se complexifie lorsque le cercle chevauche les limites où l'utilisateur est en attente d'un

[9] Le langage FAUST ne permet pas de mettre en œuvre des objets avec un nombre de canaux dynamiques ou des interfaces graphiques, il est ainsi peu adapté à une phase d'expérimentation dans ce contexte de l'ambisonie bien qu'offrant des avantages certains tant sur le plan du déploiement multi-plateforme que sur celui des optimisations.

[10] Tcl (Tool Command Language) est un langage de programmation et Tk est une bibliothèque logicielle pour la création d'interfaces graphiques, http://www.tcl.tk/.

[11] *Application programming interface* ou interface de programmation

[12] Le CICM Wrapper est disponible sur le répertoire Git du CICM, https://github.com/CICM/CicmWrapper.

cercle partiellement caché ; or cette opération est complexe à mettre en œuvre, nécessite de nombreux calculs et amène immanquablement un manque de clarté dans le code.

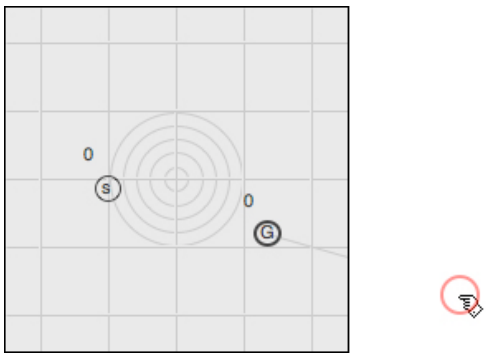


Figure 3. L'objet hoa.map où une source sonore, représentée par le cercle rouge, est déplacée en dehors des limites de l'objet.

Enfin une dernière caractéristique dérangeante, liée au dessin dans Pure Data, est le fait que le programmeur doive lui même dessiner le contour de sa boîte ainsi que les entrées et les sorties alors que les *canvas* définissent à quel endroit l'utilisateur doit cliquer afin de créer des lignes et de raccorder les objets. En somme, il revient au programmeur de redéfinir cette formule, qui répond en grande partie à de fins réajustements, permettant de calculer l'emplacement des entrées et sorties afin de les dessiner aux endroits qu'il convient. Opération redondante qui, là encore, complexifie le code.

Sur le plan des interactions, l'API de Pure Data est encore ici relativement restreignant. Les *canvas* sont rattachés via Tcl/Tk à un certain nombre d'événements desquels ils reçoivent des informations, comme les coordonnées de la souris lors de son déplacement ou les touches du clavier activées, qu'ils renvoient par la suite aux objets lorsque ceux-ci possèdent les méthodes adéquates. L'architecture peut sembler tout à fait valable mais le problème réside dans le manque de variété des événements auxquels un objet peut être rattaché. Ceux-ci sont seulement au nombre de deux pour les événements liés à la souris, son déplacement et le clic de la souris potentiellement associés à deux touches : *Alt* et *Shift*. Dans l'objet hoa.map, le comportement de l'objet diffère non seulement selon la combinaison des touches mais offre, de plus, la possibilité de faire apparaître un menu contextuel lorsque l'utilisateur effectue un clic droit sur l'interface ; or cet événement est exclusivement rattaché au menu contextuel natif de Pure Data (permettant d'accéder à l'aide ou aux propriétés de l'objet) ainsi, il est impossible d'en être notifié et de modifier ce comportement. Aussi, il nous a très rapidement semblé que la variété des événements auxquels nous avons accès dans Pure Data est trop limitée en comparaison de l'ensemble des événements usuellement offerts dans les interfaces utilisateurs [5].

La création d'une fenêtre de propriétés nous a semblé aussi être un problème récurrent. Il est, en effet, nécessaire de réaliser une nouvelle interface en Tcl/Tk pour chaque objet affichant les différentes valeurs des propriétés telles que la taille de l'objet, la police ou encore les couleurs. A cela, il faut ajouter une série de méthodes permettant de recevoir et envoyer ces valeurs afin de communiquer entre l'objet et la fenêtre de propriétés. De plus, les fenêtres de propriétés ne diffèrent d'un objet à un autre que par leur nombre, mais les types de variables (données numériques ou données textuelles) et leurs représentation (zone de texte, sélecteur de couleur, boîte nombre, etc.) se retrouvent d'un objet à un autre. Ainsi, cette opération est, encore ici, redondante et ne fait que diminuer la clarté du code. Il nous aurait donc semblé préférable de pouvoir générer automatiquement cette fenêtre de propriété, sans avoir nécessairement à réécrire l'ensemble du script, et d'accéder et modifier ces valeurs exclusivement à partir du code source C de l'objet. Notons encore que ces propriétés sont, dans les usages, des variables que l'utilisateur souhaite sauvegarder et retrouver à la même valeur à l'ouverture d'un patch. Là encore, il revient au programmeur de réaliser la sauvegarde des propriétés à l'enregistrement du patch et leur récupération et initialisation à l'ouverture. En somme, le programmeur peut, ici aussi, souhaiter faire appel à des fonctions usuelles, tel que proposées dans le système d'attributs présent dans le SDK du logiciel Max.

De manière plus générale, un ensemble de fonctions ou d'outils usuels, pouvant grandement faciliter la création d'objets dans Pure Data, nous semblait manquer [4]. Le présent contexte ne permet pas, comme nous l'avons dit, de tous les énumérer mais nous nous intéressons plus particulièrement à deux d'entre eux. D'une part la gestion des fonctions du traitement du signal qui devient complexe à mettre en œuvre dès lors que l'on souhaite un nombre d'entrées et de sorties dynamique ou lorsque les vecteurs d'entrées et sorties doivent être différents (afin de réaliser des traitements du signal *out-of-place*) et d'autre part, un système de type *proxy* pour les entrées permettant, par exemple, d'avoir plusieurs entrées pouvant recevoir tous types de données mais offrant la possibilité d'obtenir l'index de l'entrée par laquelle chaque message arrive.

3. RESOLUTION DES PROBLEMES

Notre travail a été orienté par certaines nécessités. Cette bibliothèque se doit de fonctionner sur l'ensemble des systèmes d'exploitations auxquels est destiné Pure Data notamment Linux, Mac OS et Windows, c'est pourquoi notre choix s'est très rapidement porté sur l'utilisation seule des dépendances internes à Pure Data. Ainsi, il nous a paru préférable, malgré les problèmes rencontrés, de proposer une version utilisant uniquement Tcl/Tk. Il s'agissait aussi de ne pas limiter les possibilités de programmation donc de laisser ouverte l'utilisation des fonctions natives et aussi de pouvoir être

facilement intégré à Pure Data ou Pure Data Extended. Enfin, nous avons tenté de nous rapprocher au maximum de l'API du logiciel Max afin de faciliter par la suite le passage d'un objet Max au logiciel Pure Data et inversement.

La première approche employée utilisait la structure d'objet graphique aujourd'hui généralement répandue et qui se trouve dans les objets graphiques natifs de Pure Data et de tenter par la suite de contourner et résoudre les problèmes. Cependant, cette approche est une impasse, il est, en effet, impossible d'être notifié de toutes les interactions réalisées sur l'objet et rogner les dessins dans les objets [6] reste très compliqué à réaliser. De plus, cette approche engendre le développement d'un code source très complexe, rempli d'exceptions impliquant de nombreux problèmes, notamment sur la potentielle réutilisation et relecture du code.

La solution retenue est apparue en étudiant les codes sources des objets tels que entry ou popup[13]. Bien que leurs mises en œuvre ne semblent pas optimales au premier abord - de nombreuses difficultés compliquent l'utilisation de ces objets - l'approche qui consiste à créer des *widgets* via Tcl/Tk dans le patch, comme un menu contextuel ou champ de saisie textuel, nous a semblé pouvoir offrir une solution à nos problèmes notamment par le fait qu'il est possible d'être rattaché à l'ensemble des événements utilisateurs sans passer par le *canvas*. Notre solution consiste donc à créer un *widget* de type fenêtre dans lequel nous créons une nouvelle instance d'un *canvas*, cet outil peut alors être vu comme une fenêtre au contour invisible possédant son propre *canvas* et incrusté dans le *canvas* initial. Ce procédé offre deux principaux avantages, d'une part, il est possible de rattacher le *canvas* créé à l'ensemble des événements proposés dans Tcl/Tk afin d'en être notifié et d'autre part, de ne plus se soucier de dessiner hors des limites étant donné que la fenêtre dans laquelle nous nous plaçons tronque automatiquement le dessin à l'intérieur de ses limites. Néanmoins cette approche implique que le *canvas* initial ne soit plus notifié des événements de utilisateur lorsque le focus est sur notre objet. Ainsi cela amène l'inconvénient de devoir notifier en retour le *canvas* initial d'un grand nombre d'événements comme la sélection de notre objet, son déplacement, le raccordement de ses entrées ou sorties, méthodes que nous avons dû implémenter.

Afin de généraliser cette approche et de pouvoir accéder facilement à ce système dans de multiples mises en œuvre, nous avons fait le choix de développer une API (Figure 4). Cette interface de programmation comprend ainsi un certain nombre de structures et de fonctions sur lesquelles nous revenons brièvement afin d'en comprendre non pas le fonctionnement, car le présent document ne pourrait suffire, mais du moins son utilisation.

[13]L'objet *entry* et *popup* font parti de la bibliothèque *flatgui* développé par Ben Bogart pour Pure Data Extented.

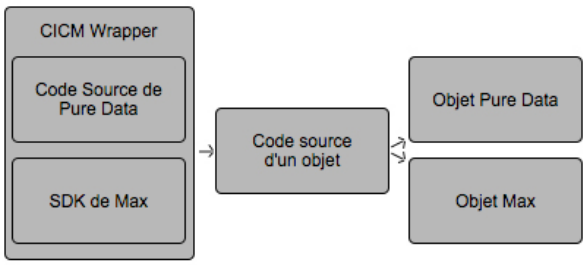


Figure 4. Représentation du CICM Wrapper dans le processus de création d'un objet Pure Data ou Max.

Il a fallu étendre la structure de *classe* d'objet native de Pure Data, structure définissant un type d'objet, afin d'offrir les outils nécessaires à nos attentes. A l'instanciation d'une nouvelle classe d'objet, qui se réalise de manière très similaire à ce que nous pouvons trouver dans l'API de Max, nous définissons d'une part si l'objet est de type graphique et/ou traitement du signal afin d'initialiser la structure des objets selon leurs fonctionnalité et d'autre part les méthodes de l'objet, nous permettant alors d'initialiser un certains nombre de fonctions enveloppant le fonctionnement interne de l'objet pour ne laisser transparaître au programmeur que ce qui nous semble nécessaire.

```
CLASS_ATTR_RGBA                 (c, "bgcolor", 0, t_bang, f_color_background);
CLASS_ATTR_LABEL                (c, "bgcolor", 0, "Background Color");
CLASS_ATTR_ORDER                (c, "bgcolor", 0, "1");
CLASS_ATTR_DEFAULT_SAVE_PAINT   (c, "bgcolor", 0, "0.75 0.75 0.75 1.");
CLASS_ATTR_STYLE                (c, "bgcolor", 0, "color");
```

Figure 5 Portion du code de l'objet c.bang présentant la syntaxe du CICM Wrapper, similaire à celle utilisée dans MAX, pour créer des attributs.

Il est aussi possible au programmeur de créer des attributs pour cette classe qui seront dès lors sauvegardés et initialisés de façon automatique à la création et à la sauvegarde de chaque instance de l'objet. La création de ces attributs se fait via un système de macro-définitions[14], ici aussi très similaires à celles proposées dans l'API de Max permettant de définir un style aux attributs (texte, nombre, menu, couleur, etc.), des valeurs par défaut, une étiquette de présentation, et d'autres caractéristiques qui apparaîtront dans la fenêtre de propriété (Figure 5).

Le fonctionnement interne de l'objet est, quant à lui, relativement opaque au programmeur mais les méthodes à mettre en place sont sensiblement plus simples à réaliser. Les deux points les plus problématiques dans la création d'un objet, le dessin et les interactions, sont nettement plus simples à mettre en place. Le dessin dans un objet se fait en créant une méthode répondant au message *paint*. Dans cette méthode, il est possible de créer des calques répondant à un nom et dans ces calques il est possible de dessiner toutes formes de figures. Afin de créer des dessins, nous offrons des fonctions qui

[14]Système permettant de remplacer un indicateur, sous forme de texte, par un texte.

s'occupent de mettre en forme et d'envoyer le script Tcl/Tk et qui gèrent notamment le système de *tag* évitant ainsi toute erreur de syntaxe ; notons néanmoins qu'il est toujours possible de passer outre ce système si besoin[15]. En plus des fonctions de dessin usuelles telles que la création de forme géométriques simples ou l'écriture de texte, nous offrons un système de matrices permettant de réaliser des rotations et des translations. Il est aussi possible de définir les couleurs à utiliser dans les formats RGB, HSL ou en hexadécimal, offrant plus de flexibilité. Cette méthode de dessin est appelée soit de manière opaque à chaque fois qu'il est nécessaire de redessiner l'objet, lorsque l'objet bouge ou qu'un attribut lié au graphisme est changé par exemple, soit de manière transparente dans le code de l'objet. Il est ainsi possible au programmeur, selon le contexte, d'invalider un calque spécifique ; seulement ce calque-là sera alors redessiné à l'appel de la méthode *paint* afin d'alléger le processus (Figure 6). Enfin le dessin des bordures, des entrées et sorties est réalisé de façon automatique.

```
// Fonction qui dessine l'arrière plan de l'objet
void draw_background(t_bang *x, t_object *view, t_rect *rect)
{
    // Le rayon du cercle inscrit de l'objet
    float radius = rect->width * 0.5;

    // Creation d'un nouveau calque défini par le symbol
       "background_layer"
    t_elayer *g = ebox_start_layer((t_ebox *)x,
       gensym("background_layer"), rect->width, rect->height);
    // t_jgraphics *g = jbox_start_layer((t_object *)x, view,
       gensym("background_layer"), rect->width, rect->height);

    // Si le calque est nouveau ou a été invalidé, il est possible de
       le redéfinir
    if (g)
    {
        // Definition de la couleur utilisée
        egraphics_set_color_rgba(g, &x->f_color_background);
        // jgraphics_set_source_jrgba(g, &x->f_color_background);

        // Création d'un cercle
        egraphics_arc(g, radius, radius, radius * 0.9, 0, EPD_2PI);
        // jgraphics_arc(g, radius, radius, radius * 0.9, 0, EPD_2PI);

        // Ajout du cercle dans le calque en remplissant
        egraphics_fill(g);
        // jgraphics_fill(g);

        ebox_end_layer((t_ebox*)x, gensym("background_layer"));
        // jbox_end_layer((t_jbox*)x, view,
           gensym("background_layer"));
    }

    // Dessin du calque
    ebox_paint_layer((t_ebox *)x, gensym("background_layer"), 0., 0.);
    // jbox_paint_layer((t_jbox *)x, view,
       gensym("background_layer"), 0., 0.);
}

// Fonction appelée au click de la souris
void mouse_down(t_bang *x, t_object *view, t_pt pt, long modifiers)
{
    // Invalidation du calque de l'arrière plan de l'objet
    ebox_invalidate_layer((t_ebox *)x, gensym("background_layer"));
    //jbox_invalidate_layer((t_object *)x, NULL,
       gensym("background_layer"));

    // Notification à l'objet de redessiner
    ebox_redraw((t_ebox *)x);
    // jbox_redraw((t_jbox *)x);

    outlet_bang(x->f_out);
}
```

Figure 6. Portion du code de l'objet c.bang présentant la syntaxe utilisée dans le CICM Wrapper pour dessiner et gérer le système de calques. Les fonctions commentées utilisent la syntaxe de Max qu'il est aussi possible d'utiliser.

[15]Nous invitons le lecteur à regarder le code de l'objet c.blackboard afin d'en avoir un exemple.

Les interactions de l'utilisateur sont à présent très variées et font appel à plusieurs méthodes selon les actions : entrer dans l'objet, sortir de l'objet, survoler l'objet, cliquer, « traîner », relâcher, utiliser la molette et double cliquer, presser une touche clavier quelconque, presser une touche clavier « spécifique » (Tab, Supprimer, Espace, etc.). Ces méthodes relatives aux événements de la souris transmettent la position de la souris relative à l'objet mais aussi les clefs Shift, Maj, Ctrl et Atl. En somme, il est à présent possible de réaliser un objet au fonctionnement élaboré et aux dessins complexes sans pour autant rencontrer de grandes difficultés de programmation.

Notons encore que les objets offrent à présent un système de *proxy* pour les entrées très facile à mettre en œuvre ainsi qu'un nouveau formatage des fonctions *DSP* et des fonctions *perfom* similaires à celles offertes dans l'API de Max. A cela se rajoutent de nombreuses petites améliorations dont, entre autres, des méthodes facilitant la lecture et l'écriture de fichiers externes, la mise en place d'un système de préréglages, une fonction renvoyant la position globale de la souris ou relative au *canvas* ou encore un système de notifications[16].

4. LA MISE EN ŒUVRE : PRESENTATION DES OBJETS

Notre bibliothèque, le CICM Wrapper a, par la suite, permis de réaliser un certain nombre d'interfaces graphiques fonctionnant tout aussi bien sur Pure Data Vanilla que sur Pure Data Extended pour les systèmes d'exploitation Mac Os, Linux et Windows et ayant pour but d'améliorer l'ergonomie générale en proposant de nouvelles interactions et en offrant des représentations plus raffinées[17]. Cette série d'objets reprend les outils standards déjà présents soit dans Pure Data, Vanilla et Extented, soit dans Max. Avant d'énumérer l'ensemble des fonctionnalités offertes, nous pouvons mettre en évidence certaines caractéristiques générales à l'ensemble des ces outils.

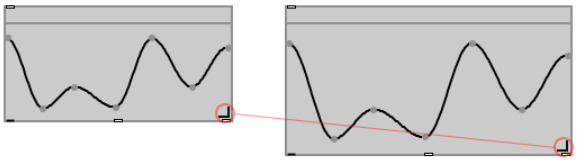


Figure 7. Redimensionnement de l'objet c.breakpoints en déplaçant le coin en bas à droite.

Les objets graphiques possèdent tous la particularité de pouvoir être redimensionnés en tirant les bordures bas ou droite ou le coin bas-droite (Figure 7). Notons que selon les objets, la hauteur et la largeur peuvent être

[16]Fonctions auxquelles le programmeur pourra se référer dans la documentation et les exemples disponibles avec la distribution du projet.

[17]La bibliothèque est disponible sur le répertoire Git du projet CICM Wrapper, https://github.com/CICM/CicmWrapper/releases.

fixées arbitrairement ou dépendre de la taille de la police tel que définie dans l'objet c.number. Afin d'offrir des patchs plus lisibles, les entrées et sorties des objets ne sont visibles qu'en mode édition, nous avons aussi fait le choix de leurs attribuer les couleurs syntaxiques par défaut de Pure Data Extented permettant de facilement repérer le type de messages qu'ils peuvent recevoir ou envoyer.

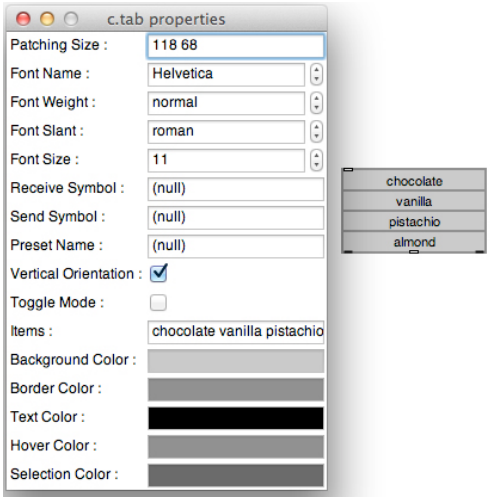


Figure 8. Fenêtre de la fenêtre de propriété de l'objet c.tab.

Tous les objets possèdent des propriétés communes (ou attributs) : la taille de l'objet (hauteur et largeur), le nom, le poids, l'inclinaison et la taille de la police utilisée par l'objet, la couleur d'arrière plan et la couleur de la bordure, auxquelles viennent se rajouter les propriétés spécifiques de chaque objet. Enfin, ils possèdent aussi un symbole d'envoi et un symbole de réception, de manière analogue aux interfaces natives de Pure Data (Figure 8). Ces propriétés peuvent, bien sûr, être modifiées de manière dynamique par des messages envoyés à l'objet et sont aussi accessibles via la fenêtre de propriétés qui facilite leur édition en offrant un champ de saisie textuel ou numérique, avec un système permettant d'incrémenter ou décrémenter des valeurs numériques ou l'index d'un menu, un bouton de type interrupteur ou encore un sélecteur de couleur selon le type de valeurs attendues. Notons enfin que l'ensemble des caractéristiques des propriétés des objets peut être affiché dans la console en envoyant le message "attrprint" à l'objet, permettant de connaître les messages attendus par l'objet pour définir ses propriétés.

Figure 9. Représentation de l'interaction permettant d'incrémenter la valeur de l'objet c.number.

Certains objets sont donc très similaires à leurs homologues natifs de Pure Data. C'est le cas des objets c.bang et c.toggle, qui hormis les améliorations générales à l'ensemble des objets, offrent des fonctionnements identiques au bang et toggle. D'autres proposent de très légères améliorations comme les objets c.slider qui deviennent verticaux ou horizontaux selon que la hauteur soit plus grande que la largeur, l'objet c.number qui permet d'incrémenter des valeurs numériques avec une précision qui dépend de la position de la souris lorsque l'utilisateur clique sur l'objet (Figure 9), l'objet c.number~ qui consiste en une combinaison de l'objet boite nombre et d'un snapshot~, l'objet c.meter~ qui, de manière identique, peut être comparé à une combinaison de l'objet vu et de l'objet pvu~ ou encore l'objet c.radio qui offre en plus, par rapport à l'objet radio, un mode de type *check-list*. D'autres interfaces sont des adaptations de certaines bibliothèques présentes dans Pure Data Extended mais qui, suite à des mises à jours ou du fait de la complexité de leur création, sont difficiles à utiliser, possèdent certains dysfonctionnements ou n'offrent pas certaines fonctionnalités essentielles. C'est le cas des objets c.scope~, un oscilloscope uni ou bidimensionnel, l'objet c.colorpanel, un sélecteur de couleur sous forme de matrice à taille variable, l'objet c.blackbloard, un tableau permettant de dessiner à la souris et par script, d'écrire du texte et d'afficher des images au format GIF, l'objet c.knob, un bouton rotatif avec un mode circulaire et un mode sans fin ou encore l'objet c.menu, qui permet d'afficher un menu déroulant. Enfin certains objets possèdent seulement leurs équivalents dans le logiciel Max et nous semblent des plus pratiques : c.function qui permet de créer une fonction par suite de points avec différents modes d'interpolations, c.incdec qui permet d'incrémenter ou décrémenter une boite nombre sans *stack overflow*, l'objet c.plane qui permet de déplacer un point sur un plan bidimensionnel, l'objet c.tab qui offre une séries de boutons à texte, l'objet c.gain~ qui permet de contrôler le volume sonore avec une glissière à curseur, l'objet c.rslider qui permet de définir une plage grâce à une glissière à double curseur et enfin l'objet c.spectrum~ qui permet d'afficher le spectre d'un signal sonore. Notons néanmoins que l'ensemble de ces objets a été réalisé afin de répondre à des attentes personnelles, ainsi leur fonctionnement peut légèrement varier vis à vis de leurs homologues natifs de Pure Data ou de Max.

Enfin, nous offrons un objet c.preset. L'ensemble des objets possédant une méthode *preset*, possède un attribut permettant de leurs attribuer un nom qui sert de référence à l'objet afin d'enregistrer l'état actuel de l'objet et revenir à cet état. Ce système est, en somme, très similaire à celui proposé dans le logiciel Max à la différence notable qu'il est possible de réaliser des interpolations. Ce système vise, en outre, à être complété par une série d'interfaces permettant d'éditer les préréglages de manière plus simple tel que le système de *pattrstorage* de Max mais, ici aussi, ayant accès à l'ensemble des méthodes et fonctionnalités des objets, nous pouvons envisager un système différent répondant plus à nos attentes.

5. CONCLUSION

Lors de la mise en œuvre de la bibliothèque HOA pour Pure data, nous avons réalisé une API afin de répondre aux nombreux problèmes rencontrés concernant la création d'objet graphiques et de traitements multicanal. Cette bibliothèque peut néanmoins être utilisée dans un large contexte et ouvre de nouvelles perspectives quant à l'ergonomie du logiciel. Elle possède de plus l'avantage, d'être libre, gratuite et de n'avoir aucune dépendance autre que Pure Data. Cela permet, en outre, d'être totalement indépendant des différentes distributions du logiciel.

Ainsi, nous pouvons envisager que son utilisation perdure avec les évolutions du logiciel et soit poursuivie au sein de la communauté. Nous espérons également voir émerger de nouvelles mises en œuvre conçues en dehors notre équipe de recherche, car suite à nos premières publications des objets, nous avons réalisé qu'il existe une forte demande d'outils stables et ergonomiques pour Pure Data.

Nous pouvons aussi envisager de nombreuses améliorations afin de faciliter la mise en œuvre d'objets. Nous pensons notamment mettre en place un système permettant de s'attacher aux notifications d'autres objets et une méthode de commentaires sur les entrées et sorties. Nous envisageons aussi la possibilité de choisir d'autres API graphiques selon les besoins. Nous pensons notamment à Juce[18] qui permettrait d'avoir un rendu plus homogène entre les systèmes d'exploitation et d'avoir accès à des méthodes tels que le *Drag & Drop* de fichiers ou le rendu tridimensionnel via OpenGl.

Enfin, suite aux nombreux retours qui nous parvenus des utilisateurs et programmeurs, nous envisageons avec les créateurs et les responsables la possibilité d'intégrer la bibliothèque à la distribution de Pure Data Extended et de PD-L2ork, dans laquelle nous pourrions tirer partie de l'utilisation des SVG.

6. REFERENCES

[1] Colafrancesco, J., Guillot P., Paris E., Sedes A., Bonardi 1. « La bibliothèque HOA, bilan et perspectives », *Actes des Journées d'informatique musicale*, St-Denis, France, p. 188-197, 2013.

[2] Guillot P., Paris E., Deneu M. « La bibliothèque de spatialisation HOA pour MaxMSP, Pure Data, VST, Faust, … », *Revue Francophone d'*Informatique *Musicale*, St-Denis, France, 2013.

[3] Zmölnig J. M. « How to Write an External for Pure Data », Institute of Electronic Music and Acoustics, Graz, Autriche, http://pdstatic.iem.at/externals-HOWTO/.

[4] Miller P. « Pure Data : Another Integrated Computer Music Environment », *Proceedings Second Intercollege Computer Music Concerts*, Tachikawa, Japon, 1996.

[5] Todoroff T. « Control of Digital Audio Effects », *DAFX – Digital Audio Effects*, J. Wiley & Son, 2002.

[6] Foley, J. D., Paris E., Van Dam A., Feiner S. K., Hughes J ; F. « Computer Graphics : Principles and Practice in C, 2nd Edition ». *Addison-Wesley*, 1995.

[18] http://www.juce.com/.

OSSIA : OPEN SCENARIO SYSTEM FOR INTERACTIVE APPLICATIONS

Théo De La Hogue — GMEA — theod@gmea.net

Pascal Baltazar — L'Arboretum — pascal@baltazars.org

Myriam Desainte Catherine — myriam.desainte-catherine@labri.fr

Jaime Chao — LaBRI — jaimitochao@gmail.com

Clément Bossut — bossut.clement@gmail.com

RÉSUMÉ

Cet article fait état des travaux réalisés dans le cadre du projet Open Scenario System for Interactive Application (OSSIA) dont l'objectif est d'offrir des outils génériques pour le développement de logiciels d'écriture de scénario interactif. Après un rappel des acquis sur lesquels le projet s'appuie, nous présenterons les méthodes adoptées pour adapter le formalisme du moteur d'écriture logico-temporel à des situations réelles de production et décrirons un exemple d'implémentation à travers une nouvelle version du logiciel *i-score*.

DES ENJEUX TRANSDISCIPLINAIRES

Qu'il s'agisse de dispositifs interactifs, de jeux vidéo, du spectacle vivant, de l'industrie culturelle ou de la muséographie, la mise en jeu de différents médias requiert l'écriture d'un scénario régissant l'ensemble des interactions. Cette écriture spécifique définissant les comportements de contenus numériques au sein d'un dispositif informatique, en fonction d'un contexte matériel ou numérique, reste cependant une opération complexe et réservée aux experts : la mise en œuvre au sein d'un environnement de programmation intermédia d'une intention scénaristique, même simple, nécessite en effet de prendre en compte l'ensemble des possibles du dispositif et de son environnement. Cette scénarisation dont le déroulement est ouvert, relève de la programmation et est très peu assistée. De plus, la moindre modification peut être source d'aberration et induire des dysfonctionnements de l'ensemble du système.

Pourtant, la description de tels systèmes par leurs concepteurs ou leurs utilisateurs non programmeurs ne reflète pas cette complexité. Elle semble hiérarchisée intuitivement en unités d'actions parfois séquencées, parfois conditionnées au sein d'un schéma mêlant les entrées et sorties du système afin de tenir compte du contexte ou de le modifier. Le projet Open Scenario System for Interactive Application (OSSIA) vise ainsi à favoriser l'écriture de tels scénarios en proposant un ensemble d'outils logiciels autorisant l'édition et l'exécution d'une description schématisée d'un système automatisé.

Reçu dans le cadre de l'appel à projet ANR 2012 Contenu et Interaction, le projet OSSIA[1] réuni un consortium d'acteurs issus de la recherche institutionnelle ou privée, de l'industrie ou œuvrant dans le domaine de la création : le Laboratoire Bordelais de Recherche en Informatique (LaBRI) formalise et développe le moteur logico-temporel, le GMEA – Centre National de Création Musicale d'Albi-Tarn coordonne le projet et structure le partage des outils, le laboratoire Cédric et l'École Nationale du Jeu et des Médias Interactifs (ENJMIN) du CNAM et les sociétés Blue Yeti et RSF intègrent respectivement les outils d'écriture pour la réalisation de scénario interactifs dans les jeux vidéo, dans des dispositifs muséographiques et dans du matériel de pilotage embarqué. OSSIA s'inscrit par ailleurs dans un réseau de nombreux collaborateurs liés à la création ou encore à la formation tels que l'Institut Supérieur des Techniques du Spectacle (ISTS) et l'Ecole Nationale Supérieure d'Arts et Techniques du Théâtre (ENSATT).

CONTEXTE DE RECHERCHE

Cette dynamique de recherche s'inscrit dans des réflexions sur les usages dont certaines conclusions, qu'il nous semble important de rappeler dans cet article, ont esquissé des spécifications pour le projet OSSIA.

2.1. Historique

C'est à l'initiative de la Compagnie Incidents Mémorable/didascalie.net que l'Association Française d'Informatique Musicale (AFIM) a constitué en 2006 un groupe de travail pour mener une étude [1], visant à mettre en œuvre un questionnement des pratiques, des outils et des métiers du sonore dans le contexte du spectacle vivant (théâtre, danse, concert étendu ou multimédia) pour en dégager les particularités. L'objectif était d'établir un état de l'art des environnements logiciels existants pour la composition et l'interprétation du sonore dans le cadre du spectacle vivant et de dégager les prospectives de développement futur de tels environnements. La conclusion qu'une mutualisation de la recherche vers le développement d'interfaces numériques de création multimédia était

[1] Référence ANR-12-CORD-0024

nécessaire a impulsé la création de la plateforme Virage en 2008.

Missionné par l'Agence Nationale de la Recherche, la Plateforme Virage[2] avait pour objectifs de définir un cahier des charges et des spécifications générales pour des développements futurs de nouvelles interfaces de contrôle et d'écriture de contenus multimédia temps-réel (son, image, lumière, machineries, ...) pour la création artistique et les industries culturelles. Suite à une étude approfondie des usages, pratiques et besoins des professionnels de ces domaines [2], le projet a débouché sur de nombreux prototypes (dont le séquenceur Virage) et sur le constat qu'une structure pour partager les développements informatiques permettrait une dynamique de travail en réseau :

> *« La conclusion principale de cette étude des usages a confirmé la nécessité de travailler autour des notions d'interopérabilité et de modularité, en visant la combinaison de nos développements avec les environnements logiciels et matériels existants et correspondant aux habitudes des praticiens, plutôt que l'intégration de cette réalité complexe et multiple dans un illusoire logiciel unique et certainement trop polyvalent pour être viable. »* [3]

2.2. La réalisation de dispositif intermédia

Afin de bien cerner les enjeux d'une recherche au sujet des outils d'écriture, voici un état des lieux des outils actuels propres à la réalisation de dispositifs constitués de plusieurs médias interconnectés dont il faut assurer le contrôle et/ou en écrire le comportement.

Les différents éléments à notre disposition sont :

- des *environnements de création* : Live, VDMX, DLigth, Reaktor, Modul8, MadMapper, adobe Flash, Unity3D, *etc.*,
- des *environnements de programmation* : Max, Pure Data, Isadora, Arduino, Processing, *etc.*,
- des *interfaces de contrôle et de captation* : surface tactile, Kinect, capteur posturologique/physiologique, caméra, micro, BCF2000, MPD24, Monome, *etc.*,
- et du *matériel de diffusion contrôlable ou non* : projecteur de lumière, haut-parleur, moteur, vidéo-projeteur, *etc*.

[2] www.virage-platform.org

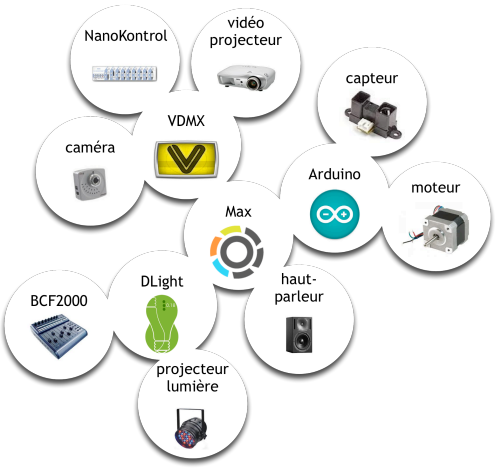


Figure 1. Exemple de dispositif intermédia

2.3. Les moteurs d'écritures

Si les environnements informatiques offrent de réel potentiels d'interconnections de médias, l'écriture du comportement dans le temps de ces éléments interconnectés ramène souvent le réalisateur intermédia à un travail de programmation préalable ou conjoint au travail de création.

Parfois les moteurs d'écriture sont propre à l'environnement de création (Live, VDMX, DLight) tandis que d'autres solutions assurent un contrôle externe (Qlab, Duration, Vezér, i-score, Yannix). Mais certains projets nécessitent le développement d'un moteur d'écriture spécifique, adapté à des exigences esthétiques, à des contraintes techniques ou à des utilisateurs non-experts. Dès lors, les compétences requises pour architecturer correctement un moteur d'écriture se rapprochent de l'ingénierie logicielle (gestion de base de données pour adresser les ressources, mémoriser et rappeler leurs états, programmation d'automate, programmation concurrente, *etc.*).

Ainsi ces constats ont motivé la conception d'une librairie logicielle au service de développements spécifiques de moteurs d'écritures de scénarios interactifs.

METHODOLOGIE

La conception d'éléments logiciels génériques et réutilisables de manière spécifique implique des architectures modulaires et interopérables. Dans cet objectif, le projet OSSIA adopte une démarche de partage des outils et les expérimente en situation de production.

3.1. La structuration du partage

OSSIA s'appuie sur une organisation de développement informatique collaboratif international et open

source (Jamoma[3]) pour constituer un base commune d'éléments logiciels[4]. Les produits de la recherche sont hétéroclites et regroupent une API[5] en cours d'écriture, la librairie C++ Score, dont le formalisme est décrit plus bas, et divers *externals* ou *patchs* Max issus de préoccupations techniques diverses mais intégrables dans différents domaines d'application spécifiques. À noter que la problématique de l'interopérabilité, dont il n'est pas question dans cet article, représente une part non-négligeable des éléments partagés au sein du projet. Pour chaque chantier, nous nous assurons en premier lieu qu'il est possible aux moteurs d'écritures de dialoguer avec les logiciels ou environnements de travail propres à chaque domaine d'application.[6]

3.2. Les chantiers d'expérimentation

Lors des chantiers d'expérimentations en situation réelle de production, des retours concrets sur l'utilisation des notions et outils issus de la recherche viennent alimenter les formalismes sous-jacents.

Durant ces échanges autour de la question des scénarios interactifs la communication entre experts est essentielle mais les discussions peuvent se heurter au fait qu'ils ne disposent pas d'un langage unique comme moyen d'échange [4]. Ainsi une médiation est nécessaire entre les différents acteurs afin de vérifier la robustesse des modèles établis par la recherche face à des cas réels d'utilisation et c'est pourquoi le projet OSSIA s'attache à proposer un formalisme d'écriture de scénarios interactifs.

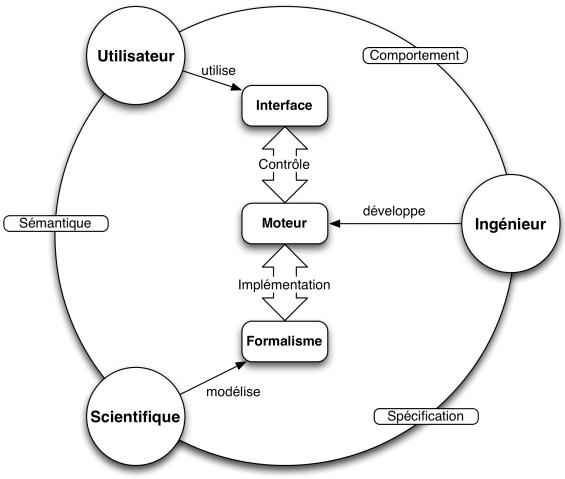


Figure 2. Relations entre les différents acteurs à travers les chantiers d'expérimentations

[3] www.jamoma.org propose des solutions fondées sur les besoins issus des pratiques de la création artistique.

[4] github.com/OSSIA

[5] github.com/OSSIA/API

[6] À titre d'exemple les environnements Live, Modul8 et Unity3D font l'objet d'un effort particulier pour être rendu interopérable en vue d'une utilisation lors des chantiers d'expérimentation.

VERS UN FORMALISME D'ECRITURE

Les travaux menés dans le cadre du projet OSSIA définissent par « scénario interactif » une structure contenant potentiellement du temps souple, conditionnel, non-linéaire et hiérarchique. Ces notions renvoient aux possibilités de contrôler la vitesse d'exécution, d'attendre ou d'ignorer le déclenchement de certaines parties, d'en répéter l'exécution ou d'autoriser des choix entre plusieurs d'entre elles. La hiérarchie renvoie à des besoins d'organisation en sous-partie d'une structure temporelle.

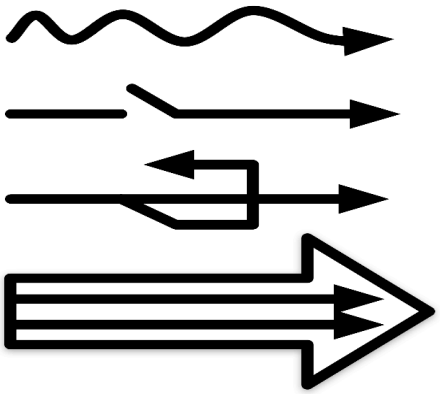


Figure 3. Temps Souple, Conditionnel, Non-Linéaire, Hiérarchique

4.1. Modélisation du temps

La formalisation informatique de ces propriétés temporelles s'appuie sur le modèle existant de partitions interactives [5] et consiste à l'étendre aux conditions, à permettre les répétitions [6] et à autoriser une structuration hiérarchique (propriété modélisée récemment au moyen du langage RML [7]).

Parallèlement, la conception de la librairie *Score* a pris le parti de revisiter les accès au formalisme proposé dans *libIscore* [8][9] à la lumière des retours de l'expérimentation du séquenceur Virage [10]. Le nouveau formalisme s'appose en quelque sorte au-dessus de l'ancien formalisme pour le réorganiser à travers des éléments modulaires plus intuitifs à manipuler ; *Score* considérant par exemple les processus temporels comme encadrés par des évènements eux-mêmes partageables entre processus (l'évènement de fin d'un processus pouvant être l'évènement de début d'un autre processus par exemple) là où *libIscore* ne considérait que des durées possédant chacune leur début et leur fin.

4.2. Lexique

En premier lieu il est nécessaire d'établir un lexique pour décrire les éléments constitutifs d'un scénario interactif :

- *Service* : ce qu'offre une ressource pour son utilisation. Nous distinguons :
  - le *paramètre* dont l'état est susceptible d'être mémorisé (ex : le volume d'une piste audio),
  - le *message* dont l'état ne peut être mémorisé (ex : le déclenchement de la lecture),
  - le *retour* dont l'état ne peut pas être contrôlé (ex : l'enveloppe de l'amplitude ou la fin de lecture).
- *État* : la valeur d'un ou plusieurs services.
- *Namespace* : l'ensemble des noms des services organisés sous la forme d'une arborescence. Par exemple :
  /player
  /play
  /volume
  /end
- *Actions* : une opération faites sur l'état
  - *set* : mise à jour de l'état.
  - *get* : demande de l'état.
  - *listen* : écoute de l'état.
- *Condition* : vérification effectuée sur l'état d'un service. Par exemple on peut vérifier l'expression « /player/volume > 12 ».
- *Événement* : un point dans le temps déclenché à une date précise ou à la validation d'une condition. Des actions peuvent lui être associées.
- *Processus* : une opération effectuée pendant un temps qui peut être fixe, borné ou libre selon la nature de l'évènement qui marque sa fin :
  - *Intervalle* : une quantité de temps qui peut être fixe, bornée ou libre.
  - *Automation* : met à jour l'état d'un service dans le temps selon une courbe.
  - *Mapping* : écoute de l'état d'un service pour mettre à jour l'état d'un autre service.
  - *Scénario* : contient et supervise des évènements, des conditions et des processus.
  - *autres* : d'autres types de processus sont envisageables (ex : générateur).

4.3. Application à un cas réel d'installation

Nous collectons un maximum de cas réels d'utilisation afin de vérifier la pertinence du formalisme. À titre d'exemple le formalisme de *Score* est ici utilisé pour décrire le comportement d'une installation réalisée par Abtin Sarabi, étudiant à l'Institut Supérieur Des Arts de Toulouse (ISDAT).

En premier lieu voici une description textuelle du comportement de son installation : *le visage d'un prêtre est affiché sur une télévision. Tant que personne ne marche devant l'écran, plusieurs films montrent combien le prêtre s'ennuie (il regarde le ciel, sa montre, etc.). Mais si une personne marche devant l'écran (et que le film le montrant en train de s'ennuyer se termine), le prêtre commence à regarder la personne fixement. Ses yeux suivent la personne pour attirer son attention. Puis, lorsque la personne s'en va, le prêtre s'ennuie à nouveau.*

Afin de bien cerner la mise en œuvre de cette installation et donc l'écriture de son comportement, en voici la description des ressources : il y a un lecteur de film, une banque de films, un système de détection par caméra qui retourne s'il y a une présence et la position de cette présence. Nous considérons que l'état de ces ressources peut être adressé, mémorisé, rappelé ou écouté via des solutions protocoles.

Le *namespace* des ressources se résume ainsi :
/bank
/select : message pour sélectionner un film.

/movie
/play : message pour lancer le film.
/position : paramètre pour positionner la tête de lecture proportionnellement à la taille du film.
/end : retour avertissant que le film est terminé.

/detection
/presence : retour de détection d'une présence.
/position : retour de la position de la présence.

Dès lors nous avons tout le nécessaire pour formaliser avec *Score* la description textuelle du comportement de l'installation[7] :

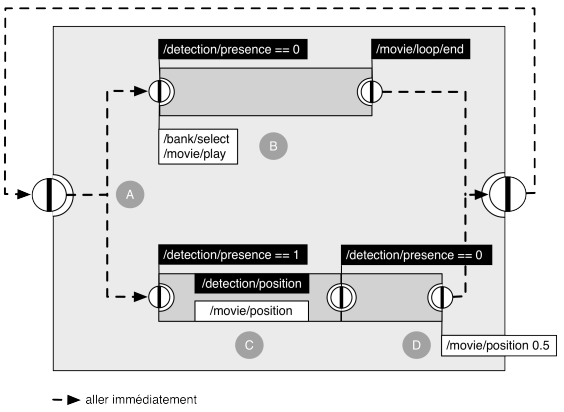


Figure 4. Scénario du cas réel d'installation formalisé

[7] Cette représentation graphique est amenée à évoluer à mesure que sa manipulation, pour modéliser des cas réels de scénario interactifs dans différents domaines, en montre les limites.

Le scénario de la figure 4 peut se traduire comme suit :

- A : Deux évènements interactifs sont conditionnés par l'état de */detection/presence* (*get*) pour sélectionner soit B (si égal à 0) soit C (si égal à 1).
- B : Sélectionne un autre film avec le message */bank/select* (*set*) et le lance en utilisant le message */movie/*play (*set*). Puis le retour */movie/end* est attendu (*listen*) avant de reboucler sur A.
- C : Mets en lien le retour */detection/position* (*listen*) sur le paramètre */movie/position* (*set*). Le mapping est actif jusqu'à ce que */detection/presence* devienne 0 (*listen*). Puis on passe à D.
- D : Ramène le paramètre */movie/position* depuis sa position courante (*get*) à 0.5 (*set*) en un temps précis avant de reboucler sur A.

Cette formalisation du comportement de l'installation prend tout son sens comparée à son implémentation dans l'environnement de programmation Max :

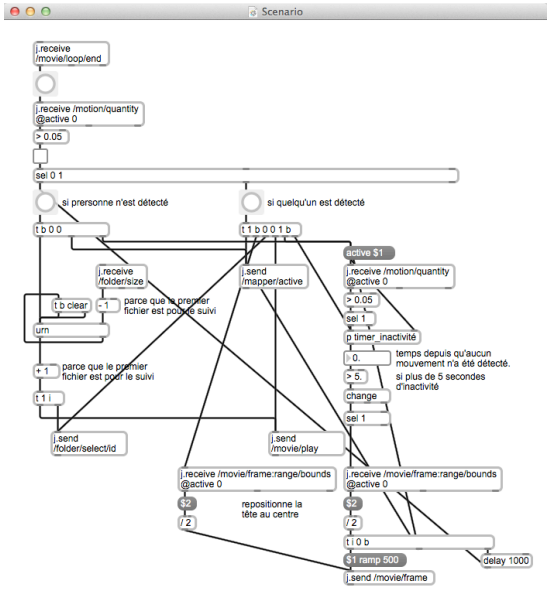


Figure 2. Scénario du cas réel d'installation dans Max

D'un point de vue syntaxique, *Score* offre donc la possibilité d'exprimer un même comportement logico-temporel en convoquant beaucoup moins d'éléments que d'autres langages de programmation. Néanmoins cette représentation graphique du formalisme n'est pas satisfaisante d'un point de vue ergonomique. La présentation ci-après du logiciel *i-score* montre un exemple d'application du formalisme de *Score* dans une manipulation de type séquenceur.

ARCHITECTURE DE LA LIBRAIRIE SCORE

La libraire *Score*[8] est organisée en quatre classes principales : *TimeEvent*, *TimeCondition*, *TimeProcess* et *TimeContainer*.

5.1 La classe *TimeEvent*

La classe *TimeEvent* possède une date indicative, un statut (*waiting*, *pending*, *happened*, *disposed*) et peut référencer des mémorisations d'états à rappeler lors de son activation. Un *TimeEvent* connait la *TimeCondition* à laquelle il est soumis ainsi que le *TimeContainer* auquel il appartient.

5.2 La classe *TimeCondition*

La classe *TimeCondition* possède une table d'évènements classé par *Expression* à évaluer pour valider l'activation ou non de chaque l'évènement. Une *TimeCondition* peut être prête ou non à s'évaluer, selon que tous les évènements qu'elle supervise sont *pending* ou non. La classe connait le *TimeContainer* auquel elle appartient.

5.3 La classe *TimeProcess*

La classe *TimeProcess* définit une interface pour la compilation, le démarrage, l'exécution et la terminaison d'un contenu temporel quelconque. Un *TimeProcess* se lie à deux *TimeEvent* (associés à son début et à sa fin) dont il observe le statut *happened* ou *disposed*. Lorsque le statut de son événement de début devient *happened*, un *TimeProcess* active une unité d'horloge et appelle une méthode virtuelle *ProcessStart* implémentée dans chacun des *plugins* de *TimeProcess*. À chaque pas de progression de l'unité d'horloge un *TimeProcess* appelle une méthode virtuelle *Process* implémentée dans chacun des *plugins* de *TimeProcess* et où la valeur de la progression est utilisée de manière spécifique. Lorsque le statut de son événement de fin devient *happened*, un *TimeProcess* désactive son unité de d'horloge et appelle une méthode virtuelle *ProcessEnd* implémentée dans chacun des *plugins* de *TimeProcess*.

À noter que le type d'horloge est sélectionnable selon que l'on souhaite se référer à l'horloge du système ou à une horloge externe par un système de *plugin* permettant de créer tout type d'horloge. Par exemple nous prévoyons la possibilité de synchroniser l'exécution via le réseau pour une exécution distribuée sur plusieurs ordinateurs.

La classe *TimeProcess* peut aussi définir une durée minimum et maximum au cas où la fin de son exécution soit déterminée par un *TimeEvent* soumis à une *TimeCondition* (bien que d'autres cas puissent aussi conduire à définir ces bornes).

Il existe pour l'instant deux *plugins* de *TimePro-*

[8] https://github.com/OSSIA/Score

cess : *Intervalle* et *Automation*. Nous prévoyons le développement d'un *plugin* de *mapping*, de générateurs et le système permet d'envisager le développement de *plugins* dédiés à la lecture de média.

5.4 La classe *TimeContainer*

La classe *TimeContainer* hérite de *TimeProcess*. La classe définie une interface pour l'ajout, la suppression et la supervision d'actions sur des listes de *TimeEvent*, *TimeCondition* et *TimeProcess*.

Il existe pour l'instant un seul *plugin* de *TimeContainer* : *Scenario* qui offre un gestionnaire de contrainte pour conserver les relations de précédence entre évènements lors de l'édition et compile un réseau de Petri pour en vérifier l'exécution.

IMPLEMENTATION DANS I-SCORE

Depuis octobre 2013 une équipe de développement du LaBRI entreprend une refonte complète du logiciel *i-score*[9] afin de mieux tirer partie des propriétés du formalisme proposé par Score tout en offrant plus de confort à l'utilisateur. Nous présentons ici le résultat de réflexions qui sont en cours d'implémentation dans une nouvelle version d'*i-score*.

6.1 Une représentation modulaire

Le logiciel *i-score* est une interface graphique permettant à l'utilisateur de manipuler simplement un scénario logico-temporel pendant ses phases d'édition ou d'exécution. L'utilisateur peut écrire un scénario en plaçant et en manipulant des objets graphiques sur une ligne de temps s'écoulant de la gauche vers la droite.

L'édition consiste en premier lieu à placer des évènements temporels dans le scénario (Figure 6.). Chaque évènement temporel (a) peut posséder un état (b) ou plusieurs (c), une interaction (d) ou plusieurs (e). Un état peut être lié à une interaction (f). A noter que l'ajout de plusieurs interactions autorise l'édition de choix ; lorsqu'une interaction est validée seul les éléments situés directement sous son influence sont exécutés.

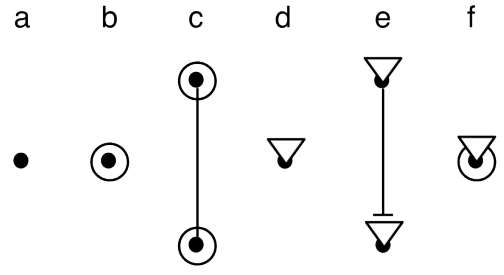


Figure 6. Évènement, état et interaction

[9] https://github.com/OSSIA/i-score

Par la suite un processus *intervalle*, symbolisé par une barre horizontale (Figure 7.), peut être inséré entre une paire d'évènements ; déclarant par là même une relation de précédence entre eux et ainsi une contrainte lors du déplacement de l'un des deux. A priori la durée est fixe (a) mais si une interaction est insérée sur l'événement de fin, la durée devient souple et l'interaction sera possible n'importe quand dès que l'événement de début sera passé (b). Afin de préciser une période d'interaction, il est possible de faire apparaitre une *fourchette* de réglage. La borne gauche de la *fourchette* permet de préciser à partir de quand l'interaction peut arriver (c, d). La borne droite de la *fourchette* permet de préciser quand l'interaction devra être déclenchée si elle n'arrivait pas (d) ; l'absence de borne droite signifiant que l'interaction peut être attendue indéfiniment (c).

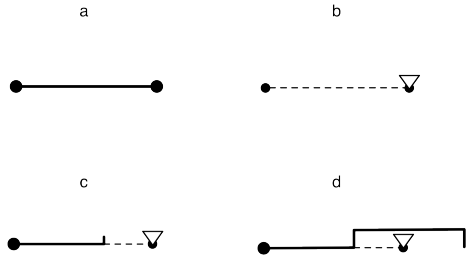


Figure 7. Processus, évènements et interaction

Un *intervalle* peut être transformé en un objet temporel plus complexe (Figure 8.). L'objet TimeBox peut contenir plusieurs processus partageant les mêmes évènements de début et de fin, et permettre de se replier sur son en-tête (b), ou de se dérouler sur plusieurs étages pour éditer le contenu de chacun de ses processus (c). Chaque *plugin* de processus possède une ou plusieurs représentations ; l'ensemble des représentations étant aussi architecturé en *plugin*. Dans le cas du processus *automation* plusieurs représentations sont envisagées selon le type du service à manipuler. Par exemple, une interface pour dessiner une trajectoire en deux dimensions sera préférable pour manipuler la conduite d'un paramètre de position, pour un paramètre de couleur une succession de gradient pourra être plus adapté, *etc*.

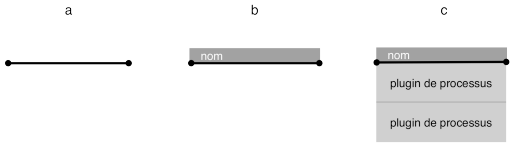


Figure 8. TimeBox

Pour une organisation hiérarchique du temps, un scénario est aussi un processus dans lequel tous les éléments graphiques présentés peuvent aussi être combinés au sein d'un sous-scénario (Figure 9.).

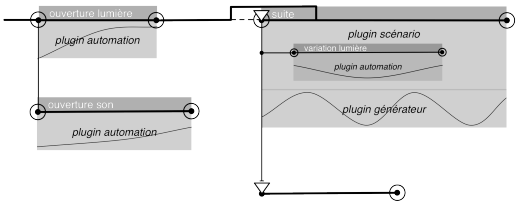


Figure 9. Exemple de scenario dans *i-score*

6.2. Prospections

Toutes les propriétés temporelles ne sont pas encore représentées dans *i-score* et font encore l'objet de discussions. Le cas des choix, lorsque que plusieurs interactions sont ajoutées à un événement, pose la question de la représentation du choix par défaut et d'une période de simultanéité durant laquelle plusieurs informations extérieures peuvent être attendues pour évaluer plusieurs interactions en même temps. La représentation des boucles reste aussi un problème. Par exemple, comment indiquer graphiquement l'équivalent d'un *do while* ou d'une boucle *for* ? Pour tous ces problèmes nous devons interroger les usages afin de déceler les démarches qui interviennent dans une écriture convoquant des choix ou des itérations.

CONCLUSION

Avec un moteur d'édition et d'exécution de scénario interactifs non dépendant des interfaces graphiques d'autres représentations que celle d'*i-score* sont imaginables selon les contextes d'utilisation et habitudes de travail : une représentation par *cuelist* peut être plus appropriée dans un contexte de spectacle vivant, un accès purement logique être souhaité dans un contexte muséographique tandis que certains outils de développement de jeux vidéos adopte une représentations spatiales pour des scénarisations non-linéaires. Cette indépendance permet aussi d'envisager qu'une fois écrit, un scénario interactif puisse être exécuté sur une plateforme plus légère sans soucier de sa représentation.

C'est pour toutes ces raisons que le projet OSSIA s'attache à rendre le formalisme de *Score* ouvert d'un point de vue logiciel en proposant une architecture de *plugin* pour les processus et les contenant temporel et qu'une API est en cours de rédaction. Grâce à un formalisme partagé et expérimenté dans des contextes de développements variés[10] tels que les jeux-vidéos, le spectacle vivant ou la muséographie, nous espérons proposer une librairie unifiée, générique et suffisamment souple pour répondre aux besoins spécifiques des écritures intermédias.

[10] Une implémentation de *Score* dans Max est prévue. Nous espérons ainsi permettre de créer rapidement des moteurs d'écritures spécifiques.

REFERENCES

[1] Baltazar, P., Gagneré, G. « Outils et pratiques du sonore dans le spectacle vivant », Actes des 12e Journées d'Informatique Musicale, Lyon, avril 2007. http://www.afim-asso.org/IMG/pdf/GT_son_spect_viv.pdf

[2] Santini, A., Sèdes, A., Simon, B. « Virage : Analyse des usages/État de l'art », rapport interne, version n° 3, Paris, 2009.

[3] Baltazar, P., Allombert, A., Marczak, R., Couturier, J-M., Roy, M., Sèdes, A., Desainte-Catherine, M. « Virage : Une réflexion pluridisciplinaire autour du temps dans la création numérique », Actes des 14e Journées d'Informatique Musicale, Grenoble, 2009.

[4] Meyssonier T. « Analyse des relations entre création artistique, modélisation mathématique et implémentation logicielle dans la problématique de l'écriture de structures temporelles », Rapport de stage effectué au LaBRI sous la direction de Desainte-Catherine M., Bordeaux, 2013.

[5] Allombert, A., Desainte-Catherine, M., Laralde J., Assayag, G. « A System of Interactive Scores Based on Qualitative and Quantitative Temporal Constraints », p1-8, Proceedings of ARTECH the Fourth International Conference on Digital Arts, Porto, Portugal, 2008.

[6] Toro, M., Desainte-Catherine, M., « Concurrent Constraint Conditional Branching Interactive Scores », p270-273, Proceedings of Sound and Music Computing, Barcelone, Espagne, Juillet 2010.

[7] Arias, J., Desainte-Catherine, M., Salvati, S., Rueda, C., « Executing Hierarchical Interactive Scores in Reactive ML », Journées d'Informatique Musicale, Bourges, Mai 2014, en soumission.

[8] Desainte-Catherine, M., Allombert, A., Assayag, G. « Towards a Hybrid Temporal Paradigm for Musical Composition and Performance : The Case of Musical Interpretation », p61-72, Vol. 37, No. 2, Computer Music Journal Summer, 2013.

[9] Marczak, R., Desainte-Catherine, M., Allombert, A. « Real-Time Temporal Control of Musical Processes », in Proceedings of the Third International Conferences on Advances in Multimedia, Budapest, Hongrie, Avril 2011.

[10] Allombert, A., Marczak, R., Desainte-Catherine, M., Baltazar, P., Garnier, L. « Virage : Designing an interactive intermedia sequencer from users requirements and the background », International Computer Music Conference, New York, USA, Juin 2010.

Journée Transformation
jeudi, 22 mai 2014

FROM CONCEPT TO SOUND: TRANSFORMATION THROUGH LIVE INTERACTIVE COMPOSITION

DOMINANTE DE LA JOURNEE (KEYNOTE) - 1

Jônatas Manzolli
Interdisciplinary Nucleus for Sound Studies (NICS)
Art Institute, Music Department
University of Campinas (UNICAMP) - Brazil
jonatas@nics.unicamp.br

No one knows exactly what went through the mind of pre-historic man when he first struck two rough stones together in order to produce sounds and rhythms. To what extent did the resulting reverberations captivate him? To what extent was he already projecting his own ideas in order to transform what he was doing? As the millenniums passed, he continued polishing his stones, producing more and more sophisticated instruments. In the era of digital sound generation, real time interactions and virtually unbounded possibilities, the computer has become our new stone with which we project and transform our ideas into sounds.

Music composition has evolved from symbolic notated pitches to expressions of the internal organization of sound. This can be observed in the extended instrumental techniques developed from the 1940's onwards up to the more recent compositional strategies that have emerged from the "new interfaces for musical expression". The dynamic organization of sound material in "real" time, however, adds new dimensions to musical information and to its symbolic representations.

This talk will explore the convergence between ideas developed as a result of live interactive composition and ideas that have emerged in an interdisciplinary framework involving mathematical modeling, computer and data processing and models for real time interaction. We will introduce the interdisciplinary research activities developed at NICS/Unicamp Brazil, and will discuss a conceptual view point anchored in the development of systems that produce sound material in real time, through the iteration of simple rules and independent of symbolic notation.

With the advent of new technologies that have emphasized interaction and novel music interfaces, alternative forms and modes of interactive media have been realized. These developments raise fundamental questions regarding the role of embodiment as well as the environment and interaction in live interactive composition focusing on our understanding of the man-machine interplay. In addition, it emphasizes a more situated and externalist view.

The study that we are developing at NICS is to integrate algorithmic composition and interactive narratives with scientific sonification and visualization. It is possible to conceive new methods of combining music interface technologies within a theoretical approach driven towards shaping human sensory experiences through new forms of enhanced perception and action through interplay with music performance. For example, new interactive technologies such as interactive environments can function as a laboratory setting where we can test computational models and interactive musical behavior. Aligned with this perspective, we are working with multimodal

interaction, interactive installations and audiovisual works to combine multimodalities using interactive media in order to produce immersion, based on the concept of Presence.

The structural foundation of this endeavor is not a written score or textual narrative, but rather the concept of recursion used as a way of constructing musical meaning, or, more specifically, the interaction between human agents and machines that produced recurring changes in the physical world. Our aim is to experience the ways in which internal and external representations of the world can be joined together in a performance to create sound and video.

These experiences raise a fundamental question: Is a universal, even basic, structure underlying human auditory and visual experiences that is based on a single cognitive function in the brain, as opposed to one that is fragmented along a number of modality-specific properties? This invariant function could be equated with the brain's drive to find meaning in events organized in time, or to define a narrative structure for multimodal experiences to which it is exposed.

To illustrate our research trajectory, we will present a number of musical systems and works develop at NICS that have emerged from the concepts expressed during this lecture. In short, these works illustrate that the aesthetic experience can be at least partially obtained from the emergence of structures produced as a result of the interaction between humans and interactive systems.

EXPERIMENTING WITH TRANSFORMATION

DOMINANTE DE LA JOURNEE (KEYNOTE) - 2

Fernando Iazzetta
Universidade de São Paulo - Brazil
iazzetta@usp.br

Transformation: to change the form of something, to cross the borders of one state to reach another one. That is exactly what we do when we use technologies in creative processes. Transformation is one of main leading ideas that guide the activities of NuSom - Núcleo de Pesquisas em Sonologia (Research Centre on Sonology) at the University of São Paulo. In the last ten years a group of artists and researchers related to NuSom have been working on the interfaces between academic research and artistic creation, exploring local conditions and facilities to transform the artistic experience of its members and audiences. Most of our work is guided by the concept of experimentation. The term is taken here in a double sense: as an empirical method to test something; and as an attitude of exploring what is not yet established or finalized, sometimes by transgressing or reinventing artistic boundaries. In this talk I'll briefly present some initiatives at NuSom in the direction of transforming technologies, practices and concepts within a creative framework. These initiatives will be assembled in four thematic fields:

1) transforming spaces: extrapolating the ideas of sound projection and sound localization often recalled when one think about spatiality in music, we present a few projects that explore space as a musical parameter, among them, a series of network concerts and installation projects;

2) transforming context: leaving the concert hall to explore other social spaces can lead to very rich experiences in terms of reallocating the roles of artists and audiences; part of our works focus on the incorporation of contextual situations as the material for music creation;

3) transforming forms: by recombining musical practices, technologies and genres, we try to transfigure regular approaches to music creation into new ones, for example, by recreating electroacoustic music as instrumental performances or revisiting standard music works in collective recreations;

4) transforming actions: the "classic" issues of music interaction and gesture are taken into consideration in an experimental perspective to construct new instruments and performative instances.

For each of these four fields we will provide a couple of examples among the works created by the NuSom in the last years and will point out a few issues related to the use of music technology in a creative way.

DEPARTEMENT DE MUSIQUE ELECTROACOUSTIQUE ET DE CREATION, CONSERVATOIRE DE MUSIQUE ET DE DANSE DE BOURGES

Roger Cochini
Professeur
Conservatoire de musique et de danse de Bourges
34 rue Henri Sellier, 18000 Bourges
roger.cochini@wanadoo.fr

Philippe Macé
Directeur
Conservatoire de musique et de danse de Bourges
34 rue Henri Sellier, 18000 Bourges

1. INTRODUCTION, GENESE

La Classe de Musique électroacoustique du Conservatoire de Bourges est l'héritière d'une très ancienne tradition d'enseignement et de formation en musique électroacoustique dans la ville et au delà de la ville.

Dès le début des années 1970, année de sa fondation, le Groupe de Musique Expérimentale de Bourges, devenu IMEB, dirigé par Françoise Barrière et Christian Clozier, ouvre un studio de création dédié aux amateurs, le studio Marco Polo.

Cette activité est immédiatement suivie par un stage international professionnel de deux années, en collaboration avec le CROUS de Paris, ayant pour vocation de former de jeunes compositeurs qui iront à leur tour créer des studios dans le monde. L'enseignement y est assuré par les compositeurs Alain Savouret et Roger Cochini.

En 1981, la Direction de la Musique du Ministère décide de mettre en place des mesures économiques incitatives, pour créer des classes de musique électroacoustiques dans les conservatoires. C'est ainsi qu'est fondée la classe de Bourges initialement accueillie dans les locaux du GMEB. Le premier enseignant en fut Pierre Boeswillwald. Depuis 1983 le professeur titulaire actuel est Roger Cochini. L'assistant de la classe est Stéphane Joly.

Conjointement, en 1972 Madame Renée Andraud alors inspectrice des écoles maternelles du département du Cher, demande au Gmeb de contribuer à l'éveil au monde sonore et à l'éducation de l'écoute des tout-petits, en s'appuyant sur les apports de la discipline électroacoustique. C'est le début d'une très longue aventure pédagogique en direction de la petite enfance, de l'enfance et de la jeunesse.

La classe de Bourges est fondée sur cet héritage patrimonial.

2. LES STUDIOS

D'abord à l'extérieur du conservatoire et depuis 2007, dans un conservatoire très récemment construit, le département de musique électroacoustique déploie des activités intra muros articulées autour de 6 studios.

Pierre Schaeffer, Pierre Henry, Charles Cros, Luc Ferrari, Edgard Varèse, Henri Dutilleux.

Chacun d'entre eux comporte un poste de travail individuel pour les travaux pratiques des élèves. Un ordinateur iMac 20 pouces, une carte son, un disque dur externe, une console de mixage, des Haut-parleurs Dynaudio ou Genelec, une matrice audio d'entrée/sortie permettant d'accueillir entre autres, les ordinateurs personnels des élèves.

Le studio Edgard Varèse est aussi équipé d'un instrument collectif accueillant des groupes d'enfants.

Le studio Henri Dutilleux est aussi équipé d'une station 8 pistes accueillant les étudiants du Postdiplôme Art et Création Sonore, organisé en partenariat avec l'Ecole Nationale Supérieure d'Art de Bourges.

Un ensemble mobile est constitué d'équipements de prise de son, synthèse, claviers et contrôleurs MIDI…

3. LA PEDAGOGIE ET LES ACTIVITES

Les activités sont articulées autour de 3 axes principaux :

- Des activités spécifiquement électroacoustiques ;
- Des activités internes interdisciplinaires ;
- Des activités externes interprofessionnelles ;

Elles accueillent des publics depuis l'âge de 5ans jusqu'à l'enseignement supérieur.

L'enseignement électroacoustique diplômant : 1er, 2ème, 3ème cycles CEM, et spécialisé DEM, et classes de perfectionnement.

Ateliers d'Art Sonore non diplômant, pout tous les publics, offrant une formation fondamentale sonore aux multiples activités artistiques. Radio, danse, vidéo, arts plastiques, pratiques live et mixtes, éducation, rééducation,…

Interdisciplinarité avec la Formation Musicale, le Chant la Danse et les instruments.

Partenariats : Education Nationale, école élémentaire, collèges, lycées, conseillers pédagogiques, Classes à Horaires aménagés primaires et collèges; Friche culturelle, Ecole nationale supérieure d'Art, Abbaye de Noirlac, Ina-GRM, et autres.

4. CONCERTS

Le dernier concert a eu lieu le jeudi 10 avril dernier. Il comportait des créations avec des élèves communs aux classes de percussion, danse contemporaine, musique électroacoustique.

5. CONCLUSION.

Poursuivre et réussir l'intégration de la musique électroacoustique dans les conservatoires.

STUDIO REPORT: CONSERVATOIRE DE TOURS

Vittorio Montalti

CRR Tours: Conservatoire Francis Poulenc
2 Ter Rue du Petit Pré
37000 Tours
France

e-mail: vittoriomontalti@gmail.com

Abstract

Le studio fait partie du departement d'écriture du Conservatoire à rayonnement régional de Tours.
Il s'agit d'un studio pour produire de la musique électronique dont les matériaux peuvent être utilisés pour faire des concerts de musique acousmatique et de musique mixte.
La configuration et l'installation du studio a été fait par Vittorio Montalti.

1. Introduction

Le studio est né en 2013 par une forte volonté de Christophe Wallet (directeur du Conservatoire) et Anne Aubert (professeur d'écriture) de créer un cours de musique électronique et nouvelles technologies adressée aux étudiants du cours de composition d'Alessandro Solbiati, qui enseigne au Conservatoire de Tours depuis le 2012.

2. Configuration du studio

Le studio du Conservatoire de Tours est équipé d' un ordinateur *Mac Pro* (processeur Quad-Core Intel Xeon à 2,8 Ghz, 3 Go de mémoire, disque dur de 1 To) avec deux moniteurs *Samsung T220*.
La machine est reliée à une carte son *RME Fireface 800* avec une connexion firewire.
La carte est reliée à la table de mixage numérique *Yamaha 01v96i* par une connexion ADAT.
Enfin la table de mixage est connecté à quatre Monitors *Yamaha MSP7* et à un Subwoofer *Yamaha SW 10*.

Pour l'écoute en casque on utilise des casques *AKG K99*.

Pour l'énregistrement on utilise deux micros Neumann KM184.

Le studio est aussi equipé d'un clavier midi *M-Audio Keystation 88es* et d'une pedale midi *M-Audio SP-2*.

Au niveau de software le studio est équipé avec les suivantes logiciels: *Pro Tools 10, N-Track, Grm Tools, Max Msp 6, Ableton Live 9, Csound, UviWorkstation, Audacity, Open Music, PWGL, Soundflower, SoundFilesMerger, Spear, Vlc, Finale*.

3. Pedagogie

Pour l'instant le studio a été utilisé surtout au niveau pedagogique.
Vittorio Montalti a donné trois stage, visant à intégrer les cours de composition de Solbiati dans le but d'amener les élèves à écrire leur premier pièce de musique électronique.

Les premieres trois stage on été sur:

- MaxMsp: introduction à MaxMsp, Synthese et Traitements.
- Open Music: introduction à OM
- DAW: introduction à Logic et N-Track

Il a eu aussi des cours individuels centrée plutot sur le travail compositionnel.

4. Concerts

Le materiaux du studio ont été utilisés aussi pour la réalisation d'un concert de musique mixte et amplifiée.

Le jeudi 7 novembre 2013 l'ATMUSICA de Tours a proposé une programme avec une tryptique de Daniel D'Adamo (*Lips, your lips, Keep your furies, Air lié;* synthèse de ses études sur le souffle, l'émission du son, la dynamique …) et des pièces amplifié de Helmut Lachenmann (*Pression*) et George Crumb (*Vox balanae*).
La musique a été jouée par Isabel Soccoja (mezzo-soprano), Nicolas Vallette (flûtes), Agnès Bonjean (piano), Delphine Biron (violoncelle) et Charles Bascou (réalisateur en informatique musicale).

Enfin les materiaux du studio ont été utilisés pour enregistrer le concert du premiere anné des élèves de composition d'Alessandro Solbiati.

5. Conclusion

Le studio de Tours est équipé avec des matériaux de haute qualité et se prête à des productions professionnelles et des concerts dans des petits espaces.

On espère que le studio soit utilisé de plus en plus non seulement pour l'activité pedagogique mais aussi pour des production de musique électronique des étudiants et des compositeurs invité.

VERS UN MODELE POUR L'ANALYSE DE L'ORCHESTRATION : RAPPORT DE RECHERHE EN COURS

Didier Guigue
UBPB – *Mus*^{3} – CNPQ
didierguigue@gmail.com

RÉSUMÉ

Cette communication présente l'état actuel de mon projet d'élaboration d'un modèle méthodologique pour l'analyse de l'impact que les stratégies d'orchestration peuvent avoir sur la dynamique formelle et sa perception. Le modèle aborde l'œuvre à partir de deux angles: la partition, de laquelle sont extraites les informations d'orchestration et instrumentation consignées par le compositeur, et un ou plusieurs enregistrements, qui sont analysés par des descriptuers ad hoc afin de juger de l'effet concret de ces prescriptions, et de voir jusqu'à quel point leur impact sur la forme en est le reflet. Ce modèle fait appel a des outils informatiques qui aident le musicologue dans l'obtention des données nécessaires. Ceux-ci sont développés dans Audiosculpt et OpenMusic. Au final, ces outils pourraient aboutir à une application dédiée. Ce projet, qui en est à son début, est réalisé dans le cadre d'une collaboration entre le groupe de recherche Mus3 et le centre de recherche NICS, au Brésil, et l'IRCAM, en France, et reçoit l'apport financier du CNPQ au Brésil.

1. UNE PROCEDURE POUR L'ANALYSE DE L'ORCHESTRATION

Le processus de « désymbolisation » des instruments de l'orchestre, entamé au début du XIXe siècle, en particulier par Berlioz ([20]:63), a contribué à l'intégration définitive des stratégies d'orchestration aux dimensions structurantes de la composition. Mon objectif est d'établir, au moyen d'applications expérimentales sur divers répertoires, une procédure d'analyse qui permettrait d'évaluer l'impact de ces stratégies sur la structure ou sur l'expression musicales. En effet, si le développement de logiciels d'aide à l'orchestration a commencé à mettre récemment à la disposition du compositeur plusieurs produits fonctionnels [4, 19], la littérature n'a encore apporté que très peu de propositions qui satisfassent à une approche analytique systématique de l'utilisation de ressources orchestrales pour l'expression de la forme. Tout ce que nous disposons sur ce sujet fournit, au mieux, quelques bons indices [2, 7, 10, 12, 22], ou consiste en des approches limitées ou trop simplistes [5, 9]. Ma contribution ne se situe pas au niveau d'un logiciel dédié, tout au moins pour l'instant; elle propose un cadre méthodologique, et quelques outils informatiques dans l'environnement *OpenMusic*, dont le but est d'aider le musicologue dans l'organisation des données pertinentes et l'appréciation de leur fonctionalité.

La figure 1 reproduit l'organigramme de la procédure envisagée. La terminologie anglophone en sera conservée dans la description qui suit. Dans ce modèle, l'orchestration est évaluée à partir de deux points de départ. D'un côté – *Score Analysis* – on recense les prescriptions consignées par le compositeur dans la partition, en comptabilisant les diverses combinaisons instrumentales *(Local Sonic Setup* ou LSS) utilisées au cours de l'œuvre et en les qualifiant en fonction de leur organisation interne.

D'autre part – *Sound Analysis* –, les fichiers audio de ces mêmes LSS sont auscultés au moyen de descripteurs ad hoc afin de jauger l'effet concret des prescriptions, et de voir jusqu'à quel point leur impact sur la forme en est le reflet.

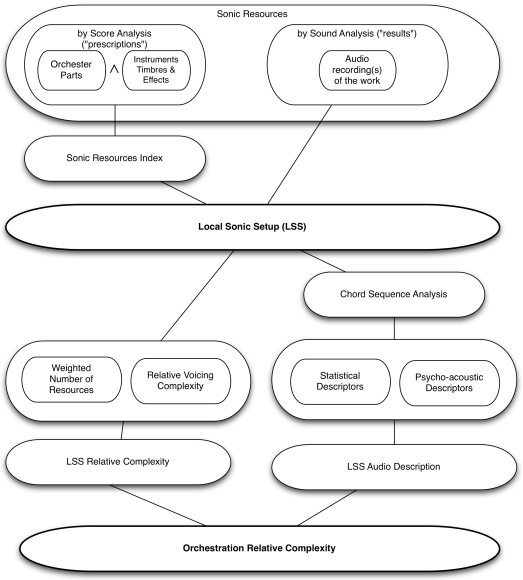


Figure 1. Organigramme du modèle d'analyse de l'orchestration.

En d'autres termes, il s'agit d'aborder l'objet d'étude en même temps au niveau prescriptif du code écrit et au niveau concret de sa réalisation sonore, deux angles d'approche dont on pressent que la convergence et synchronie ne seront pas toujours au rendez-vous. L'objectif est de rendre le modèle capable de supporter ces éventuelles dichotomies comme intrinsèques à la

dimension orchestration, étant donné que l'infinité d'impondérables qui peuvent intervenir dans la réalisation d'une œuvre pour orchestre en situation de concert n'a jamais inhibé les compositeurs dans leur volonté de fixer sur le papier les moindres détails de l'exécution, comme si ceux-ci fussent aussi objectifs et stables qu'une indication métronomique.

Pour illustrer cet exposé, j'ai choisi un fragment de la 5e Symphonie de Beethoven (1807), précisément la transition qui relie la fin du 3e mouvement (à partir de la m. 237) au début du 4e mouvement (m.1-4). Cet extrait me semble caractéristique de l'importance du rôle de l'orchestration dans la gestion de la tension dramatique chez ce compositeur. En effet, il part d'un presque rien sonore (Figure 2) pour aboutir progressivement à un colossal tutti orchestral, sans précédent à l'époque pour une symphonie (Figure 3). Cette transition provoque comme une suspension thématique et temporelle dans laquelle règne la fragmentation sonore. C'est pourquoi elle m'a paru offrir un excellent terrain d'expérience.

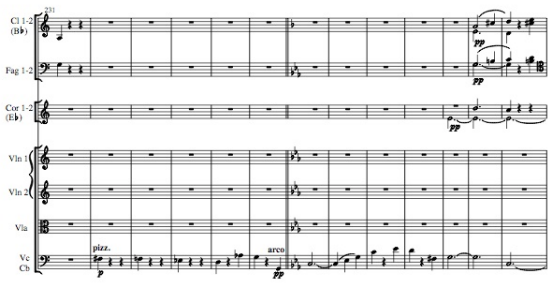


Figure 2. Beethoven, Symphonie nº5. IIIème mouvement, m. 231-239.

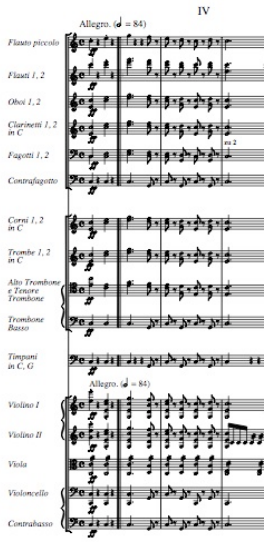


Figure 3. Beethoven, Symphonie nº5. IVème mouvement, m. 1-4.

2. *SCORE ANALYSIS* : EVALUATION DE LA COMPLEXITE RELATIVE DES CONFIGURATIONS INSTRUMENTALES

Le premier volet du modèle, qui analyse les prescriptions compositionnelles, a été décrit récemment en détail [18]. Je ne vais qu'en résumer ici l'essentiel.

L'œuvre (ou le fragment) est découpée en une séquence de "configurations instrumentales locales" (*Local Sonic Setup*). Ces configurations sont considérées comme des sous-ensembles de l'ensemble des instruments, techniques de jeu et effets que le compositeur utilise dans l'œuvre – ce que j'appelle les "ressources sonores" (*Sonic Resources*), dont on aura préalablement établi un *Index* complet (Figure 4). Sur un vecteur du simple au complexe, plus une configuration s'approche de l'ensemble, plus elle sera dite "complexe" et recevra un poids proche de (1.0)[1].

Cette quantification (*Weighted Number of Resources*, ou WNR) est ensuite qualifiée par un composant appelé *Relative Voicing Complexity* (RVC), qui évalue la manière par laquelles les ressources sonores sont combinées dans chaque setup en des flux plus ou moins indépendants. Je prends comme base une relecture de la Théorie des Partitions par Gentil-Nunes [11], ainsi que le modèle d'analyse des textures de Wallace Berry [2]. Plus les ressources sont "agglomérées" (homophonie, homorythmie, …), plus le *Voicing* est considéré comme "simple"; et plus elles sont "dispersées", "complexe". RVC agit alors comme un modulateur de WNR, c'est-à-dire qu'il en modifie la pondération initiale dans une proportion qui va de zéro à un maximum fixé par l'utilisateur. D'une manière générale, on peut s'attendre à observer une plus grande indépendance des parties instrumentales (pondérations plus élevées de RVC) chez un Webern que chez Mozart, par exemple, mais, par contre, il est probable que les configurations locales des symphonies de ce dernier soient plus fournies (WNR plus élevé) que celles du compositeur sériel [2].

On obtient en synthèse une évaluation globale du setup appelée *Local Sonic Setup Relative Complexity*. Une feuille de calcul *Excell* (Figure 4) et un patch *OpenMusic* (Figure 5) facilitent ces opérations, encore que la collecte des données de base doive se faire à la main et à l'œil, si je puis dire, car un fichier numérique ou encodé en MIDI de la partition est de peu de secours pour ces finalités. Un graphique (Figure 6) montre l'évolution de la complexité relative des setup de l'extrait de la Symphonie de Beethoven.

[1] Je reviens sur la notion de *complexité relative* plus avant.

[2] A propos de cet aspect de l'orchestration de Webern, cf. [17, 18, 21].

	A	B	C	D	E
1	**SRI || bars->**	**237**	**242**	**246**	**252**
2	piccolo				
3	flauto I				
4	flauto II				
5	Oboe I				
6	Oboe II				
7	Clarinetto I		1		
8	Clarinetto II		1		
9	Fagotto I		1	1	1
10	Fagotto II		1		
11	Contrafag.				
12	Corno I		1		
13	Corno II		1		
14	Tromba I				
15	Tromba II				
16	Trombone Alt				
17	Trombone Ten				
18	Trombone Bs				
19	Timp.				
20	Violino I arco				
21	Violino I pizz				1
22	Violino II arco				
23	Violino II pizz				1
24	Viola arco				
25	Viola pizz				1
26	Cello arco	1	1		
27	Cello pizz			1	1
28	CB arco	1	1		
29	CB pizz				
30	**LSS**	**2**	**8**	**2**	**5**
31	WNR	0,21	0,62	0,21	0,48
32	Relative Voicing Compl	0,08	0,64	0,08	0,73
33	**Setup Rel Complexit**	**0,22**	**0,82**	**0,22**	**0,66**

Figure 4. Vue partielle de la feuille de calcul Excell pour le relevé et l'analyse des *Local Sonic Setup*. La colonne de gauche constitue l'*Index des Ressources Sonores* (SRI) utilisées dans la 5e Symphonie. Les setup (disposés sur l'axe horizontal) sont étiquetés par le numéro de mesure où ils commencent.

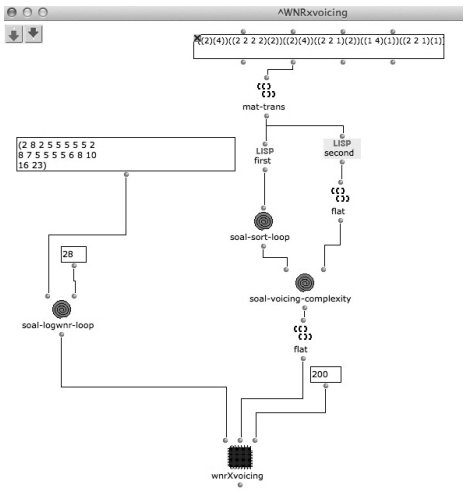


Figure 5. Premier niveau du patch *OpenMusic* pour le relevé et l'analyse des *Local Sonic Setup*.

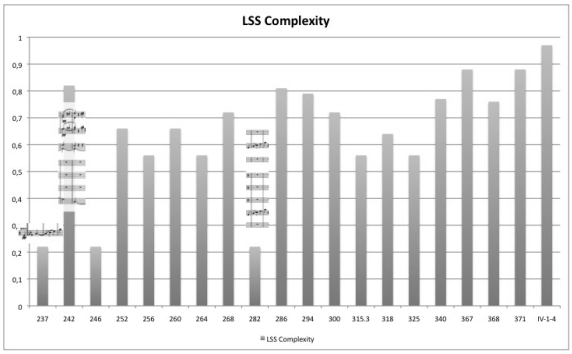


Figure 6. Histogramme de l'évolution de la complexité relative des *Local Sonic Setup* dans l'extrait de Beethoven. Les données sont étiquetées par le numéro de mesure.

3. *SOUND ANALYSIS* : ANALYSE DU RESULTAT SONORE DES PRESCRIPTIONS ECRITES

3.1. Préparation du matériel audio

Le découpage de la partition en unités sonores consistantes du point de vue de l'orchestration, expliqué ci-dessus, doit être reproduit à l'identique pour le fichier audio de l'œuvre, ceci afin d'analyser comment et jusqu'à quel point se concrétisent les prescriptions écrites, et d'évaluer les convergences et divergences entre l'une et l'autre[3]. Cette tâche peut être automatisée grace à la fonction *Generate Markers* disponible dans le logiciel *Audiosculpt*[4]. La programmation adéquate des paramètres du générateur exige quelques essais et dépend en tout état de cause de l'œuvre et de son enregistrement. Selon mon expérience, la segmentation par *Positive Spectral Differencing* détecte très finement les changements d'énergie, lesquels, en règle générale, correspondent aux changements de sonorités, c'est-à-dire de *setup*. On ajuste ensuite manuellement les marqueurs qui ne seraient pas placées exactement où on le souhaite – une attaque synchrone sur la partition peut ne pas l'être autant dans le monde réel de sa réalisation acoustique – , on ajoute les segments que la partition induit mais qui sont passés inarperçus à la détection (ou on reparamètre la fonction pour qu'elle les reconnaisse), on retranche ceux qui ne correspondent pas à la segmentation préétablie – à moins que l'on ne préfère les incorporer au découpage original (et revenir à l'étape antérieure).

L'étape suivante, toujours dans *Audiosculpt*, utilise la fonction *Chord Sequence Analysis* et sa méthode de calcul du spectre moyen (*Average Spectrum*). *Audiosculpt* entend par « accord » un groupe de partiels et d'amplitudes normalisés, dont le début et la fin sont déterminées par les marqueurs. L'analyse calcule la moyenne du spectrogramme sur le segment et fait coincider un partiel à chaque pic [8]. J'ai utilisé, pour la musique d'orchestre, un nombre maximum entre 40 et 60 partiels, et un seuil d'amplitude de -70 dB ou lié au sonogramme. Les résultats de l'analyse (comme celle représentée Figure 7) sont sauvegardés, d'une part sous forme d'une collection de petits fichiers audio dont chacun correspond à un segment, donc à un LSS, et d'autre part sous forme d'un fichier unique SDIF. C'est ce matériel qui va être requis pour l'analyse de l'orchestration proprement dite, dans *OpenMusic*.

[3] C'est aussi par ce moyen que l'on pourra analyser divergences entre enregistrements différents de la même oeuvre.

[4] *Audiosculpt* n'est bien sûr pas le seul logiciel sur le marché à proposer une génération automatique de marqueurs. Mais ce sont les fonctionalités complémentaires auxquelles je fais appel qui le rendent préférable pour mon projet.

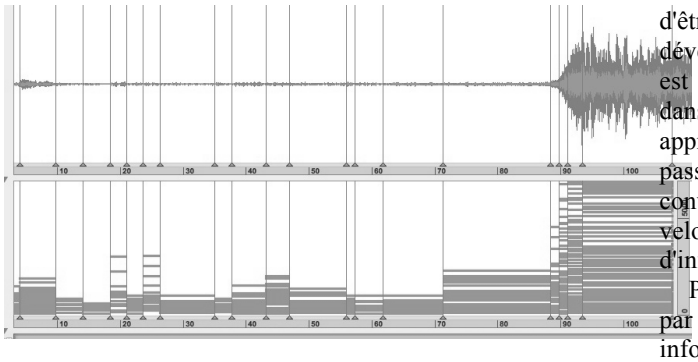


Figure 7. Une capture d'écran d' *Audiosculpt* montrant, fenêtre du bas, les résultats de la *Chord Sequence Analysis by Average Spectrum*, pour le même extrait de Beethoven. Les LSS sont identifiés par les marqueurs verticaux, et chaque ligne horizontale correspond à un des partiels détectés par l'analyse.

3.2. *SOAL* et l'analyse de l'orchestration

Dans le cadre de mon projet d'analyse de la musique à partir du concept de *sonorité* [13, 14, 15], j'ai été amené, avec mon groupe de recherches *Mus*[3] et en convention avec l'IRCAM, à développer une bibliothèque dédiée pour *OpenMusic*, *SOAL* (pour *Sonic Object Analysis Library*) [16][5]. Le concept-clef de la bibliothèque est la segmentation de l'œuvre en une séquence d'*unités sonores composées* (*sonic object* dans le jargon de *SOAL*) ([15]:37 *et sq.*) et d'analyser l'évolution de celles-ci au cours de l'œuvre par divers descripteurs (appelés *composants*) de leurs caractéristiques « hors-temps » et « en-temps ». Ces composants sont, pour la plupart, d'ordre statistique: densités, ambitus, modes de distribution « spatiale » et temporelle des sons, entropie relative, entre autres. La règle pour l'évaluation de l'importance d'un composant donné est le calcul de son taux de *complexité relative*. Celle-ci établit un poids maximum (1.0), qui correspond à la configuration musicale qui rendrait la sonorité la plus "complexe" possible, dans le domaine du composant. Les configurations observées sont donc calibrées sur un vecteur de simplicité/complexité relative[6]. Cette règle de relativité universelle est fondamentale (tous les composants doivent pouvoir se soumettre à cette normalisation) pour le succès de l'analyse, car c'est le seul moyen de comparer et manipuler des paramètres musicaux hétérogènes, dont les qualités intrinsèques ne sont pas comparables ([15]:40 *et sq.*).

Programmé d'abord exclusivement pour l'analyse de fichiers MIDI, l'interface de lecture *soal-reader* a évolué et accepte depuis, entre autres, les fichiers au format SDIF. En fait, les données transmises par le format SDIF sont converties internement au format MIDI avant d'être traitées, ce qui permet de tirer profit des fonctions développées jusqu'alors pour le MIDI. Le midicent, qui est l'unité standard pour la codification des hauteurs dans OpenMusic, permet une transcription suffisamment approchée des fréquences exprimées en Herz (sans passer par une normalisation chromatique). Les autres conversions (time en onsets, amplitudes linéaires en velocity) se font aussi sans perte significative d'informations.

Parmi la collection de fonctions analytiques proposée par *SOAL*, un certain nombre peut déjà fournir des informations pertinentes pour notre objectif, soit en l'état, soit adaptées ou incluses dans un patch approprié. Je ne vais pas redécrire ici ces fonctions, encore moins leurs algoritmes, puisque ces informations sont disponibles depuis longtemps en ligne[7] . Je vais plutôt mettre l'accent sur la contribution qu'elles peuvent apporter à une compréhension de la fonction de l'orchestration sur la forme.

L'analyse réalisée par *Audiosculpt* retient pour chaque setup un certain nombre de partiels. *SOAL* peut extraire de cela des séries de données, toutes normalisées sur le vecteur simplicité-complexité et qui, une fois transposées visuellement (graphes, histogrammes…), donnent une image de l'évolution de l'impact de l'utilisation des ressources sonores sur la forme en fonction: (1) du nombre de partiels par setup; (2) de la densité relative de partiels par setup; (3) de la distribution de ces partiels le long du spectre, sur un vecteur où la distribution la plus harmonique est pondérée comme la plus "simple"; (4) de l'ambitus relatif – c'est-à-dire la distance observée entre les premier et dernier partiels – de chaque setup; (5) de la manière dont les partiels se répartissent le long du spectre, donnant si nécessaire un relevé du nombre de partiels pour chaque bande de fréquence choisie, et une pondération des setup en fonction du registre des bandes occupées[8] ; (5) de l'amplitude moyenne relative de chaque setup; et, si pertinent dans le contexte, (6) de la durée relative de chaque setup.

A titre d'exemple de la fonctionalité de ces descriptions, la Figure 8 met en corrélation l'évolution des LSS telle que déjà montrée Figure 6, avec celle des densités relatives et de la « linéarité » relative. Pour cette dernière, il faut noter que les pondérations les plus basses indiquent une structure spectrale qui tend à l'harmonicité, comme c'est le cas du dernier LSS, qui correspond au tutti en Do Majeur du début du 4e mouvement de la symphonie. La figure 9 est une capture d'écran du patch *OpenMusic* montrant le fonctionnement de *SOAL* à partir du fichier SDIF de Beethoven.

[5] Accessible, avec sa documentation, sur:www.ccta.ufpb.br/mus3.

[6] Pour la densité, par exemple, le vecteur circulera du *vide* au *saturé*.

[7] Ainsi que, pour les plus anciennes, dans [13].

[8] Ces bandes de fréquences et ces pondérations sont définies par l'utilisateur. On peut penser qu'une analyse peut souhaiter pondérer plus lourdement les setup possédant une majorité de fréquences très graves ou très aiguës, par exemple, à cause de leur gros impact sur la complexité sonore global.

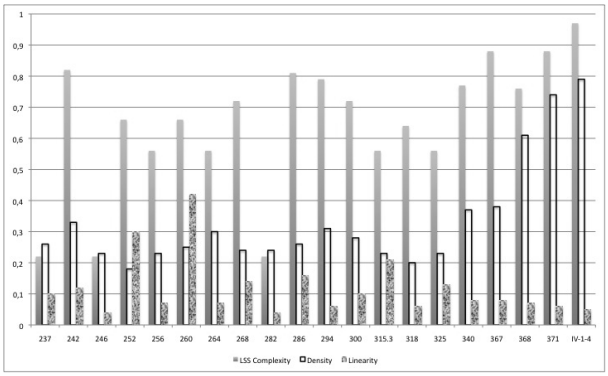


Figure 8. Une analyse de l'extrait de Beethoven par deux descripteurs de *SOAL*.

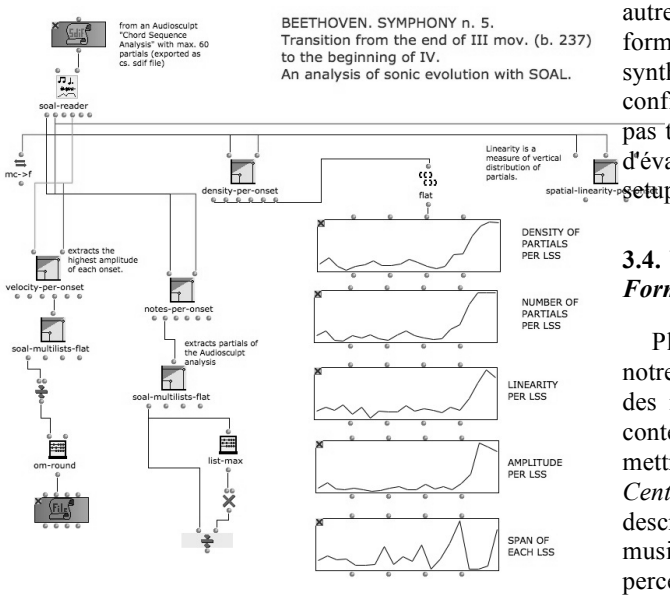


Figure 9. Patch *OpenMusic* montrant l'analyse de Beethoven avec *SOAL*.

3.3. Un réseau de descripteurs psycho-acoustiques

Outre les descripteurs d'ordre statistique énumérés ci-dessus, il est fondamental d'aborder également l'analyse de l'impact de l'orchestration à partir d'un où plusieurs descripteurs psycho-acoustiques. Comme l'a déjà bien montré Philippe Lalitte dans son analyse de la *Sequenza VII* de Berio, « l'extraction d'informations sub-symboliques opérée par les descripteurs psychoacoustiques offre la possibilité d'approcher la perception de plusieurs dimensions musicales telles que le temps […], la hauteur (rugosité, saillance, […], etc.) et le timbre (brillance, inharmonicité, enveloppe, clarté spectrale, etc.) » ([24]:116). Une des bibliothèques la plus connue et peut-être la plus ample est celle développée par l'Université Queen Mary de Londres, pour le logiciel *Sonic Vizualizer* [6][9]. Un grand nombre d'entre eux peut se montrer adéquat pour mettre en évidence le domaine du timbre que l'orchestration du compositeur affecte le plus, ou celui qui offre le plus de saillance à la perception. Le problème réside en fait dans la quantité de descripteurs disponibles, et l'hétérogénéité des informations produites. En effet, il important de garder à l'esprit que le concept de base d'un descripteur est d'« aplatir » un composant multi-dimensionnel, le timbre, sur une seule dimension. En d'autres termes, chacun opère une réduction orientée du signal, en fonction de l'information qu'il est censé retourner. Par conséquent, aucun n'est capable, tout seul, de fournir l'ensemble des informations que nous souhaitons. L'idée est donc de constituer un réseau modulable de descripteurs auxquels il serait fait appel en fonction des contextes, et normaliser leur résultats sur le même vecteur de complexité relative auquel ont été soumis les autres composants du modèle. L'avantage de cette formatation est qu'elle peut produire une vision plus synthétique, plus hollistique, du comportement des configurations sonores sur le temps. En effet, il ne s'agit pas tant de qualifier les setup dans l'absolu, mais plutôt d'évaluer le degré de différence, ou distance, entre un setup et l'autre, dans un domaine donné.

3.4. Un descripteur psycho-acoustique experimental: *Formantic Centroid*

Plusieurs descripteurs ont déjà été, ou sont, testés par notre groupe de recherche sur plusieurs œuvres, avec des résultats dont la pertinence varie em fonction du contexte musical, et, surtout, de ce que l'on cherche à mettre en évidence. Un des plus connus est la *Spectral Centroid*. De nombreuses études ont déjà qualifié ce descripteur comme étant un bon outil pour l'analyse musicale, étant donné sa forte corrélation avec la perception de la brillance du son (cf. [23, 25, 26]). Comme on le sait, il indique où se situe le « centre de gravité » du spectre. Une manière simple de le décrire consiste à dire que la centroide spectrale est le résultat de la moyenne des fréquences d'un spectre, de la plus grave à la plus aiguë, pondérée par leurs amplitudes respectives..

Le problème de cette formule est que cette moyenne arithmétique peut définir un point sans existence physique réelle dans l'objet observé. Il est possible aussi que par ce calcul, malgré la pondération exercée par les amplitudes, la toujours grande quantité de fréquences aiguës biaise un peu la réalité perceptive de l'objet sonore, en poussant irréellement les moyennes vers le haut du spectre. Il semble donc permis de penser qu'une sélection préalable des fréquences les plus proéminentes pourrait engendrer une corrélation plus fine avec la perception. Or, les fréquences les plus saillantes d'un spectre se regroupent en ce que l'on appelle des *formants*, qui sont le résultat de pics d'énergie dans une région plus ou moins étroite du spectre [8]. D'où il semble logique qu'un calcul de la centroïde des formants puisse mieux décrire la brillance de certains complexes sonores. Nous avons donc élaboré une fonction qui opère ce calcul, *Formantic Centroid*, que nous avons

[9] Mais voir aussi [1, 3], entre autres.

implementée dans *OpenMusic* (Figure 10) de manière à faciliter son inclusion dans le modèle d'analyse de l'orchestration[10]. Cette inclusion implique entre autres une normalisation sur le vecteur de complexité relative, pour laquelle il est nécessaire de fixer une valeur paradigmatique maximum. Pour celle-ci, j'ai adopté la Formant Centroid d'un bruit blanc (10.922,781 Hz), que l'on peut considérer comme le son le plus complexe dans ce domaine. On obtient alors, par factorisation, la *Relative Formant Centroid* (RFC) de chaque setup.

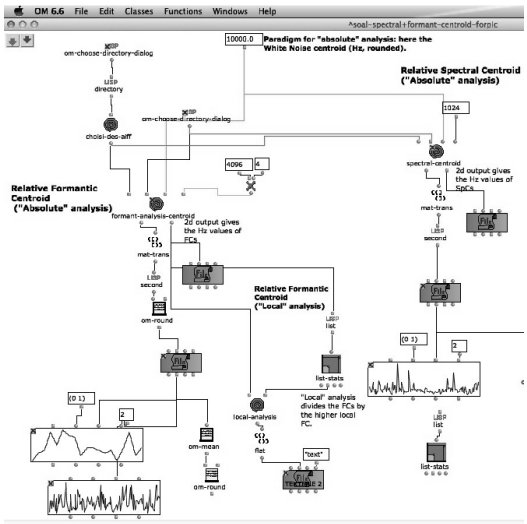


Figure 10. Le patch *OpenMusic* pour l'analyse comparée de la *Spectral Centroid* et de la *Formantic Centroid*. *Absolute Analysis* correspond à la normalisation des valeurs sur une échelle où celle du bruit blanc est le maximum. *Local Analysis* prend comme paradigme l'unité locale possédant la plus haute valeur.

RFC doit encore passer par une batterie de tests, y compris en comparaison avec la centroïde spectrale conventionnelle. Au cas où elle s'avère réellement fonctionnelle dans certains contextes, sa description détaillée fera l'objet d'une publication en temps opportun.

Pour l'heure, la figure 11 montre l'évolution de la RFC pour le même extrait de Beethoven, mise en relation avec celle obtenue par le calcul de la *Relative Spectral Centroid* (c'est-à-dire le calcul standard calibré sur l'échelle de complexité relative à partir du bruit blanc), ce qui met permet de constater que les deux descripteurs retournent des évaluations différentes. Dans la plupart des cas (les exceptions sont les LSS 367 et 368), RFC retourne des pondérations inférieures au calcul standard, ce qui signifie qu'elle tend à rebaisser la centroide vers des fréquences plus graves[11]. Il reste donc à étudier si cette rectification traduit mieux la réalité de la perception des timbres orchestraux.

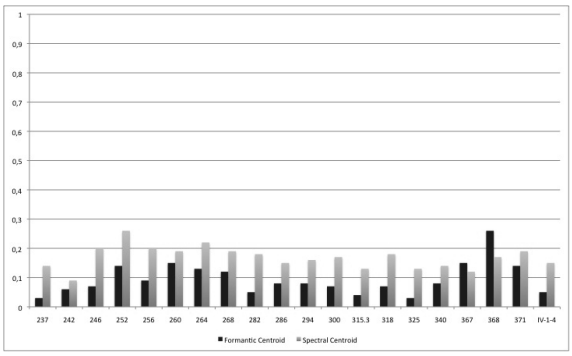


Figure 11. Le même extrait de Beethoven analysé par deux formules de calcul de la centroide.

4. CONCLUSION

Cet exposé a montré l'ossature d'un modèle pour l'analyse de l'orchestration. Celle-ci constitue une sorte de guide pour le développement, non seulement d'une méthode et de ses outils informatiques d'appui, mais aussi, simultanément, d'une théorie. S'agissant d'une dimension encore peu formalisée dans le domaine de l'analyse musicale, j'ai choisi une méthode empirique, qui consiste à tester systématiquement les composantes du modèle sur des œuvres tenues pour exemplaires, pour juger de leur éventuelle validité. Le but est de parvenir le plus tôt possible à un modèle assez cohérent – encore que toujours perfectible, bien évidemment – pour produire des résultats musicologiquement pertinents sur certains corpus d'œuvres, au sujet desquelles on sait le rôle fondamental de l'orchestration sans pour autant être en mesure de l'expliciter plus rigoureusement. Illustrations, figures et tableaux

5. REFERENCES

[1] Bogdanov, D. *et al*. « Essentia: An Audio Analysis Library For Music Information Retrieval » , *Proceedings of 14th International Society for Music Information Retrieval Conference,* Curitiba, ISMIR 2013, p. 493-498.

[2] Berry, W. *Structural functions in music,* Dover, New York, 1987.

[3] Cabrera, D., *et al.* « PsySound3: an integrated environment for the analysis of sound recordings » , *Acoustics 2008: Proceedings of the Australian Acoustical Society Conference*, Geelong, 2008.

[4] Carpentier, G. "Aide logicielle à l'orchestration: Un état des lieux". Musimediane, 2011, n. 6. http://www.musimediane.com/spip.php?article134.

[5] Cogan, R. *New Images of Musical Sound*, Harvard University Press, Cambridge, 1984.

[10] Je remercie Mikhail Malt pour sa contribution dans le développement de cette fonction.

[11] Puisque le paradigme de complexité maximum est le bruit blanc à 10.922 Hz (valeur (1,0) de l'échelle), la hauteur des histogrammes indique donc la hauteur relative de la fréquence de la centroide.

[6] Cook, N.; Leech-Wilkinson, D. *A Musicologist's guide to Sonic Vizualiser,* King's College, London, 2009.

[7] Dalbavie, M.A. « Pour sortir de l'avant-garde » , *Le timbre, métaphore pour la composition* (Barrière, J.B., org.), Christian Bourgois, Paris, 1991, p. 303-334.

[8] Diatkine, C. *Audiosculpt 3.0 User Manual,* IRCAM, Paris, 2011. http://support.ircam.fr/docs/AudioSculpt/3.0/co/AudioSculptguideWeb.html.

[9] Dolan, E. I. *The Orchestral Revolution. Haydn and the Technologies of Timbre*, Cambridge University Press, Cambridge, 2013.

[10] Erickson, R. *Sound structure in music,* University of Chicago Press, Berkeley,1975.

[11] Gentil-Nunes, P. *Análise particional: uma mediação entre composição musical e a teoria das partições,* Thèse de Doctorat. Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2009.

[12] Goubault, C. *Histoire de l'instrumentation et de l'orchestration*, Minerve, Paris, 2009.

[13] Guigue, D. *Une Etude 'pour les Sonorités Opposées' – Pour une analyse orientée objets de l'oeuvre pour piano de Debussy et de la musique du XXe siècle,* Atelier National de Reproduction des Thèses, Lille, 1997.

[14] _____ , *et al.* « SOAL For Music Analysis : A Study Case With Berio's Sequenza IV », *Actes des JIM05 — Journées d'Informatique Musicale*, CICM, Paris, 2005. http://jim2005.mshparisnord.org/articles.htm.

[15] _____ *Esthétique de la Sonorité. L'héritage de Debussy dans la musique pour piano du XX^e siècle*, L'Harmattan, Paris, 2009.

[16] _____ *Sonic Object Analysis Library – OpenMusic Tools For Analyzing Musical Objects Structure*. Software Documentation. *Mus^3*, João Pessoa, 2010. http://www.ccta.ufpb.br/Mus3.est

[17] ____ « Diário de bordo. Webern, Variações para orquestra op. 30 », I*V. Seminário Ciência Música Tecnologia: Fronteiras e Rupturas,* São Paulo, ECA/USP, 2012. http://www2.eca.usp.br/smct/ojs/index.php/smct/issue/view/6 (visité en 07/2013).

[18] _____ . « Une méthode d'analyse de l'orchestration: rapport de recherche appliquée aux Variations op. 30 de Webern », *Actes des Journées d'Analyse Musicale*, SociétéFrançais d'Analyse Musicale, Rennes, 2013 (sous presse).

[19] Hummel, T. *conTimbre.* Software documentation. 2014. http://www.contimbre.com/index.php?option=com_content&view=article&id=51&Itemid=82&lang=en.

[20] Kaltenecker, M. « Confiture de neige. Le bruit dans la musique du 19^e siècle, entre discours esthétique et pratique instrumentale » , *Analyse Musicale*, nº 56, p. 60=72, 2007.

[21] Kim, J. *Représentation et analyse musicale assistée par base de données relationnelle de la partition des* Variations pour orchestre *op. 30 d'Anton Webern*. Thèse de Doctorat, Université de Paris-IV-Sorbonne, Paris, 2007.

[22] Koechlin, Ch. *Traité de l'Orchestration*, Max Eschig, Paris,1954.

[23] Lalitte, P. « Densité 21,5 de Varèse : un condensé d'harmonie-timbre » in: Horodyski, T. & Lalitte, P. *Edgard Varèse. Du son organisé aux arts audio*, L'Harmattan, Paris, 2008, p. 245-277.

[24] _____ . « Du son au sens : vers une approche sub-symbolique de l'analyse musicale assistée par ordinateur. » *Musurgia*, Vol. XVIII, p. 100.116, 2011.

[25] Maia, I. L.; Schaub, S. « Analyzing textural progressions in Iannis Xenakis's Aroura (1971) for twelve strings - Some Preliminary Results. », *XXII Congresso da Associação Nacional de Pesquisa e Pós-Graduação em Música (ANPPOM)*, João Pessoa, ANPPOM, Vol. 1, p. 496-503, 2012.

[26] Malt. M., & Jourdan, E. « Le « BStD – une representation graphique de la brillance et de l'ecart type spectral, comme possible representation de l'evolution du timbre sonore. » in: Hashner, X., & Ayari, M. *L'analyse musicale aujourd'hui, crise ou (r)évolution?* Université de Strasbourg/SFAM, Strasbourg, 2009 (document communiqué par les auteurs).

STRUCTURES MODALES ET ORNEMENTATIONS DANS LES MUSIQUES ARABES : MODELISATIONS ET PRESENTATION D'UNE BIBLIOTHEQUE D'OBJETS DANS CSOUND

Raed BELHASSEN
CICM- Centre de recherche en Informatique et Création Musicale -EA 1572
Université Paris 8
MSH Paris Nord
brmusique80@yahoo.fr

RÉSUMÉ

Dans la perspective de création d'une bibliothèque d'objets dédiée à la musique arabe dans l'environnement Csound, nous exposons tout d'abord quelques éléments intrinsèques idiosyncratiques des langages musicaux concernés. La physionomie homophonique est esquissée et la structure modale basée sur le *maqâm* est détaillée. Nous abordons la causalité des échelles intervalliques grâce à une approche historico-acoustique afin de dégager les composantes élémentaires. Les attitudes compositionnelles et interprétatives musicales sont analysées, en particulier, les ornementations et les subtilités d'exécution.
En second lieu et dans l'optique de repenser l'espace composable dans ce contexte, la notion de modèle à but de création est expliquée et une modélisation informatique dans Csound est proposée.
Les difficultés liées à la composition assistée par ordinateur, en ce qui concerne la musique arabe, sont soulignées et une solution est proposée : une bibliothèque d'objets dans Csound. Ses unités intrinsèques : UDO (User Defined Opcode), table de fonction, Opcodes, boucles et conditions sont exposées. Nous analysons son fonctionnement, les stratégies opératoires employées ainsi que ses techniques d'acheminement. Enfin, nous évoquons les perspectives futures.

1. INTRODUCTION

Composer la musique arabe via les nouvelles technologies soulève de nombreuses problématiques intrinsèques. En effet, prendre en compte les idiosyncrasies des langages musicaux concernés, entre le caractère polymorphe de l'interprétation musicale, l'aspect homophonique et le langage modal inféodé à un système intervallique, est substantiel dans une telle perspective.
Il est évident que travailler sur une telle approche, peu développée jusqu'ici dans le domaine de l'informatique musicale[1], implique une certaine démarche de modélisation à double niveau : modélisation à des fins d'analyses et modélisation dans le but de recréer et de reproduire. Ainsi, la notion de modèle nous permet d' « explorer certaines propriétés des composantes abstraites dégagées par un travail d'analyse » et « sur un plan plus abstrait, il (le modèle) opère à des fins d'interprétation ou de comparaison » [7].
Notre motivation première consiste, non seulement à modéliser quelques idiomes des musiques arabes à des fins d'analyses, mais notre travail vise aussi les possibilités de synthèse, de création et d'émulation.
Nous tenterons dans un premier lieu de dégager les aspects formels inhérents aux musiques arabes, ensuite nous détaillerons le processus opératoire et les méthodes employées dans le cadre d'une modélisation informatique et musicale à la fois. Nous présenterons une bibliothèque d'objets dans l'environnement Csound, basée sur des UDO et des tables de fonction, tout en détaillant le fonctionnement de ses composantes intrinsèques.

2. LES MUSIQUES ARABES IMPROVISEES : PROPRIETES MUSICALES ET ESTHETIQUES INHERENTES

Afin de cerner explicitement notre objet d'étude, nous proposons, dans cette section, de désagréger les éléments intrinsèques aux musiques arabes[2]. Cette phase d'analyse est fondamentale dans la démarche de modélisation : les idiomes analysés dans cette section serviront dans la création des modèles informatiques par la suite.

[1] Notre démarche de modélisation consiste à prendre en compte certains idiomes des musiques arabes dans une seule bibliothèque sous Csound : l'aspect homophonique, les systèmes intervalliques tempérés ou non, les maqâms et les subtilités d'exécution. Cette modélisation nous permet de composer les musiques arabes. Nous citons à titre d'exemple quelques travaux de modélisation informatique à but d'analyse ou/et de génération de musique tels que les travaux d'Olivier Lartillot, Mondher Ayari, Gérard Assayag ou encore les travaux de Marc Chemillier. D'autres travaux d'implémentations des échelles microtonales sont intéressants, nous citons par exemple : le logiciel Sequenza ou encore l'extension Omicron dans OpenMusic. (Liste non exhaustive).

[2] Ne desservant pas notre étude, nous avons volontairement fait le choix d'exclure l'étude du système rythmique et les relations musique/poésie/forme vocale.

2.1. Structure modale des langages musicaux arabes

Les langages musicaux arabes sont caractérisés par leurs aspects monodique, monophonique et hétérophonique, « régis par le système modal arabe » [10].

2.1.1. Le maqâm

Le terme mode nous renvoie vers le mot *maqâm*, ce qui signifie lieu en arabe, devenu au XVIII[e] siècle la signification d'un emplacement sur le manche d'un instrument de musique et il désigne une succession d'intervalles. Poché définit le *maqâm* comme « une série d'intervalles qui, selon les cultures, seront regroupés en genre ou en famille (*jins*) » [28].

2.1.2. Le jins

Genre en français, le terme *jins* (pluriel-*ajnas*) désigne une distribution de cellules mélodiques, intrinsèques au *maqâms*, en tricorde, tétracorde et pentacorde[3] [28], exécutés selon un enchainement d'intervalles donné.

A titre d'exemple, nous présentons le *maqâm Bayati*, ses cellules mélodiques endogènes ainsi que ses variantes (Figure 1). Le genre *Bayati* en Ré est une cellule fondamentale qui sert de point de retour à chaque exploration modale. Lorsque la deuxième cellule est un *jins Hijaz*, le mode change d'aspect et s'appellera désormais *Bayati Churi*.

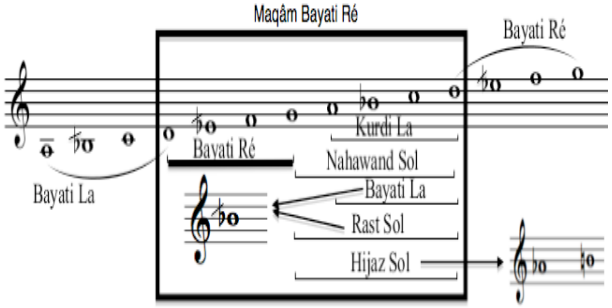


Figure 1. Le *maqâm Bayati*, ses *jins* et ses variantes.

2.2. Intervalles musicaux : l'héritage musical arabe

La concomitance des pratiques musicales et de l'échelle intervallique arabes nous conduit à esquisser rapidement les grandes lignes de son évolution en adoptant une approche acoustique/historique.

2.2.1. Intervalles musicaux jusqu'au XIII[e] siècle

Les théoriciens arabes ont défini les intervalles en fonction de l'emplacement des doigts sur le manche des instruments. Nous ne possédons pas de documents écrits antérieurs à ceux d'Al Farabi[4] [13]. Nonobstant, nous savons que jusqu'au X[e] siècle, il existe bel et bien au moins deux systèmes acoustiques différents : « Le premier repose sur une division de l'échelle de deux octaves en quarante parties égales (…), tandis que le deuxième système comporte les intervalles de limma, de ton majeur, de tierce mineure, de tierce majeure, de quarte juste et de quinte juste » (Figure 3) [10]. Les ligatures sur le manche de l'instrument nous donnent une indication sur les intervalles musicaux, Al kawarezmi [6] nous informe que la première ligature (ligature de l'ancien médius) se place environ autour du quart entre la ligature de l'index et celle de l'annulaire, la deuxième est celle du médius perse et se place en leur milieu environ et la troisième est celle de Zalzal[5], aux trois quart entre les deux ligatures à peu près. L'assimilation de la musique arabe à l'intervalle qui correspond au médius de Zalzal, appelé tierce neutre, est devenu « symbole de la différence entre la théorie musicale occidentale et la musique arabe » [25], en repoussant « le dualisme traditionnel de tierces majeures et mineures » [14], un intervalle exprimé en rapport numérique à (27/22) (Figure 3) [10]. Au XIII[e] siècle, Saffiyu al-Din Al Urmawi écrit son livre *Kitab al-Adwar-Le livre des cycles,* qui « se fonde sur un système basé sur le cycle des quintes, qui était déjà connu. Mais son organisation des intervalles et des cycles, en donnant leur nom utilisé dans la pratique, en fait un ouvrage exceptionnel » [25].

Les écrits de Saffiyu al-Din nous donnent une idée précise des « cycles » de son époque avec une approche certes globale mais explicite, des modes qui prennent « appui sur les cellules mélodiques basiques que sont les genres tétracordaux et pentacordaux, à partir desquels se construisent des structures modales (heptacordales et octacordales) plus complexes » [2]. Nous représentons ci-dessous un exemple de cycle musical décrit par Al Urmawi ainsi que ses intervalles en rapports numériques [9-23] (Figure 2).

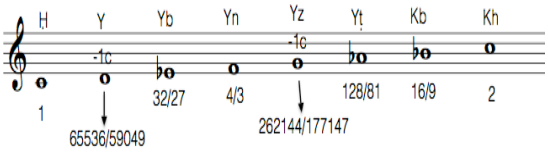


Figure 2. Le mode Husayni – en haut la présentation d'intervalles par Saffiyu al-Din, en bas la représentation en rapports numériques.

[3] Un *maqâm* (*tba'* en Tunisie) peut être composé de plusieurs *jins* créant des regroupements qui peuvent être assemblés, désassemblés ou imbriqués [15].

[4] Philosophe, musicien et musicologue du X[e] siècle ; il est également l'auteur du livre « *Kitab al Musiqa al Kabir* » (Le grand livre de la musique).

[5] Musicien et instrumentiste très doué de la cours abbaside à la fin du VIII[e] siècle.

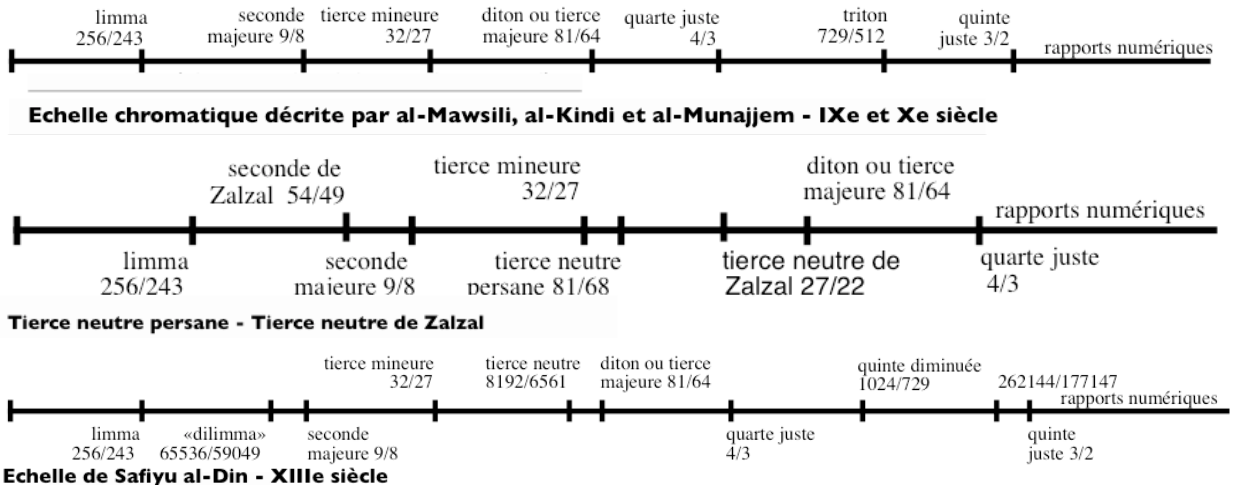


Figure 3. Les échelles intervalliques respectives du IX^e^/X^e^ siècles, tierce persane/tierce neutre de Zalzal et de Safiyu al-Din-XIII. [10]

2.2.2. La gamme arabe dans la conjoncture musicale actuelle

La musique arabe a subi une transformation spectaculaire, résultat d'un phénomène acculturatif[6] allochtone et exogène, ou indigène même, due aux différentes pratiques musicales imprégnées des répertoires divers tout en gardant leur essence, leur lien avec la tradition [1–16]. Les idiosyncrasies musicales autochtones dans le monde arabo-irano-turque au début du XX^e^ siècle ont créé un véritable besoin identitaire. Le congrès de musique du Caire de 1932 a engendré une véritable dichotomie esthétique entre la musique traditionnelle et les pratiques nouvelles, y compris l'adoption d'une échelle tempérée de 24 quarts de tons [27].

2.3. Attitudes compositionnelles et interprétatives substantielles

2.3.1. L'improvisation instrumentale

Grâce à un schéma préétabli basé sur un matériau modal, l'instrumentiste explore les aspects formels du maqâm en suivant un tracé hiérarchique s'inscrivant dans une certaine temporalité dynamique avec un déploiement mélodico-rythmique instantané. Ayari [10] écrit à propos du *taqsim* (improvisation instrumentale explorant le maqâm en exploitant beaucoup de possibilités techniques) : « En tenant compte du grand éventail de possibilités que propose la tradition, l'interprète prévoit les conduites mélodiques et organise subtilement l'évolution des progressions temporelles (…), l'interprète est libre de privilégier certaines déterminations mélodiques (…) la variété de ces patrons mélodico-rythmiques caractérise le style propre de chaque interprète, mais elle dépend surtout de ses capacités d'improvisation ». Cette liberté dans l'interprétation est l'essence même de la musique arabe : l'instrumentiste, non seulement, explore un espace sonore modal donné, mais il suit un processus opératoire personnel formulant des variations de texture sonore, de timbre, d'intensité et d'ornementations[7].

2.3.2. Les ornementations

Peu d'études s'intéressent aux ornementations dans les musiques arabes[8], pourtant, elles sont considérées comme étant un « procédé de composition et non un artifice d'exécution »[9] [29]. AL Faruqi [5] écrit à propos : « Pour l'artiste arabe, l'ornementation n'est donc pas un additif, un élément superflu ou amovible de son art. C'est le matériau même à partir duquel il façonne des formes à l'infini (…). L'ornementation peut comporter des additions mélodiques (…). Une grande partie de ces additions sont comparables à celles que l'on rencontre dans la musique occidentale (fioritures, trilles, mordants, etc) ». A partir de ce constat, nous savons qu'une broderie est au cœur de plusieurs ornements, tel qu'un trille qui est une suite de broderies rapides, le mordant correspond à une broderie supérieure ou inférieure ainsi

[6] Sur le plan musical, nous avons opté pour le terme acculturation au sein d'une même culture arabo-islamique générale, où les répertoires artistiques se diffèrent selon les pratiques musicales locales [19].

[7] Abou Mrad distingue trois types d'improvisations : monomodulaire, plurimodulaire à caractère cantillatoire et plurimodulaire d'interpolation. Nous citons à propos des ornementations : « l'improvisation monomodulaire consiste en la formulation de variations autour d'un phrasé précomposé et mesuré. Ces variations sont le plus souvent ornementales » [1].

[8] Nous pouvons citer à titre d'exemple le travail de Zouari dans sa thèse de doctorat où elle propose une notation basée sur une symbolisation des ornementations en musique arabe en passant par une phase d'analyse d'un rendu sonore (enregistrement sonore de la troupe tunisienne pendant le congrès du Caire en 1932). [31]

[9] Viret souligne le rôle fondamental que les ornementations jouent en grégorien comme dans toutes les musiques modales. [29]

que le gruppetto qui est une double broderie [3], nous proposons de les examiner en détails (Figure 4) [31].

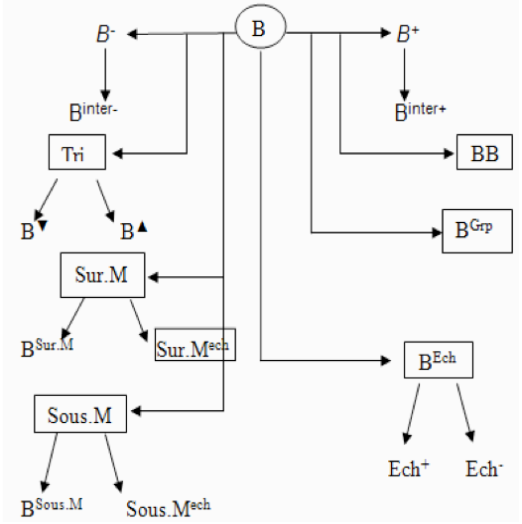


Figure 4. Arbre des ornements issus d'une broderie.

2.3.2.1 Multiplicité formelle des ornementations :

Revenons à la citation d'AL Faruqi ci-dessus concernant le multi façonnage formel, nous nous posons cette question : est-t-il possible, dans une perspective de modélisation, d'inscrire une démarche taxinomique concernant les ornementations en musique arabe ?

Pour répondre à cette question, nous distinguons deux types d'ornementations, baptisés *micro-ornement* et *macro-ornement*.

Micro-ornements

Ce sont les ornementations de base très semblables à celles de la musique occidentale. Ci-dessous une sélection des ornements qui concernent notre étude (Tableau 1) [3-12-24-31][10].

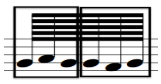
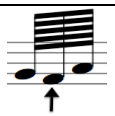
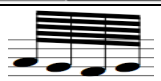
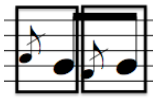
Nom de l'ornement	Sa réalisation
Broderies	
Doubles broderies	
Trilles	
Trémolo	
Echappé	
Sur-marche et Sous-marche	
Gruppetto	
Appoggiatures	

Tableau 1. Les micro-ornementations.

Macro-ornements

Les macro-ornements sont le fruit de congruences, d'imbrications et de conglomérations motiviques des micro-ornements. Une infinité de possibilités est offerte à un interprète dans une démarche d'exécution musicale, avec des improvisations « sous-jacentes »[11], inféodées à son expérience, son vécu, ses connaissances, s'inscrivant dans un processus d'actualisation d'un matériau musical[12] en perpétuelle transformation. Non seulement ce matériau musical arabe inextricable concerne les aspects modaux et les intervalles, mais aussi la manière d'ornementer, avec des micro-ornements, ou plus complexe encore, des macro-ornements. Ci-dessous, un exemple de macro-ornement avec différentes possibilités d'analyses (Figure 5).

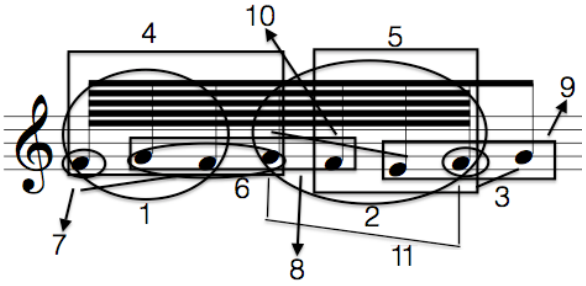


Figure 5. Un macro-ornement : 1-Broderie, 2-Gruppetto, 3-Appogiature, 4-Trille, 5-Broderie, 6-Broderie, 7-Appogiature, 8-Trille, 9-Petite Sous-marche, 10-Petite Sur-marche et 11-Double broderie.

[10] Nous engageons ici une approche adaptée à la musique arabe grâce à une recherche sélective dans les quatre références citées ci dessus. Nous opterons pour les mêmes définitions que celles définies par Féron concernant le trémolo : « fluctuations régulières de l'amplitude sonore », le trille : « alternance entre deux hauteurs distinctes ». En ce qui concerne la Sur-marche et la Sous-marche, nous empruntons ces termes à Zouari, inspirés par l'analyse schenkérienne [17].

[11] Le terme « improvisation » ou « improvisé » dans la musique arabe nous renvoie vers deux significations qui méritent d'être soulignées : l'improvisation au sens conventionnel du terme, et d'un autre coté au sens inhérent à une musique de tradition orale, où chaque réinterprétation d'une même œuvre possède des caractéristiques différentes, connue comme une sorte de liberté d'interprétation propre à chacun, qui dépend d'un schéma cognitif inextricable. Nous l'appellerons « improvisation sous-jacente ».

[12] Pour plus d'information concernant la notion d'actualisation d'un matériau musical pendant une interprétation/improvisation, son inscription dans une directionalité/temporalité, voir [21].

2.3.3. Subtilités interprétatives substantielles diverses

En plus des ornementations, les instrumentistes arabes produisent des variations très semblables aux techniques d'interprétation musicale occidentale : nuances, phrasés, intonations, caractères, articulations etc. Dans notre démarche de modélisation, nous mettrons l'accent sur le vibrato, le glissando, le legato et les effets liés aux variations de l'intensité. Ci-après nous visualisons les spectres respectifs de l'enregistrement de l'orchestre tunisien au congrès du Caire en 1932 (Qacida tab' Husseini) et une improvisation réalisée par un violoniste égyptien[13]. Le premier s'inscrit dans une perspective traditionnaliste et le deuxième, il s'agit d'une improvisation moderne : la saillance des variations de hauteurs est frappante dans les deux exemples (Figure 6).

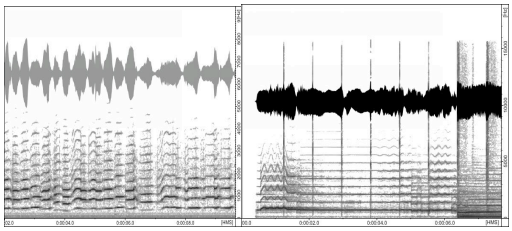


Figure 6. Représentations spectrales de deux exemples sonores : ancien et moderne.

Nous remarquons facilement la présence des glissandos dans le premier exemple et des vibratos à plusieurs reprises dans le deuxième : ceci démontre formellement que l'exécution musicale arabe, qu'elle s'inscrive dans une optique musicale moderniste ou conservatrice, ne dépend pas seulement de la structure modale et des ornementations, mais aussi des techniques de jeux multiples propres à chaque instrumentiste.

3. MODELISATION INFORMATIQUE : UNE BIBLIOTHEQUE D'OBJETS DANS CSOUND

Arom [8] définit le modèle comme « une représentation à la fois globale et simplifiée de l'objet », il ajoute, « dans les musiques de tradition orale, la modélisation peut être obtenue, soit par l'émergence – c'est-à-dire la matérialisation – du modèle, soit par une série de déductions successives ». Un modèle ontologique ou une ontologie informatique est un outil de représentation d'un ensemble de connaissances ou de concepts sous une forme utilisable par un ordinateur[14]. Le modèle que nous présentons dans cette section nous permettra non seulement de comprendre les mécanismes de fonctionnement du système musical arabe, mais aussi de le reproduire via Csound en s'inscrivant dans une perspective d'analyse/synthèse voire même émulation[15].

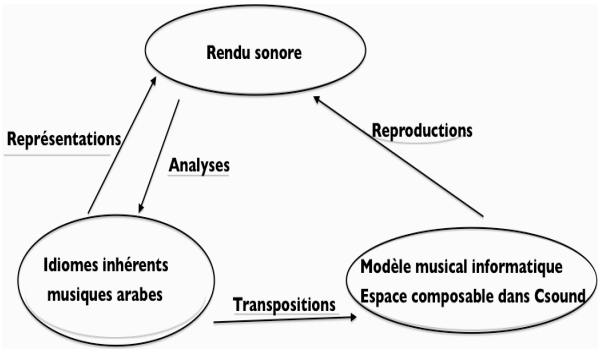


Figure 7. Modèle informatique dans Csound.

Notre démarche de modélisation à des fins de construction d'un espace composable en ce qui concerne la musique arabe[16], s'engage dans une perspective, d'abord analytique, empirique, d'un point de vue musicologique/ethnomusicologique (première section de cet article), basée sur l'observation pour arriver à une traduction formalisée, ensuite dans une logique de synthèse, de création et d'émulation. En effet, une simplification globale pendant la représentation du modèle musical, la plus fidèle possible, permet une caractérisation significative qui prend en compte plusieurs aspects inclus dans le système musical modal arabe complexe, y compris les ornementations et les subtilités d'interprétation.

3.1. Prendre en compte l'aspect homophonique de la musique dans Csound

La création sonore dans Csound passe par deux phases : à partir des instruments de l'orchestre, générés par des codes d'opérations (*Opcodes*), et suivant des événements dans une partition appelée score. Afin d'illustrer le plus explicitement possible notre démarche, nous avons choisi de travailler avec l'Opcode *pluck* qui produit un son de corde pincée à décroissance naturelle, basé sur l'algorithme de *Karplus-Strong*, pour ses qualités dissipatives, utiles pour une représentation formelle du rendu sonore des musiques arabes. Prendre en compte le caractère homophonique d'une musique, revient à créer des événements successifs[17], s'inscrivant

[13] Mahmoud Serour, premier prix du meilleur violoniste du monde arabe en 1996.

[14] Gruber définit l'ontologie comme une spécification explicite de la conceptualisation d'un domaine : « An ontology is an explicit specification of a conceptualization » [26].

[15] Nous pouvons non seulement envisager la simulation de l'interprétation musicale arabe, mais aussi la possibilité d'une émulation au sens proposé par Berthoz [18], dans un but de recréer ou de transformer. La notion d'émulation est reprise et adaptée par le groupe de travail AFIM -visualisation du son, dans un sens artistique : « émulation artistique du son ». Voir le rapport d'activité proposé par Anne Sedes au (consulté le 5 janvier 2014) : gtv.mshparisnord.org/IMG/pdf/rapportGTVisualisation.pdf

[16] L'espace composable en tant que procédures opératoires, l'ensemble de traitements et de variables définis auparavant et avec lesquels nous composons. Voir [20].

[17] En ce qui concerne Csound, nous entendrons par événement tout déclenchement d'un son.

dans une perspective d'écriture horizontale. Nous détaillerons ce point ci-après.

3.2. Modéliser l'aspect musical modal dans Csound

A l'inverse de la première section de cet article, ci-après nous détaillerons la stratégie opératoire utilisée dans Csound pour modéliser les maqâms en suivant une logique de reconstruction.

3.2.1. Echelles et tempéraments

3.2.1.1 Tables de fonctions et maqâms

Nous proposons de créer une table de fonction à l'entête de l'orchestre[18] :

```
giRast13 ftgen 114, 0, 8, -2, 9.00, irastsib,
irastladb, irastsol, irastfa, irastmidb,
irastre, 8.00
```

L'Opcode *ftgen* nous permet de créer une table, référencée par un numéro choisi, que nous nommerons en fonction du maqâm concerné, et qui nous permet de stocker huit classes de hauteurs correspondantes aux notes musicales d'une octave (en l'occurrence, pour cet exemple il s'agit du maqâm Rast, n° 114, avec huit valeurs de classes de hauteurs correspondantes à des notes musicales) (Figure 8).

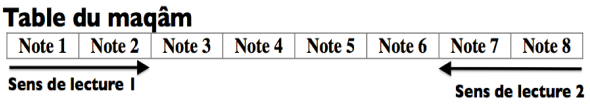


Figure 8. Une table de fonction avec deux possibilités d'extraction de données.

3.2.1.2 Classes de hauteurs et intervalles musicaux

Les hauteurs peuvent être exprimées dans Csound en fréquences ou en classe de hauteurs. Une classe de hauteur est composée d'un nombre entier qui correspond à l'octave et d'une partie décimale qui représente les douze notes musicales en tempérament égal (en fonction de l'Opcode utilisé). Par exemple, pour un Do3, la classe de hauteur est 8.00, pour un Ré#3 : 8.03, un Mi : 8.04 ou un Do4 : 9.00. Pour reproduire un quart de ton tempéré sur la note Mi par exemple, il faut la baisser de la moitié : 8.035.
Sur une échelle non tempérée, il faut convertir la classe de hauteur d'une note fondamentale en fréquence, la multiplier par le rapport numérique qui détermine l'intervalle, ensuite la reconvertir en classe de hauteur[19].

Voici le code d'un exemple parlant du maqâm Husayni basé sur l'échelle du Saffiyu al-Din (Figure 2) :

```
; maqam Husayni
ihusaynisib = pchoct(octcps(cpspch(8.00) *
(16/9)))
ihusaynilab = pchoct(octcps(cpspch(8.00) *
(128/81)))
ihusaynisold = pchoct(octcps(cpspch(8.00) *
(262144/177147)))
ihusaynifa = pchoct(octcps(cpspch(8.00) *
(4/3)))
ihusaynimib = pchoct(octcps(cpspch(8.00) *
(32/27)))
ihusaynired = pchoct(octcps(cpspch(8.00) *
(65536/59049)))
giHusayni13 ftgen 116, 0, 8, -2, 9.00,
ihusaynisib, ihusaynilab, ihusaynisold,
ihusaynifa, ihusaynimib, ihusaynired, 8.00
```

3.3. Modéliser l'interprétation musicale arabe dans Csound

Dans cette même logique de reconstruction, les aspects liés à l'interprétation musicale arabe doivent être sérieusement pris en compte dans le processus compositionnel sous Csound.

3.3.1. La nécessité d'une adaptation

Composer la musique arabe dans Csound soulève plusieurs problématiques liées, en grande partie, à la question de l'ornementation : comment ornementer une phrase mélodique dans une composition ? Quel processus d'intégration des macro-ornements serait envisageable ?
Nous reprenons l'exemple d'un macro-ornement, schématisé plus haut (Figure 5), voici le code sous Csound, évidemment, dans la partie score :

```
i1        1       .07             8.05
i1        +       .                 8.07
i1        +       .                 8.05
i1        +       .                 8.07
i1        +       .                 8.05
i1        +       .                 8.035
i1        +       .                 8.05
i1        +      .51              8.07
```

L'instrument 1 (i1) doit jouer le son de la fréquence qui correspond à la classe de hauteur 8.05 à l'instant 1 pendant 0,07 seconde, ensuite la fréquence qui correspond à 8.07 à l'instant 1+0,07 et ainsi de suite[20]. Le macro-ornement, mis à part son rôle esthétique, dure une seconde et consiste à appliquer une fluctuation de hauteur en dessous de la note finale. Voici le code sans ornementations :

```
i1        1       1                 8.07
```

Imaginons pendant l'acte de composition d'une musique riche en ornements, la tache fastidieuse à cause des nombreuses contraintes émergentes : multiplicité de

[18] Stratégie utilisée notamment par L. Ayres et A. Horner. Voir [11].
[19] Le fonctionnement avec des classes de hauteurs est plus pratique que des fréquences dans le processus de création musicale. La conversion, dans Csound, d'une fréquence vers une classe de hauteur n'est pas directe, contrairement à l'inverse.

[20] Soulignons ici l'aspect successif du déclenchement des événements.

lignes de code, recherches minutieuses de chaque classe de hauteur, calcul de la durée, sans oublier les autres aspects d'interprétation tels que les variations dans les nuances ou dans l'amplitude. Ces nombreuses réflexions nous ont amené à repenser l'espace composable dans Csound dans ce cadre précis.

3.3.2. *UDO et Opcode event*

Placés dans l'entête de l'orchestre, les UDO (User-Defined Opcodes ou Opcodes définis par l'utilisateur) nous permettent de créer des Opcodes adaptés à une problématique donnée. Nous avons mis en place une stratégie de traitement des micro/macro-ornements avec l'Opcode *event* [30], qui permet de générer un événement de partition depuis un instrument de l'orchestre, que nous appelons « l'instrument de contrôle ». La syntaxe de l'Opcode *event* est la suivante :

```
event "scorechar", kinsnum, kdelay, kdur, [, kp4] [, kp5] [, ...]
```

Scorechar correspond à *i* (instrument), *kinsnum* correspond au numéro de l'instrument à appeler (« instrument contrôlé »), *kdelay* est le temps du début de l'événement en seconde, *kdur* est la durée de l'événement et *kp4, kp5* etc. sont les paramètres utilisés par l'instrument appelé si nécessaire. Notons que l'instant 0 de l'Opcode *event* correspond au début de l'événement depuis le score, et non pas à l'instant 0 dans la partition (par exemple, si l'événement dans le score commence à 1, si *kdelay = 0,* l'événement se déclenchera à l'instant 1)[21].

3.3.2.1 *Boucles manuelles*

Voici le code pour l'UDO *macroonment2* concernant l'ornementation décrite dans la première section (Figure 5) :

```
opcode macroornement2, 0, kkkkkkkkk
ktime2 init 0.07
event "i", 2, 0, ktime2, p4, p5, p6, p7, p8,
p9, p10
kampl = p4-1
event "i", 2, ktime2, .07, kampl, p5+p12,
p6+p12, p7, p8, p9, .1
kampl = p4-1
ktime2 = ktime2 + .07
event "i", 2, ktime2, .07, kampl, p5, p6, p7,
p8, p9, .1
kampl = p4-1
ktime2 = ktime2 + .07
event "i", 2, ktime2, .07, kampl, p5+p12,
p6+p12, p7, p8, p9, .1
kampl = p4-1
ktime2 = ktime2 + .07
event "i", 2, ktime2, .07, kampl, p5, p6, p7,
p8, p9, .1
kampl = p4-1
ktime2 = ktime2 + .07
event "i", 2, ktime2, .07, kampl, p5+p13,
p6+p13, p7, p8, p9, .1
kampl = p4-1
ktime2 = ktime2 + .07
event "i", 2, ktime2, .07, kampl, p5, p6, p7,
p8, p9, .1
kampl = p4-1
ktime2 = ktime2 + .07
event "i", 2, ktime2, .51, p4, p5+p12, p6+p12,
p7, p8, p9, p10
endop
```

L'UDO *macroonment2* possède neuf entrées, correspondantes aux champs p4, p5, p6, p7, p8, p9, p10, p12, p13 de la partition et zéro sortie : il déclenche les événements sonores en envoyant de nouvelles valeurs à l'instrument contrôlé.

Le code dans la partition, contrôlant à son tour l'UDO en question, est le suivant (p-champs en haut, code en bas) :

```
;p1  p2  p3  p4    p5     p6       p7      p8
;p9  p10  p11  p12    p13
 i1    1   1   80   8.05   8.05   12.09   3   4
0         2   .02     -.015
```

p1 : instrument de contrôle, p2 : instant de début en seconde, p3 : durée, p4 : amplitude en décibel, p5 et p6 : classes de hauteurs de la note « fondamentale », p7 : période d'échantillons –Opcode *pluck*, p8 : amplitude d'un *lfo*, p9 : fréquence du *lfo*, p10 : contrôle d'attaque, p11 : n° de UDO à utiliser, p12 : « rapport de fluctuation ascendant », p13 : « rapport de fluctuation descendant ».

Nous définissons avec une seule ligne de code dans la partition la note musicale sur laquelle nous envisageons d'appliquer le macro-ornement, en fixant les rapports de fluctuations autour de cette note. L'UDO permet aussi une variation d'intensité et de phrasé (l'effet legato) (Figure 9).

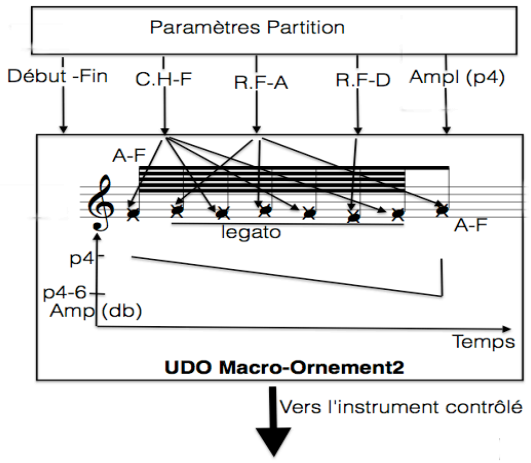


Figure 9. Fonctionnement de la macro-ornement2. (C.H-F : classe de hauteur de la note fondamentale, R.F-A : rapport de fluctuation ascendant, R.F-D : rapport de fluctuation descendant, A-F : attaque franche).

[21] Il existe d'autres Opcodes qui déclenchent un événement depuis l'orchestre, tel que *schedule* et *scorline*. Pour une étude comparative des diverses possibilités de création de boucle (répétition des événements), voir [4].

3.3.2.2 *Boucles automatiques*

L'isochronie des événements générés par un macro-ornement se fait automatiquement avec l'Opcode *metro*, voici sa syntaxe :

```
ktrig  metro  kfreq [, initphase]
```

ktrig varie entre 0 et 1 (0 = pas d'impulsion, 1= impulsion) et *kfreq* est la fréquence des déclenchements en HZ. Par exemple, un événement dure 0,06s, la fréquence du métronome est 1/0,06.

Prenons un exemple d'ornementation, que nous appelons *macroornement11*, qui parcourt toutes les notes musicales d'un maqâm en mouvement descendant (Figure 10):

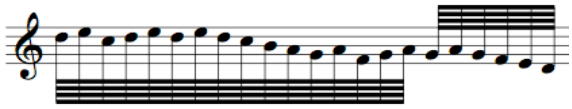


Figure 10. *Macroornement11*

Procédons par une logique heuristique dans une démarche de simplification, nous distinguons trois niveaux de fluctuations appliqués à une note fondamentale : un premier rapport libre choisi, un rapport ascendant et un rapport descendant. Pendant la lecture d'une table qui contient des classes de hauteurs, nous devons définir le premier rapport dans le p-champ p12, les autres rapports sont générés automatiquement de cette façon (Figure 11) :

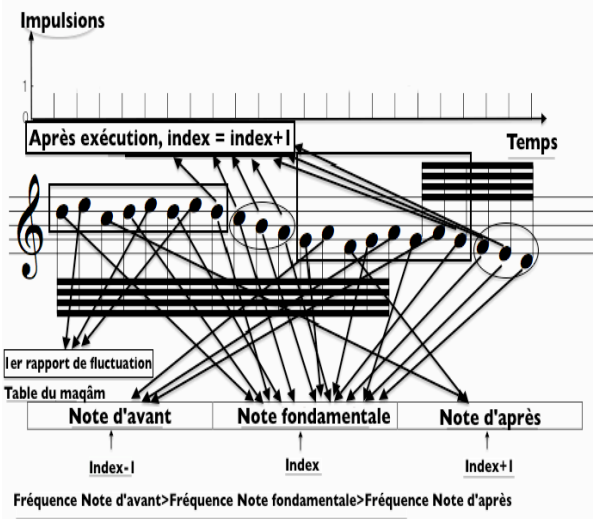


Figure 11. Fonctionnement du *macroornement11*

Les notes fondamentales sont les notes extraites de la table du maqâm, sur lesquelles nous devons appliquer des fluctuations précises et en respectant l'ossature du macro-ornement, ce qui crée quatre passages mélodiques[22]. Le changement de la fondamentale s'effectue avec l'incrémentation d'un compteur index.

Les effets de nuances et de phrasés s'ajoutent sur ce principe de fonctionnement avec une intensité sonore décroissante et une reproduction du legato avec des attaques franches pour le premier et le dernier événements.

3.3.2.3 *Transposabilité*

En partant d'une classe de hauteur dans la partition, un macro-ornement parcourt le contenu d'une table d'un maqâm donnée. Or, si la classe de hauteur de la partition ne correspond pas à la première classe de hauteur de la table, le macro-ornement doit générer et mettre à jour automatiquement les autres valeurs.

La fonction *semitone* permet de calculer un facteur pour élever ou abaisser une fréquence d'un certain nombre de demi-tons. La difficulté de ce genre d'opération repose sur les différents traitements de conversion. Rappelons que les classes de hauteurs sont exprimées en deux parties : un nombre entier et une partie décimale, il faudra donc en premier lieu extraire la différence d'octave entre les deux classes de hauteurs, ensuite, il faut extraire la différence dans la partie décimale. Le nombre d'octave de différence sera multiplié par sa valeur correspondante dans la partie décimale (une octave = 12 notes, le nombre d'octave est multiplié par 0,12), qui s'ajoutera à la partie décimale de la classe la plus élevée, la partie décimale de la deuxième classe sera enlevée de ce résultat, qui sera multiplié par cent. Cette valeur sera positive si la première classe de hauteur de la table est inférieure à la classe reçue de la partition, elle permettra d'élever les notes du maqâm, si la première classe est supérieure, le facteur de transposition permettra de baisser les notes. Si les deux classes de hauteurs se valent, les valeurs de la table restent inchangées.

Voici le code du *macro-ornement11*, illustrant tout le processus opératoire décrit auparavant:

```
opcode macroornement11, 0, kkkkkkkkk
klegato init p10
kmetro init 0
kindextable init 0
kamplitude init p4
kmetronome metro (1/0.06)
;première phase d'extraction
if kmetronome == 1 then
kfrequence table kindextable , p14
kmetro = kmetro +1
kamplitude = kamplitude - (p3/10)
;algorithme de : « après exécution, index =
index +1 »
if kmetro == 8 || kmetro == 9|| kmetro == 10||
kmetro == 11|| kmetro == 19|| kmetro == 20||
kmetro == 21 then
kindextable = kindextable +1
;les notes d'avant
elseif kmetro == 13 || kmetro == 16 || kmetro
== 18 then
kfrequence table kindextable - 1 , p14
;premier rapport de fluctuation et legato à
partir de la 2ème note
elseif kmetro == 2 || kmetro == 5 || kmetro ==
7 then
klegato = 0.1
kfrequence = kfrequence + p12
;note d'après
elseif kmetro == 3 || kmetro == 14 then
kfrequence table kindextable + 1 , p14
;fin du légato pour la dernière note (attaque
franche)
elseif kmetro == 22 then
```

[22] Nous pouvons les considérer comme des macro-ornements.

```
klegato = p10
endif
;transposabilité
kfrequencerep table 0, p14
if kfrequencerep > p5 then
koctave = int(kfrequencerep)-int(p5)
kintable = kfrequencerep - int(kfrequencerep) +
(0.12 * koctave)
kintp = p5 - int(p5)
ktone = kintable - kintp
kdiff = -ktone*100
ktranspose = semitone(kdiff)
elseif kfrequencerep < p5 then
koctave = int(p5) - int(kfrequencerep)
kintable = kfrequencerep - int(kfrequencerep)
kintp = p5 - int(p5) + (0.12 * koctave)
ktone = kintp - kintable
kdiff = ktone*100
ktranspose = semitone(kdiff)
else
ktranspose = 1
endif
kcycle = cpspch(kfrequence)*ktranspose
kcycle1 = octcps(kcycle)
kcycle2 = pchoct(kcycle1)
;génération de l'événement
event "i", 2, 0, .06, kamplitude, kcycle2,
kcycle2, p7, p8, p9, klegato
endif
endop
```

3.3.3. *Procédure d'appel des UDO*

L'instrument de contrôle permet la sélection d'un UDO en fonction du p-champ *p11* défini dans la partition. Il associe une valeur au nom du macro-ornement afin d'être appelé depuis le score. Il transfère également les valeurs d'entrées afin d'être utilisées et mises à jour.

Ci-dessous un extrait de code dans l'instrument de contrôle :

```
instr 1
ktrigger init p11
if (ktrigger == 1) then
macroornement1  p4,  p5,  p6,  p7,  p8,  p9,  p10,
p12, p13
ktrigger = 0
elseif (ktrigger == 2) then
macroornement2  p4,  p5,  p6,  p7,  p8,  p9,  p10,
p12, p13
ktrigger = 0
;etc…
elseif (ktrigger == 14) then
macroornement14 p4, p5, p7, p8, p9, p10, p14
endif
endin
```

3.4. Fonctionnement général de la bibliothèque

Bien que l'instrument contrôlé dans l'orchestre joue un rôle d'exécution, résultat d'une interactivité et d'acheminements multiples entre les unités fonctionnelles, piloté par l'instrument de contrôle, il joue aussi un rôle exécutoire plus direct. Le jonglage entre ces deux procédés permet l'accomplissement de l'œuvre. Ce deuxième rôle d'instrument autonome permet l'enchainement mélodique, ornementé ou non, créant ainsi une sorte de fluidité donnant une ossature générale à l'œuvre. Voici son code source :

```
instr 2
kvib lfo p8, p9, 3
kgliss linseg cpspch(p5), p3/3,cpspch(p6)
kenv linen ampdb(p4),p3*p10 , p3, p3/4
kgliss=kgliss + kvib
asig pluck kenv, kgliss, cpspch(p7), 0, 1
out asig
endin
```

Les traitements de hauteurs reposent sur *kvib* et *kgliss*, avec un oscillateur basse fréquence pour reproduire des fluctuations[23] et des glissandos avec le traçage d'une suite de segments de droite entre les points spécifiés. Les effets de phrasé sont reproduits avec une enveloppe qui applique un motif d'une attaque et d'une chute en segments de droite, obtenus avec l'Opcode *linen*.
L'événement dans la partition envoie les paramètres de contrôle soit, vers l'instrument de contrôle qui les transfère par la suite vers la bibliothèque d'objets qui les actualise et les réachemine vers l'instrument contrôlé pour les interpréter avec ces nouvelles données, soit directement vers ce dernier pour reproduire le phénomène sonore (Figure 12).

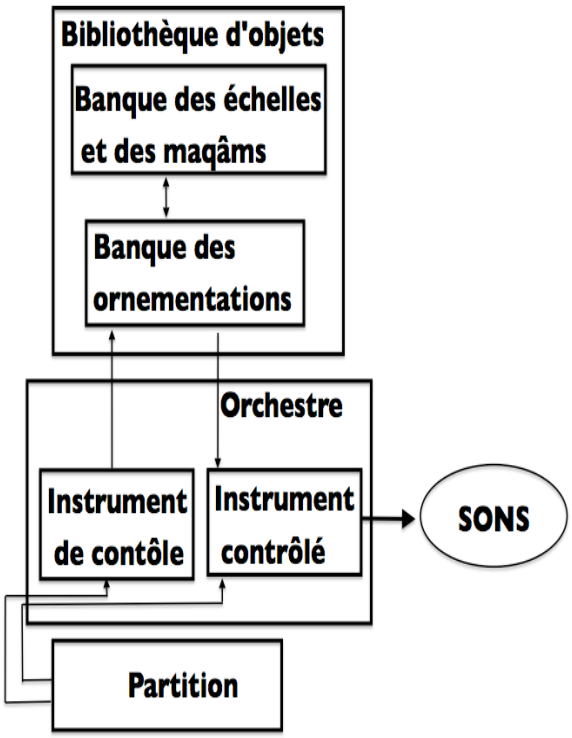


Figure 12. Principe de fonctionnement de la bibliothèque d'objets dans Csound.

[23] Utile notamment en ce qui concerne le quart de ton fluctuant. During montre que cet intervalle fluctue « entre 32 et 39 savarts environ » [22].

4. CONCLUSION, PERSPECTIVES

Pour aborder la question de la composition musicale assistée par ordinateur en ce qui concerne les musiques arabes, nous avons passé en revue leurs grandes lignes physionomiques dans une optique analytique nécessaire pour la modélisation, et par la suite avec une approche visant la création musicale par les moyens de l'informatique.

Nous avons souligné la nécessité d'adaptation qui a abouti à la création d'une bibliothèque d'objets sous Csound et nous avons détaillé son fonctionnement et quelques uns de ses paradigmes intrinsèques, elle compte actuellement environ vingt-trois objets entre maqâms et ornementations.

Dans la perspective du long terme, en premier lieu, un contrôle plus approfondi de la vitesse d'exécution sera mis en place, qui prend en compte les irrégularités dans les passages mélodiques tout en conservant un tempo fixe, ou légèrement plus rapide, en ce qui concerne les micro/macro-ornements.

Ensuite, la bibliothèque sera alimentée régulièrement en UDO et en maqâms de façon à diversifier son contenu en essayant de reproduire une large palette de pratique musicale arabe différente. A ce stade, il est primordial de prendre en compte l'aspect cognitif lié à différentes pratiques musicales arabes selon un matériau musical sans cesse renouvelé.

D'un autre coté, un élargissement de l'instrumentarium, grâce à la synthèse sonore, sera envisagé avec plus de dynamismes et de variations dans le timbre. Les difficultés à ce niveau et à premier abord seront d'envisager la création des UDO propres à chaque instrument, qui soient étudiés selon une logique d'exécution musicale appropriée à chaque instrument. Nous envisagerons l'implémentations des sonorités hybrides afin d'exploiter le potentiel de la synthèse sonore. Cette phase permettra l'aboutissement à d'autres formes d'interprétations musicales tel que l'hétérophonie par exemple.

En dernier lieu, nous n'exclurons pas la possibilité de transposer ce système sur d'autres plateformes logicielles, nous citons à titre d'exemple la possibilité d'implémenter cette bibliothèque dans OpenMusic ou encore dans PWGL. La possibilité de créer un logiciel d'aide aux compositions musicales arabes sous C++ serait éventuellement envisageable.

5. REFERENCES

[1] Abou Mrad, N. « Formes vocales et instrumentales de la tradition musicale savante issue de la renaissance de l'Orient arabe », *Cahiers d'ethnomusicologie* [En ligne], p. 1-24, 2012. Consulté le 19 janvier 2014. URL : http://ethnomusicologie.revues.org/464

[2] Abou Mrad, N. « Musique de l'époque abbasside Le legs de Safiy a-d-Din al Urmawi », Publications de l'Université Antonine, Maison es Cultures du Monde, Hadath-Baabda, Liban, 2005.

[3] Abromont, C., de Montalembert, E., Fourquet, P., Oriol, E., Pauset, B., *Guide de la théorie de la musique*. Fayard, Paris, 2009.

[4] Aikin, J. « Phrase lopps : A beginner's guide », *Csound journal*, Issue 15, version internet, 2011. Consulté le 19 janvier 2014. URL :http://www.csounds.com/journal/issue15/index.html

[5] al Faruqi, L. « L'ornementation dans la musique arabe improvisée : étude des relations entre les arts », *Le monde de la musique*, 4ème journal de l'institut international des études comparatives et de documentation de musique, vol. XX, p. 17-32, 1978.

[6] Al-Kawarizmi, A. *Mafatih al-ulum (Clés des sciences)*. Dar al-Manahel, Beyrouth, 1991.

[7] Arom, S. « Modélisation et modèles dans les musiques de tradition orale », *Analyse musicale*, Société Française d'analyse musicale, n°22, p. 67-78, 1991.

[8] Arom, S. « Réalisations, variations, modèles dans les musiques traditionnelles centrafricaines », *L'improvisation dans les musiques de tradition orale*, Lortat-Jacob B. et Selaf, p. 119-122, 1987.

[9] Arom, S. *Polyphonies et polyrythmies instrumentales d'Afrique centrale : structure et méthodologie*, Selaf, Paris, 1985.

[10] Ayari, M. *L'écoute des musiques arabes improvisées Essai de psychologie cognitive de l'audition*. L'harmattan, Paris, 2003

[11] Ayres, L., Horner, A., « Synthesizing music in csound with a Javanese Gong Ageng », *Proceedings of the 2005 international computer music conference,* Barcelona, Spain, p. 644-647, 2005.

[12] Bach, C.P.E. *Essai sur la véritable manière de jouer d'un instrument à clavier*, Tome I, Traduction de Coulon, J.P. Consulté le 14 janvier 2014. URL : http ://icking-music-archive.org

[13] Barkechli, M. « La gamme de la musique Iranienne », *Cahier d'acoustique n°14*, Annales des télécommunications, Tome 5, n° 5, p. 195-203,1950.

[14] Barkechli, M. « L'évolution de la gamme dans la musique orientale », *Acoustique musicale*, CNRS, p. 39-46,1959.

[15] Belhassen, R. *L'impact de l'échantillonnage dans la musique tunisienne*, mémoire de master 1, Université de Marne La Vallée, 2009.

[16] Belhassen, R. *Mutation esthétique de la musique arabe*, mémoire de master 2, Université de Marne La Vallée, 2010.

[17] Bent, I., Drabkin, W., *L'analyse musicale : histoire et méthodes*, Main d'œuvre, Nice, 1998.

[18] Berthoz, A. *La décision*, Odile Jacob, Paris, 2003.

[19] Camilleri, C., Cohen-Emerique, M. Collectif. *Chocs des cultures ; concepts et enjeux pratiques de l'interculturel*, L'Harmattan, Paris, 1989.

[20] Carvalho, G. « Formaliser la forme », *Actes des journées d'informatique musicale*, Saint Denis, France, version internet, 2005. Consulté le 11/02/2014. URL : http://jim2005.mshparisnord.org/download/GCarvalho.pdf

[21] Dahan, K., Laliberté, M., « Réflexions autour de la question d'interprétation de la musique électroacoustique, *Actes des Journées d'informatique musicale*, Albi, France, version internet, 2008. Consulté le 8/01/2014. URL : http://www.gmea.net/upload/09_KAHAN_JIM2008-final.pdf

[22] During, J., Dufourt, H., Fauquet, J.-M., Hurard, F., « Systèmes acoustiques et systèmes métaphysiques dans les traditions orientales », *L'esprit de la musique*, Klinsieck, p. 177-184, 1992.

[23] Erlanger D', R. *La musique arabe*. Libraire orientaliste Paul Geuthner, Tome III, Paris, 1938.

[24] Féron, F.X., « Les variations dans la vibration : vibratos, trémolos et trilles dans la Trilogie – Les trois stades de l'homme (1956 – 1965) pour violoncelle seul de Giacinto Scelsi », *Filigrane – Musique, esthétique, sciences, société*, n° 15, version internet, 2014. Consulté le 8/01/2014. URL : http://revues.mshparisnord.org/filigrane/index.php?id=516

[25] Ghrab, A. *Commentaire anonyme du kitab al-Adwar Edition critique, traduction et présentation des lectures arabes de l'œuvre de Safi al-Din al-Urmawi*, Thèse de doctorat, Université de Paris Sorbonne, 2009.

[26] Gruber, T.R. « A translation approach to portable ontology specifications », *knowledge acquisition*, n° 5, p. 199-220, 1993.

[27] Lambert, J. « Retour sur le congrès de musique arabe du Caire de 1932. Identité, diversité, acculturation : les prémisses d'une mondialisation », *Congrès des musiques dans le monde de l'islam*, Maison des cultures du monde, Assilah, Maroc, p. 1-6, 2007.

[28] Poché, C. « Le tronc commun des maqâms », Colloque Maqâm et création, Asnières-sur-Oise, France, p. 7-15, 2006.

[29] Viret, J. *Le chant grégorien et la tradition grégorienne*. L'âge d'homme, Lausanne, 2001.

[30] Yi, S. « Control flow » - Part I et II, *Csound journal*, Vol. 1, Issues 3 et 4, version internet, 2006. Consulté le 19 janvier 2014.URL : http://www.csounds.com/journal/

[31] Zouari, H. *Les ornementations dans le tba' rast el dhil : l'exemple du Congrès du Caire de 1932*, Thèse de doctorat, Université de Paris Sorbonne, 2012.

VERS UNE ANALYSE AUTOMATIQUE DES FORMES SONATES

Laurent David [1,3] *Mathieu Giraud* [1] *Richard Groult* [2] *Florence Levé* [1,2] *Corentin Louboutin* [1,4]

[1] LIFL (UMR 8022 CNRS, Universités Lille 1 et Lille 3)
[2] MIS (Université de Picardie Jules Verne, Amiens)
[3] ENSEA (Cergy-Pontoise)
[4] ENS Rennes

{laurent, mathieu, richard, florence, corentin}@algomus.fr

RÉSUMÉ

La forme sonate, bien que ce nom lui ait été donné a posteriori, était très prisée de la période classique jusqu'à la fin de la période romantique. Une pièce de cette forme est composée de trois parties complémentaires, une exposition, un développement et une réexposition. Nous défendons l'idée que l'étude des formes sonates est un défi intéressant pour la musicologie computationnelle. Nous proposons aussi quelques premières expériences sur l'analyse informatique de ce type de partitions. Sur un corpus de premiers mouvements de quatuor à cordes, nous arrivons à localiser approximativement le couple exposition/réexposition dans la majorité des cas.

1. FORMES SONATES

Nous présentons ici la forme sonate, ses modélisations musicologiques, et défendons l'idée qu'elle est un objet de choix pour l'informatique musicale symbolique.

1.1. La forme sonate

La forme sonate est une forme musicale classique développée à partir de la fin du XVIII^{e} siècle jusqu'à la fin du XIX^{e} siècle. Provenant initialement d'une forme binaire qui a été étendue, cette forme comporte trois parties jouant des rôles musicaux complémentaires (figure 1) :

- *L'exposition* a un rôle à la fois rhétorique (mélodique) et harmonique, permettant à la fois d'exposer les thèmes (deux en général, principal et secondaire) qui seront utilisés dans les autres parties et d'affirmer la tonalité principale de la pièce. En général, le *thème principal* (P) est exposé à la tonique (**I**) puis, après une transition et une césure médiane (*medial caesura*, MC), le *thème secondaire* (S) est exposé dans une autre tonalité, généralement au ton de la dominante (**V**) pour les formes sonates en mode majeur.
- *Le développement* réutilise les thèmes présentés dans l'exposition tout en créant une tension tonale. Présentant de nombreuses modulations, c'est une zone tonalement instable.
- Enfin, *la réexposition* (*recapitulation* en anglais) présente à nouveau les thèmes principaux. Cette fois-ci, le deuxième thème est joué à la tonique, ce qui permet une affirmation de la tonalité principale de la pièce.

La figure 2 montre un exemple d'une forme sonate très simple, dans une sonatine pour piano de Kuhlau.

Plusieurs marqueurs se trouvent entre les différentes parties, en particulier des cadences (fins de phrases musicales). Le thème principal présenté dans l'exposition est souvent suivi d'une transition se terminant par une *demi-cadence* (*half cadence*, HC). Le thème secondaire se termine généralement par une *cadence parfaite* (*Perfect Authentic Cadence*, PAC), qui conclut musicalement l'exposition. Enfin, l'exposition se termine par une *conclusion* (qui ne présente généralement pas de nouvel élément thématique). Les cadences et la conclusion, parfois transformées, se retrouvent dans la réexposition.

Il y a beaucoup de variations possibles sur cette structure, selon l'époque et les compositeurs. Haydn utilise par exemple beaucoup de formes sonates mono-thématiques, comme dans l'op. 33 n^{o} 2. En fait, la forme sonate ne se caractérise pas dans la succession des sections, mais plus dans un *équilibre à grand échelle* : les tensions tonales (autre tonalité) et rhétoriques (thème S, texture) créées lors de l'exposition et étendues dans le développement finissent par se résoudre dans la réexposition.

1.2. Approches musicologiques de la forme sonate

Certains des principes de composition de la forme sonate avaient été théorisés dès la fin du XVIII^{e} siècle [15, 23]. Cependant, le terme « forme sonate » a été utilisé pour la première fois dans les années 1830 par A. B. Marx dans une série d'articles dans le *Berliner allgemeine musikalische Zeitung*. La forme a été ensuite théorisée plus en

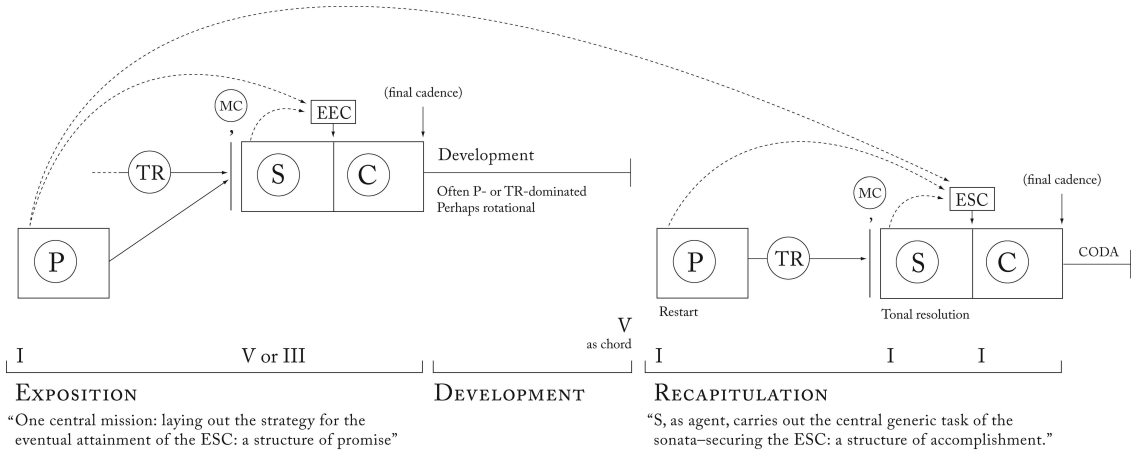


Figure 1. Structure d'ensemble d'une forme sonate, schéma et notations tirés du livre de Hepokoski et Darcy [12, page 17]. La forme sonate est construite sur un thème (ou une zone thématique) principal (P), un thème secondaire (S) et une conclusion (C). Entre les deux thèmes, la transition (TR) se termine par la césure médiane (*medial caesura*) (MC). À la fin du thème secondaire se trouve une cadence parfaite, dénommée EEC (*Essential Expositional Closure*) dans l'exposition et ESC (*Essential Structural Closure*) dans la réexposition (*recapitulation*).

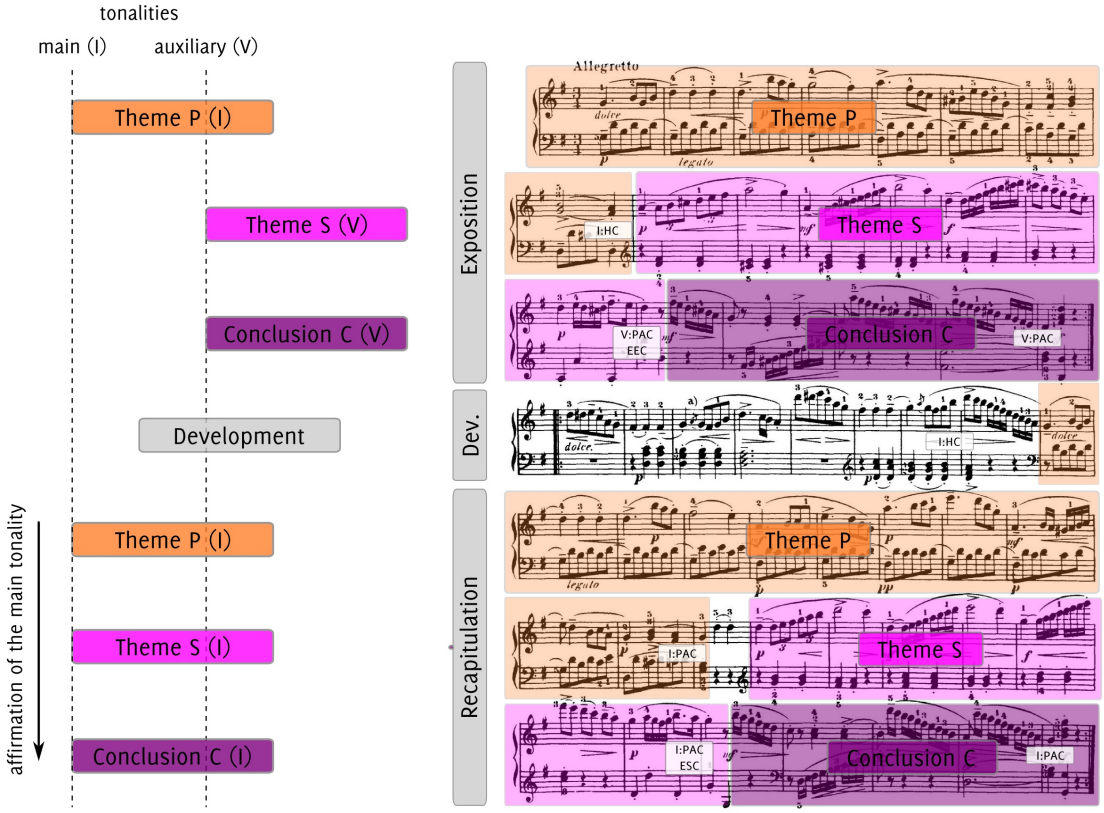


Figure 2. Analyse structurelle de l'Allegretto de la sonatine Op. 55, no. 2 de Kuhlau, pour piano, en utilisant les conventions de [12]. Les cadences (fin de phrases) sont indiquées par les termes HC (*Half Cadence*, demi-cadence) et PAC (*Perfect Authentic Cadence*, cadence parfaite), précédés de la tonalité de cette cadence (ici **I** et **V**). Les termes EEC et ESC désignent les fins de sections (voir figure 1). Cette sonatine est très simple, avec un développement très court (Dev.), et pratiquement aucune transition entre les zones thématiques P et S. Le groupe thématique P dure 8 mesures lors de l'exposition et 11 lors de la réexposition, ce qui renforce la stabilisation de la tonique dans la réexposition. Le thème S et la conclusion C sont exactement transposés entre l'exposition et la réexposition.

détail, notamment par Marx [17] et Czerny [4].

Plus récemment, de nouvelles théories systématiques sur les formes sonates ont été proposées [2, 3, 16, 18, 22, 24]. Le travail d'Hepokoski et al. [10, 11], culminant dans le livre [12], est aujourd'hui une bonne référence. Enfin, on trouve certaines analyses en profondeur de corpus spécifiques : pour les oeuvres Beethoven on se référera à [9, 21] pour les quatuors à cordes et à l'ouvrage de D. Tovey [25] pour les sonates pour piano.

Comme toute analyse humaine, l'analyse des formes sonates n'obéit pas à un ensemble de règles figées, mais dépend de choix analytiques subjectifs qui ne font pas toujours consensus. De plus, certaines pièces portent des ambiguïtés structurelles. En particulier, il est souvent délicat de déterminer précisément la fin du premier thème de la transition, surtout lorsque plusieurs motifs transitionnels ou thématiques se répètent dans l'exposition et la réexposition comme dans le quatuor K. 155 n° 2 de Mozart. Enfin, de telles analyses peuvent sembler anachroniques, la forme sonate n'existant pas en tant que telle pour les compositeurs de l'époque. Ces analyses peuvent tout de même être pertinentes en éclairant quelques principes compositionnels avec un certain recul vis-à-vis du répertoire.

1.3. Analyser informatiquement la forme sonate ?

La communauté MIR (Music Information Retrieval) s'intéresse à l'analyse musicale dans son ensemble, à partir de données audio ou symboliques. Bien que l'analyse structurelle soit au cœur de ces thématiques, à ce jour, peu de travaux en MIR se concentrent sur les formes sonates. Même si les formes sonates servent de support à l'extraction de motifs (voir par exemple [20]), leur analyse à grande échelle est peu abordée. Dans un article récent, Jiang et Müller effectuent des comparaisons de signaux audio pour retrouver le couple exposition/réexposition sur les premiers mouvements dans 28 sonates pour piano de Beethoven [14]. Ils proposent aussi de suivre les changements harmoniques à l'intérieur des différentes parties.

La donnée symbolique de la partition devrait permettre d'être bien plus précis dans l'analyse. Nous pensons que l'analyse des formes sonates est un défi intéressant pour la recherche en informatique musicale symbolique, demandant de combiner des aspects locaux (motif et thèmes, cadences...) avec des aspects globaux (plan tonal, structure d'ensemble).

Le premier défi est dans la *détection de la structure générale* : couple exposition/réexposition, avec des marqueurs précis lorsqu'ils existent, formant la structure tonale à grande échelle.

Le second est de retrouver la *structure tonale et motivique du développement*. Le développement montrant souvent une grande instabilité des tonalités, il serait intéressant de pouvoir décrire de manière cohérente ce parcours tonal. Il serait également intéressant de trouver quels éléments thématiques de l'exposition sont utilisés et sous quelle forme ils sont repris, la difficulté étant qu'il peut s'agir de très courts motifs mélodiques ou rythmiques ou qu'ils peuvent présenter des variations conséquentes.

Toute approche automatisée doit être conçue en gardant à l'esprit les limites que nous avons évoquées sur la subjectivité de toute analyse – cependant, certains points simples d'analyse peuvent faire consensus et permettre d'évaluer les algorithmes.

2. PISTES POUR UNE ANALYSE INFORMATIQUE

Nous décrivons ici quelques expériences sur l'analyse informatique de formes sonates, en partant de partitions sous forme de données symboliques. Nous partons de fichiers `.krn` [13], qui encodent les partitions plus précisément que des fichiers `.midi` (avec notamment les informations d'enharmoniques), mais nous nous limitons dans cette étude aux informations de hauteur et de durée. Est-il possible de retrouver avec ces informations la structure exposition/réexposition et d'étudier le développement ?

Pour ces expériences, nous nous sommes limités à des quatuors à cordes, ce qui facilite l'analyse étant donné que les pièces sont séparées en quatre voix (premier et deuxième violon, alto, violoncelle). Nous avons ainsi rassemblé 11 premiers mouvements de quatuors de Mozart, Haydn et Schubert (figure 4).

Afin d'évaluer nos algorithmes, nous avons manuellement identifié pour chacun de ces mouvements le couple exposition/réexposition, en nous appuyant sur les analyses de Hepokoski et Darcy [12] ainsi que sur notre propre analyse.

2.1. Détection du couple Exposition/Réexposition

La figure 3 montre une analyse obtenue sur le quatuor K. 157 n° 4 de Mozart avec les opérations suivantes :

- recherche de cadences parfaites (**V** vers **I** en position fondamentale, avec arrivée sur la tonique à la voix supérieure) [5] ;
- recherche de fins potentielles de phrases par une simple détection rythmique (notes plus longues que leur entourage ou suivies de silences) ;
- recherche des zones P/S, par un algorithme de Mongeau-Sankoff qui calcule, par programmation dynamique, les similarités entre deux mélodies en minimisant une distance d'édition [19]. Nous utilisons une fonction de score qui favorise les similarités diatoniques [7]. Cette recherche concerne uniquement la voix supérieure (premier violon).

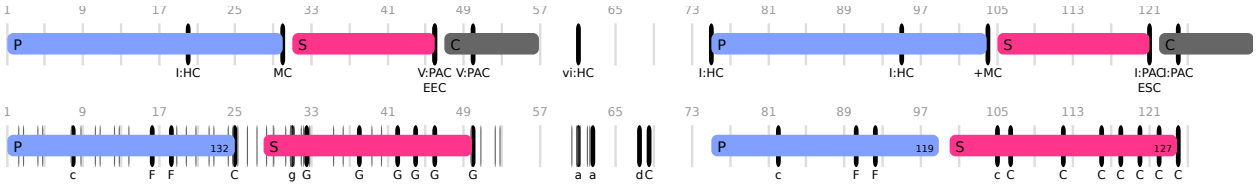


Figure 3. Recherche du couple exposition/réexposition dans le premier mouvement du quatuor à cordes K. 157 n° 4 de W. A. Mozart comprenant 126 mesures. Le schéma du haut montre une analyse possible (faite par nos soins) avec cadences et marqueurs structurels en utilisant les notations de [12] (voir figure 2). Le schéma du bas représente ce qui est trouvé par le programme. Les traits fins verticaux dans l'exposition montrent des fins de phrases potentielles, et les traits gras des cadences potentielles (majuscule : majeur, minuscule : mineur). Le couple exposition/réexposition est retrouvé, mais la zone S prédite est détectée en amont de la zone réelle S, la fin de la transition entre P et S étant déjà transposée à la dominante dans l'exposition. De plus, la zone S prédite s'étend aussi sur la conclusion C.

	ℓ	Analyses	P	S
Haydn op. 33 n° 2	90m 4/4		+	
Haydn op. 33 n° 3	167m 4/4		=	+8
Haydn op. 33 n° 5	305m 2/4		+	+3
Haydn op. 54 n° 3	189m 2/2		=	+13
Haydn op. 64 n° 4	99m 4/4		−	
Mozart K. 80 n° 1	68m 3/4		−	+4
Mozart K. 155 n° 2	119m 4/4		+	+3
Mozart K. 156 n° 3	184m 3/8		+	0
Mozart K. 157 n° 4	126m 4/4		+	−2
Mozart K. 387 n° 14	170m 4/4		=	−6
Schubert op. 125 n° 1	255m 2/2		+	−23

Figure 4. Recherche du couple exposition/réexposition sur 11 premiers mouvements de quatuors à cordes en forme sonate. Pour chaque pièce, la colonne « ℓ » indique la longueur en mesures ainsi que le mètre. La colonne « Analyses » indique, en haut, quelques éléments de notre analyse manuelle pour l'exposition comme pour la réexposition (P : zones thématique principale, éventuellement suivie d'une transition, S : zone thématique secondaire, C : conclusion, O : coda), et, en bas, l'analyse effectuée informatiquement. La colonne « P » indique si le thème est trouvé exactement (+), avec quelques mesures de décalage dues à la présence d'une introduction (=) ou mal ou non trouvé (−). Lorsque le second thème est trouvé, la colonne « S » indique le décalage, en mesures, entre cette prédiction et le « vrai » second thème. Notons que le quatuor op. 33 n° 2 de Haydn est monothématique, les deux zones P et S étant construites sur le même matériau mélodique. De plus, dans les quatuors op. 54 et 64 de Haydn, il n'y a pas de réexposition pour S.

La zone P est recherchée du départ à une fin de phrase, sans transposition. La zone S est recherchée également après la zone P, sous la contrainte d'une transposition de la dominante vers la tonique (**V** vers **I**). Dans ce mouvement du quatuor K. 157, on voit que la stratégie met en évidence le couple exposition/réexposition.

La figure 4 montre les résultats sur les 11 pièces. Le début de P, marquant le début du couple exposition/réexposition, est retrouvé 9 fois à la position correcte, dont 7 fois de manière unique. Dans 8 de ces 9 cas, une deuxième zone (S) est identifiée avec une transposition de **V** vers **I**, dans la plupart des cas à moins de 10 mesures du début réel du thème S :

- la fin de la transition entre P et S peut être identique (mais transposée) entre l'exposition et la réexposition (prédiction de S avant le S réel) ;
- le thème S peut être transformé ou repris à une autre voix (prédiction de S après le S réel).

Certains thèmes ne sont pas identifiés ou détectés à des positions incorrectes (2 thèmes P et 3 thèmes S). Par exemple, dans le quatuor K. 80, on ne trouve pas le thème P car seule la fin est reprise dans la réexposition. Dans le quatuor K. 387, le thème S est joué par le *deuxième* violon, ce qui empêche sa détection correcte.

2.2. Vers une analyse du développement

La figure 5 montre quelques pistes d'analyse du développement dans le premier mouvement du quatuor op. 33 n° 5 de Haydn, sur le plan tonal ou pour la recherche d'éléments thématiques présents dans l'exposition :

- détection de cadences parfaites ;
- recherche d'éléments thématiques des thèmes P et S (relevés manuellement), de nouveau avec un algorithme de Mongeau-Sankoff adapté ;
- et détection de marches harmoniques par l'algorithme proposé dans [6].

Ces résultats préliminaires montrent qu'il est possible de fournir quelques éléments d'analyse sur le développement. Ces techniques demanderaient à être complétées, et il serait intéressant de fournir une structure possible du développement en rassemblant ces divers éléments, par exemple par un modèle probabiliste.

3. DISCUSSION

La forme sonate est une structuration que l'on retrouve dans de nombreuses œuvres classiques et romantiques. Nos expériences sur quelques pièces montrent qu'il est possible informatiquement d'analyser ces partitions en retrouvant certains éléments du couple exposition/réexposition (début de la réexposition, parcours tonal à grande échelle). Pour progresser vers une analyse automatisée de telles formes, plusieurs pistes nous semblent importantes :

- un objectif serait d'identifier précisément (lorsque c'est possible) la fin du premier thème et le début du second, en s'appuyant sur la *césure médiane* (lorsqu'elle existe) [11, 12] ;
- plus généralement, on pourrait développer des méthodes *rassemblant plusieurs éléments analytiques* isolés (motifs, tonalités et cadences, marches harmoniques...) dans une analyse globalisante, que ce soit pour le développement ou pour la forme dans son ensemble ;
- de plus, ces méthodes devraient aussi pouvoir traiter directement du *matériau polyphonique* afin de permettre l'étude des pièces pour piano.

Enfin, pour évaluer et étalonner ces méthodes, il serait souhaitable de réaliser des *fichiers de vérité* tels que ceux réalisés sur les fugues [5]. Ces fichiers devraient s'appuyer sur des analyses de référence, comme par exemple, pour les oeuvres de Beethoven, celles de Helm, Radcliffe ou Tovey [9, 21, 25]. Ils contiendraient, pour un corpus varié, la position et la longueur des différents thèmes dans l'exposition et la réexposition, des motifs dans le développement, les cadences et le plan tonal global de chaque pièce. Ces fichiers devraient aussi permettre de coder les ambiguïtés possibles d'analyse [1].

Remerciements. Nous remercions Maxime Joos et Marc Rigaudière pour leurs discussions fructueuses.

4. REFERENCES

[1] Frédéric Bimbot, Emmanuel Deruty, Gabriel Sargent, and Emmanuel Vincent. Semiotic structure labeling of music pieces : Concepts, methods and annotation conventions. In Gouyon et al. [8], pages 235–240.

[2] William E. Caplin. *Classical Form : A Theory of Formal Functions for the Instrumental Music of Haydn, Mozart, and Beethoven.* Oxford University Press, 2000.

[3] William E. Caplin. The Classical Sonata Exposition : Cadential Goals and Form-Functional Plans. *Tijdschrift voor Muziektheorie*, 6(3) :195–209, 2001.

[4] Carl Czerny. *School of Practical Composition*. London, 1848.

[5] Mathieu Giraud, Richard Groult, Emmanuel Leguy, and Florence Levé. Computational Fugue Analysis. *submitted*, 2014.

[6] Mathieu Giraud, Richard Groult, and Florence Levé. Detecting episodes with harmonic sequences for fugue analysis. In Gouyon et al. [8], pages 457–462.

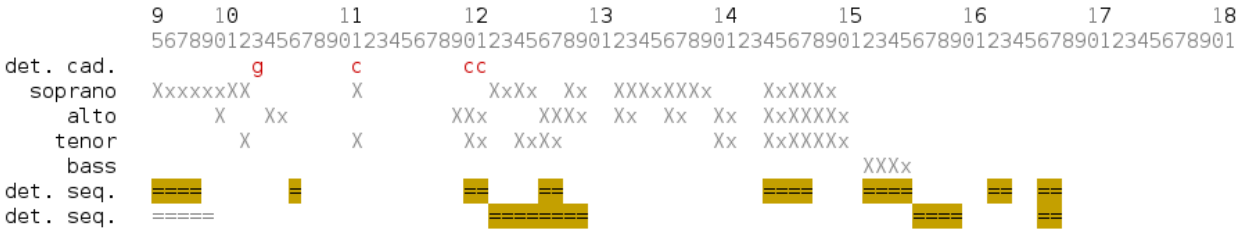


Figure 5. Expérimentation dans l'analyse de développement appliquée au premier mouvement du quatuor op. 33 n° 5 de Haydn (mesures 95 à 181). La première ligne montre une détection de cadences parfaites. Les lignes suivantes décrivent la détection d'éléments constitutifs des thèmes P et S dans le développement. Les deux dernières lignes montrent la détection de marches harmoniques.

[7] Mathieu Giraud, Richard Groult, and Florence Levé. Subject and counter-subject detection for analysis of the well-tempered clavier fugues. In Mitsuko Aramaki, Mathieu Barthet, Richard Kronland-Martinet, and Sølvi Ystad, editors, *CMMR*, volume 7900 of *Lecture Notes in Computer Science*, pages 422–438. Springer, 2012.

[8] Fabien Gouyon, Perfecto Herrera, Luis Gustavo Martins, and Meinard Müller, editors. *Proceedings of the 13th International Society for Music Information Retrieval Conference, ISMIR 2012, Mosteiro S.Bento Da Vitória, Porto, Portugal, October 8-12, 2012*. FEUP Edições, 2012.

[9] Theodor Helm. *Beethovens Streichquartette : Versuch einer technischen Analyse dieser Werke im Zusammenhange mit ihren geistigen Gehalt*. Leipzig, 1885.

[10] James Hepokoski. Beyond the sonata principle. *J. of the American Musicological Society*, 55(2) :91, 2002.

[11] James Hepokoski and Warren Darcy. The medial caesura and its role in the eighteenth-century sonata exposition. *Music Theory Spectrum*, 19(2) :115–154, 1997.

[12] James Hepokoski and Warren Darcy. *Elements of Sonata Theory : Norms, Types, and Deformations in the Late-Eighteenth-Century Sonata.* Oxford University Press, 2006.

[13] David Huron. Music information processing using the Humdrum toolkit : Concepts, examples, and lessons. *Computer Music J.*, 26(2) :11–26, 2002.

[14] Nanzhu Jiang and Meinard Müller. Automated methods for analyzing music recordings in sonata form. In Alceu de Souza Britto Jr., Fabien Gouyon, and Simon Dixon, editors, *ISMIR*, pages 595–600, 2013.

[15] Heinrich Christoph Koch. *Versuch einer Einleitung zur Composition (volume 3)*. Leipzig, 1793.

[16] Steve Larson. Recapitulation recomposition in the sonata-form first movements of Haydn's string quartets : Style change and compositional technique. *Music Analysis*, 22(1-2) :139–177, 2003.

[17] Adolph Bernhard Marx. *Die Lehre von der musikalischen Komposition (vol. 2 and 3)*. Leipzig, 1838, 1845.

[18] Jan Miyake. *The Role of Multiple New-key Themes in Selected Sonata-form Exposition*. PhD thesis, Univ. of New York, 2004.

[19] Marcel Mongeau and David Sankoff. Comparison of Musical Sequences. *Computers and the Humanities*, 24(3) :161–175, 1990.

[20] Oriol Nieto and Morwaread M. Farbood. Perceptual evaluation of automatically extracted musical motives. In *Proceedings of the 12th International Conference on Music Perception and Cognition*, pages 723–727, 2012.

[21] Phillip Radcliffe. *Beethoven's String Quartets*. E. P. Dutton, 1965.

[22] Leonard Ratner. *Classical Music : Expression, Form, and Style*. Schirmer, 1980.

[23] Anton Reicha. *Traité de haute composition musicale (volume 2)*. Paris, 1826.

[24] Charles Rosen. *Sonata Forms*. W. W. Norton, 1980.

[25] Donald Francis Tovey. *A Companion to Beethoven's Pianoforte Sonatas : Complete Analyses*. AMS Press (reedited in 1998), 1931.

REKALL : UN ENVIRONNEMENT OPEN SOURCE POUR DOCUMENTER, ANALYSER LES PROCESSUS DE CREATION ET FACILITER LA REPRISE DES ŒUVRES SCENIQUES

Clarisse Bardiot
UVHC
Clarisse_bardiot@mac.com

Thierry Coduys
Le hub
Thierry.Coduys@le-hub.org

Guillaume Jacquemin
Buzzing Light
gj@buzzinglight.com

Guillaume Marais
Buzzing Light
gm@buzzinglight.com

RÉSUMÉ

Les *Digital performances*, dont le développement commence dans les années 1960, comprennent de nombreuses œuvres musicales ou du moins une composante sonore importante. Du fait de l'obsolescence rapide des technologies, les *Digital performances*, sont non seulement éphémères comme toute œuvre scénique mais engendrent également des traces de plus en plus éphémères. Il devient ainsi très difficile non seulement de documenter les œuvres et leurs processus de création, mais également de les reprendre, voire de les insérer dans un répertoire. Dans le domaine des *digital performances*, le modèle de la partition musicale, ou celui d'un système de notation, qui a été avancé par plusieurs chercheurs, est loin de résoudre les problèmes de documentation nécessaires à la reprise des œuvres. Par ailleurs, plusieurs initiatives ont été menées pour documenter très précisément une œuvre sans développer pour autant de proposition générale qui pourrait s'appliquer à un ensemble d'œuvres, voire aux arts de la scène de manière générale. Plutôt que de développer un système de notation, nous proposons de développer un outil d'annotation qui puisse être utilisé dans un très grand nombre de cas, qu'il s'agisse de chorégraphie, de théâtre ou de musique. Rekall est un environnement *open source* pour documenter, analyser les processus de création et faciliter la reprise des œuvres scéniques. C'est un logiciel qui permet de documenter les *digital performances*, en prenant en compte le processus de création, la réception et les différentes formes d'un spectacle. Il s'adresse à la fois aux artistes, aux techniciens, aux chercheurs et au grand public. Rekall est une réponse aux problématiques de documentation et de conservation des arts à composante technologique, ainsi qu'aux difficultés rencontrées par les artistes lors de la reprise d'un spectacle dont les technologies sont devenues obsolètes, tout en étant au plus près de la démarche propre à chaque artiste, à chaque compagnie. Il permet à la fois de rendre compte des technologies utilisées dans le spectacle et d'en offrir une description pour éventuellement proposer une alternative avec d'autres composantes. Le fonctionnement de Rekall s'articule essentiellement autour des documents de création et propose une préservation active des *digital performances*.

1. INTRODUCTION

L'essor de l'électronique puis de l'informatique ont entraîné des bouleversements esthétiques majeurs dans les arts de la scène à partir des années 1960 : téléprésence, acteurs et danseurs dotés de capteurs et autres prothèses, interaction en temps réel entre les performeurs, l'image et le son… Parmi les artistes représentatifs de ce mouvement, désigné parfois sous le nom de "digital performance" ou "théâtres virtuels", citons Merce Cunningham, Trisha Brown, Denis Marleau, Dumb Type, Myriam Gourfink, Marcel.lí Antúñez Roca, Michèle Noiret, Caden Manson, Laurie Anderson, Troika Ranch, Palindrome, Blast Theory… Les *digital performances* s'étendent également à tout un pan de la création musicale contemporaine, dont l'aspect scénique devient parfois un élément essentiel, avec notamment la présence d'images projetées ou de lumières affectées en temps réel par la musique.

Les composantes technologiques des *digital performances*, qu'elles interviennent pendant le processus de création (captations vidéo d'improvisations, simulations de mise en scène et de scénographies sur divers logiciels…), ou pendant la représentation (capteurs, dispositifs de téléprésence, images et sons modifiés en temps réel…), renouvellent la question de la documentation des arts de la scène : quelle est la nature des nouveaux documents produits par/pour ces spectacles, comment les analyser, faut-il conserver les programmes informatiques spécifiquement conçus et les rendre accessibles (lisibles) en fonction de l'évolution des programmes et du matériel, dans quelle mesure le hardware et le software doivent-ils être documentés, comment effectuer une captation de ces œuvres ?

Non seulement les *digital performances* sont des œuvres éphémères, comme toute représentation scénique, mais l'on peut faire le même constat pour les documentations techniques. En effet, les technologies sont rapidement obsolètes, qu'il s'agisse du support de conservation, du *hardware* ou encore du *software*. De plus, bien souvent, les compagnies développent leurs propres instruments. Si elles ne documentent pas elles-mêmes leur développement, les différentes versions des logiciels propres, ou encore les configurations du

système, il y a très peu de chance de pouvoir analyser et conserver les composantes technologiques spécifiques à une production. Dans ce contexte, la reprise d'un spectacle à quelques années de distance s'avère très difficile. De plus, comme le soulignent Alain Bonardi et Jérôme Barthélémy à propos de la musique électroacoustique [4], « comment considérer que les résultats d'une nouvelle implémentation d'un traitement numérique sont conformes aux intentions originales du compositeur ? ».[1] À cette question, l'une des réponses les plus souvent apportées est la suivante : il est fondamental de collaborer avec les artistes afin de réunir une documentation la plus détaillée possible. Ils sont de fait les premiers conservateurs de leurs œuvres.

Aujourd'hui, toutes les régies techniques sont numériques, et une partie du processus de création a largement lieu via les ordinateurs et les réseaux : échanges de mails, traitements de texte, rendez-vous à distance via des dispositifs de téléprésence (voix sur IP), images et vidéos numériques pour rassembler des idées, des pistes de travail, usage des réseaux de partages d'images pour mettre à disposition des documents visuels pour l'ensemble de la compagnie, croquis effectués sur tablette numérique, etc. Dans ce contexte, l'obsolescence rapide des technologies devient extrêmement problématique, à la fois pour les artistes qui doivent pouvoir continuer à faire tourner leurs spectacles, et pour les chercheurs qui souhaitent en analyser les processus de création. Les documents numériques que nous avons mentionné plus haut sont alors des traces essentielles pour retracer l'histoire des arts de la scène à l'époque contemporaine.

Dans un premier temps, nous reviendrons sur les principales problématiques engendrées par l'usage du numérique dans les arts de la scène. Puis nous examinerons deux démarches : la notation et l'annotation. Enfin, nous présenterons l'état actuel du développement du logiciel Rekall.

2. PROBLEMATIQUES LIEES A L'UTILISATION DES TECHNOLOGIES NUMERIQUES DANS LES ARTS DE LA SCENE

Rekall est né d'une prise de conscience de différents acteurs (artistes, techniciens, programmateurs, chercheurs) concernant les difficultés liées aux technologies numériques employées dans les arts de la scène (notons que face à des problématiques similaires, le champ de la musique contemporaine a tenté d'apporter des réponses, par exemple via les programmes Mustica et Caspar). Ceux-ci sont confrontés à plusieurs problèmes, que l'on peut résumer ainsi :

- absence d'outil permettant aux équipes techniques une prise de note rapide pendant les répétitions sur les dispositifs technologiques et un rassemblement de tous les documents techniques. Les différentes régies sont séparées, chacun (éclairagiste, régisseur général, vidéaste, ingénieur…) a son propre système de documentation sous format numérique ou papier, plus ou moins structuré. C'est au fur et à mesure des expériences professionnelles que chaque intervenant définit sa propre méthode de travail, son propre guide de « bonnes pratiques », très souvent empirique. Ceci pose de nombreux problèmes : fiabilité de la prise de notes (il est souvent difficile de se souvenir de la signification de notes jetées rapidement d'une résidence de travail à l'autre – ces résidences pouvant être espacées de plusieurs mois) ; capacité à transmettre une régie à un autre technicien pour les tournées ou suite à un changement d'équipe ; communication et interopérabilité entre les différents corps de métier.
- absence d'outil permettant aux équipes de reprendre un spectacle à quelques mois ou quelques années de distance tout en respectant au plus près les intentions artistiques initiales. Or ceci est indispensable dans le cas des *digital performances* du fait de l'obsolescence rapide et programmée des technologies : il faut constamment adapter ces dernières pendant la tournée du spectacle. Qui plus est lorsque l'équipe de création n'est pas celle de la tournée, une documentation de ces spectacles est nécessaire et doit être rigoureuse.
- absence d'outil permettant au lieu d'accueil de ces spectacles de rassembler les différents documents nécessaires à leur diffusion : revue de presse, fiches techniques, biographies, programme, captation vidéo, etc. Aujourd'hui, tous ces documents arrivent de manière plus ou moins éparpillée.
- perte de mémoire : les *digital performances* sont très peu documentées. Il devient difficile d'en retracer l'histoire, d'autant plus que les documents qu'elles suscitent sont eux-mêmes éphémères. Or il existe une vraie demande du public pour avoir accès à des informations techniques, esthétiques, historiques, économiques, etc. sur ces œuvres.
- La captation vidéo des spectacles est aujourd'hui une pratique très répandue, avec des résultats inégaux quant à la qualité de sa réalisation. Réalisée et produite le plus souvent par la compagnie elle-même (pour des fins de promotion), elle n'en demeure pas moins l'une des traces les plus importantes de l'œuvre. Dans le cas des *digital performances*, parmi diverses difficultés, ce document a tendance à rendre illisible l'interactivité en temps réel : comment

[1] Bonardi Alain et Barthélémy Jérôme, « Le *Patch* comme document numérique : support de création et de constitution de connaissances pour les arts de la performance », in *Le Document numérique dans le monde de la science et de la recherche*, Actes du 10ème Colloque International sur le Document Numérique (CIDE), INIST, Nancy, 2007, p. 168.

différencier sur une bande vidéo ce qui relève de l'interactivité temps réel (par exemple un geste déclenchant un son ou modifiant une image), de la synchronisation d'un danseur avec des médias diffusés en temps différé ?

- Lorsque certains (rares) spectacles de ce type sont documentés (cf. par exemple les travaux autour de *9 Evenings* cités plus bas), il s'agit d'une interface créée spécifiquement. Elle n'est pas généralisable à d'autres spectacles et n'est pas conçue comme telle. Il s'agit d'une application et non d'un logiciel.

3. DE LA NOTATION…

Comme le rappelle Pip Laurenson [13], l'authenticité d'une œuvre d'art est traditionnellement évaluée dans le domaine de la conservation des arts plastiques en fonction de son intégrité physique, de son identité matérielle. La préservation consiste alors à maintenir l'œuvre–objet dans son état originel, ou du moins dans un état stable. Tout changement est alors considéré comme une perte ou une altération. Pip Laurenson constate que du fait du caractère éphémère et temporel des œuvres à composante technologique, la notion d'authenticité doit évoluer, et avec elle les mesures de préservation envisagées. Elle propose de substituer à la notion d'état celle d'identité, en s'appuyant sur le modèle musical fondé sur la partition, laquelle introduit la notion d'interprétation. La seule préservation de l'œuvre, au sens traditionnel du terme, c'est-à-dire dans ses composantes matérielles, ne saurait être suffisante : il faut aussi mettre en place une stratégie de documentation qui puisse permettre l'interprétation et l'adaptation à un nouveau contexte technologique. Tout comme dans la proposition récente de Richard Rinehart, c'est la notion de partition (et la référence à la musique) qui devient le paradigme pour préserver les œuvres d'art à composante technologique. Il faudrait donc créer un système de notation universel de ces œuvres qui permette de les interpréter avec différentes technologies.

Richard Rinehart a dirigé le programme de recherche « Archiving the Avant Garde : Documenting and Preserving Variable Media Art », lequel regroupe différentes institutions américaines (Les universités du Maine et de Californie, Rhizome.org, les archives de Franklin Furnace performance, New Langton Arts, et le Musée Whitney).

Pour Rinehart, l'art numérique est autant focalisé sur le processus et le comportement que sur l'artefact, ce qui le rapproche de la musique et des arts de la scène. C'est pourquoi il propose le modèle de la partition, autrement dit un système de notation libéré de l'environnement matériel de l'œuvre elle-même (y compris du code informatique) – un système qu'il souhaite aussi élaboré que la notation musicale. L'un des défis de ce système est de pouvoir décrire une œuvre non seulement comme un objet, mais aussi comme un événement. Rinehart propose un nouveau modèle conceptuel et descriptif : le Media Art Notation System (MANS), présenté en 2007 dans la revue *Leonardo* [18]. Un complément à MANS dédié aux arts de la scène et intitulé « Performance Art Documentation Structure » (PADS) a été développé par la suite par Stephen Gray [10].

MANS est une ontologie qui permet à la fois de décrire une œuvre de manière abstraite (son comportement, les interactions, etc.) et concrète. En effet, d'une part la « partition » permet théoriquement de re-créer l'œuvre avec des composants matériels et logiciels complètement nouveaux (du moment qu'ils respectent cette partition). Cette recréation est alors considérée comme une interprétation, de la même manière que tout concert est une interprétation d'une partition écrite. D'autre part, des éléments de MANS (comme « Part » ou « Resource ») permettent de décrire ces interprétations, ces occurrences de l'œuvre, voire d'en conserver certains éléments (fichiers, programmes…) et de donner des indications quant aux modes de préservation requis en s'appuyant sur la typologie des médias variables [5].

Rinehart constate que les institutions culturelles, les centres d'art, les fonds d'archives, les bibliothèques, etc., n'ont pas toujours les moyens de développer un système de catalogage. De plus, lorsque des catalogues sont développés, ils le sont pour des institutions, voire des collections spécifiques, mais aucune solution généralisable à toutes les collections et institutions n'est disponible à un coût modique. D'où le développement d'un système de catalogage, le Digital Asset Management Database (DAMD) », compatible avec les normes internationales, comme OAIS, et indépendant des pratiques spécifiques de telle ou telle organisation. La conformité avec les normes internationales est la garantie que ce système, téléchargeable gratuitement sur Internet, puisse être utilisé par différentes communautés et ensuite adapté en fonction des besoins spécifiques de chacune.

Dans le domaine des arts de la scène, le modèle de la partition musicale, ou d'un système de notation des œuvres, est loin de résoudre les problèmes de documentation : d'une part la partition musicale ne permet pas d'annoter la musique électroacoustique par exemple et l'utilisation de diagrammes ou de patches ; d'autre part, les outils de notation développés pour la danse (par exemple les notations Laban et Benesh) ne sont pas aussi généralisés que la notation musicale, et ne comprennent pas d'outils descriptifs permettant de rendre compte des dispositifs technologiques.

4. … A L'ANNOTATION

Plutôt que de développer un système de notation, nous proposons de développer un outil d'annotation. Devant la multiplicité des documents (parfois plusieurs centaines de fichiers, sans parler du *hardware*), il est bien sûr impossible de tout annoter. Dans un premier temps, il nous a semblé que deux stratégies étaient complémentaires : d'une part la création de liens entre

de multiples documents offre déjà la possibilité d'un commentaire et d'un éclairage des documents entre eux (création de bases de données relationnelles) ; d'autre part identifier le document principal qui serait l'épine dorsale à partir de laquelle tous les autres documents viendraient s'organiser. L'annotation, loin de s'opposer à la notation, permet de concilier à la fois la préservation des documents originaux (sous réserve que l'on puisse continuer à avoir accès au contenu de ces documents, problème qui est loin d'être résolu) et l'approche descriptive.

Une première expérience, antérieure à ce projet, a été menée par Clarisse Bardiot : la création d'un documentaire enrichi à partir des archives de *9 Evenings, Theatre & Engineering* pour la fondation Daniel Langlois à Montréal [1].

9 Evenings, Theatre & Engineering est une manifestation organisée par E.A.T. en 1966 à New York. Evènement majeur dans l'histoire des relations entre théâtre et technologie, comme dans l'histoire des nouveaux média, *9 Evenings* repose sur la collaboration de dix artistes (David Tudor, John Cage, Yvonne Rainer, Alex Hay, Deborah Hay, Robert Rauschenberg, Oyvind Fahlstrom, Steve Paxton, Robert Whitman, Lucinda Childs) et d'une trentaine d'ingénieurs de Bell Labs.

La recherche menée considère les différents aspects de la relation entre artistes et ingénieurs, ainsi que les technologies utilisées dans la manifestation. Elle s'articule autour des diagrammes publiés dans le programme. L'analyse de ces diagrammes, confrontée à des documents visuels (dont les captations tournées par Alfons Schilling), à des témoignages (interviews d'Yvonne Rainer, Robert Whitman, Deborah Hay et échanges avec des participants de *9 Evenings*), et à différents fonds d'archives (dont le fond John Cage à la New York Public Library for the Performing Arts et les archives de la Judson Church à la NYU), permet de comprendre la conception qu'a chaque artiste de la technologie. Cette analyse permet aussi de mettre en évidence comment la combinaison de mêmes éléments n'a pas imposé une esthétique identique pour toutes les performances. Enfin, elle permet de considérer *9 Evenings* comme l'une des toutes premières expériences qui applique des principes informatiques (bien que les technologies utilisées soient analogiques) dans le contexte du spectacle vivant. Cette manifestation est le précurseur des *digital performances* d'aujourd'hui.

La Fondation Daniel Langlois publie les résultats des recherches sur son site Internet, avec numérisation de certains de ses documents d'archive. La publication sur le web était l'occasion d'explorer une écriture spécifique pour ce support. Fragmentaire et discursive, ayant le désir de s'intégrer dans la structure de la base de données sur laquelle repose le site Internet de la Fondation Daniel Langlois, cette écriture est une tentative d'articulation et de tissage de textes courts, de documents d'archives numérisés, d'images, dont de nombreuses captures d'écran issues des archives filmiques (autant d'arrêts sur images qui permettent de mieux percevoir certains détails), et d'animations interactives des diagrammes (Figure 1).

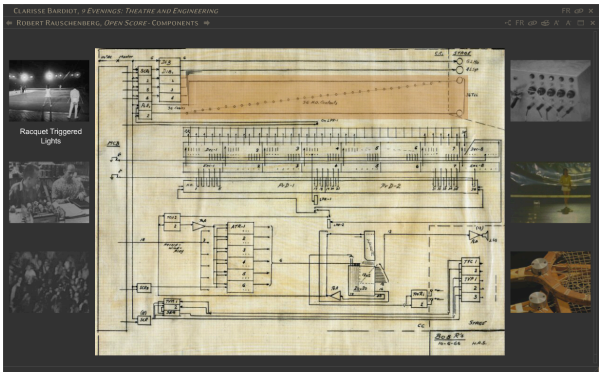


Figure 1. *9 Evenings, Theatre & Engineering*, site internet, capture d'écran.

Le cas de *9 Evenings* n'est pas isolé. Le numérique combiné aux réseaux offre la possibilité de s'approprier les archives : des outils spécifiques mis à disposition sur des plateformes en ligne permettent de les analyser, de les confronter, de tisser des liens sémantiques entre de multiples documents. Cet aspect est au cœur d'un programme européen, ECLAP, « e-library for performing arts » [7]. ECLAP réunit de nombreuses institutions européennes consacrées aux arts de la scène au sens large, contribue à la numérisation de leurs fonds (aujourd'hui plus d'un millions de documents, reliés à la bibliothèque numérique en ligne Europeana) et propose une série d'outils qui permettent à chacun d'enregistrer son propre parcours et d'annoter les documents sélectionnés via l'application MyStoryPlayer (Figure 2).

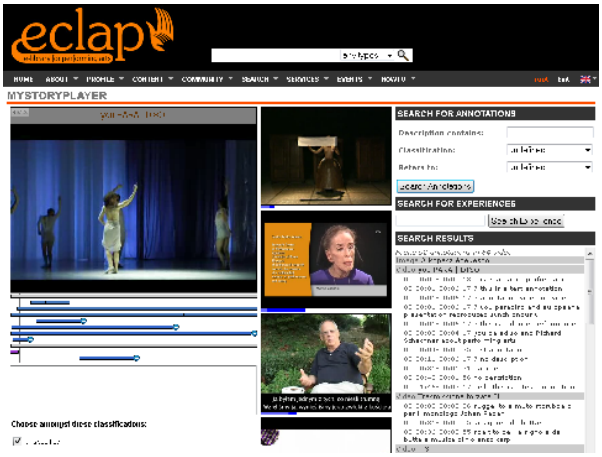


Figure 2. *MyStoryPlayer*, capture d'écran.

Une autre initiative, en Angleterre, propose de créer des carnets de notes personnalisés à partir d'un fonds d'archives disponible sur internet. Le Digital Dance Archives réunit différentes collections du National Resource Centre for Dance (NRCD) [6], dont la collection Laban, ainsi que le fonds d'archives de la chorégraphe Siobhan Davies. Des vidéos, mais aussi des photographies, des dessins, etc. sont accessibles au

public. Les documents retenus par chaque personne peuvent être annotés, classés et partagés (Figure 3).

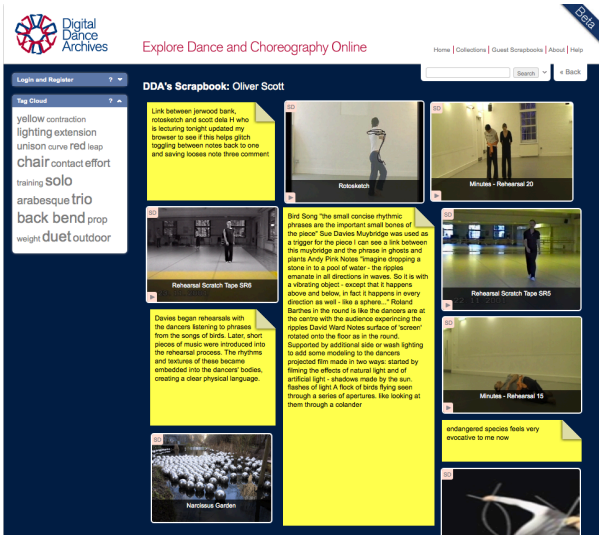


Figure 3. *Digital Dance Archive*, scrapbook, capture d'écran.

Par ailleurs, des chorégraphes et des équipes de recherche associées à un artiste ont engagé des projets de documentation de la danse contemporaine par les technologies numériques. Citons, de manière non exhaustive : Emio Greco, Siobhan Davies, Wayne McGregor, Steve Paxton, Pina Bausch (via le programme d'archivage de sa fondation), Myriam Gourfink, Jan Fabre, Deborah Hay, Bebe Miller, Thomas Hauert et bien sûr William Forsythe dont les projets sont devenus des références incontournables [11] [14]. *Improvisation Technologies, Synchronous objects* et *Motion Bank* embrassent une réflexion conduite sur vingt années, de 1993 à aujourd'hui, faisant de William Forsythe l'un des pionniers de ces questions [2]. Ces initiatives se sont développées sous l'impulsion de plusieurs phénomènes : l'impact considérable du CD-Rom *Improvisation Technologies* de William Forsythe ; l'attention portée aux processus de création à la fois par les artistes et par le public ; l'accessibilité de plus en plus grande aux outils numériques, notamment à la capture du mouvement ; l'intérêt croissant pour les différents modes de documentation et de transmission de la danse contemporaine, intérêt que l'on constate dans la multiplication des reprises des premières œuvres d'un chorégraphe (*Early Works 1966-1979* de Trisha Brown reprises en 2005, *Early Works* 1982-1987 d'Anne Teresa De Keersmaeker reprises en 2010). Ces initiatives reposent très largement sur l'annotation de documents, notamment vidéo. La plupart du temps, il s'agit de développements spécifiques à un chorégraphe, voire à une seule œuvre, la méthodologie de documentation développée étant intrinsèquement liée à la démarche de chaque artiste. Dans le cadre des projets mise en œuvre par Forsythe, une exception à cette individuation : *Piecemaker* (dont l'initiative revient à David Kern, l'un des danseurs de la compagnie William Forsythe) permet d'annoter textuellement les captations vidéo de répétitions ou de spectacles. Une version adaptée, PM2GO, est disponible en bêta depuis janvier 2014 (Figure 4). Cette solution (qui en est encore à ses balbutiements) a pour but d'annoter les vidéos de répétitions, en plaçant des commentaires à des moments précis. C'est un outil qui répond à un certain nombre de problématiques spécifiques, malheureusement essentiellement liées aux méthodes de travail de la compagnie de William Forsythe et donc trop fermées pour pouvoir convenir à beaucoup d'autres cas d'usages.

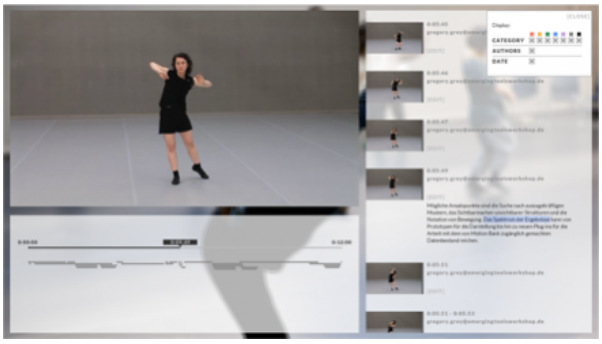


Figure 4. *PM2GO*, capture d'écran.

Le développement d'outils numériques spécifiques, tels que les possibilités d'annotation, montrent la difficulté à décrire les arts de la scène, y compris et surtout à partir des captations filmiques. Si la vidéo a semblé un temps une alternative aux différents systèmes de notation de la danse, alternative à la fois aisée à mettre en œuvre et bon marché, elle n'en demeure pas moins un document parcellaire qui appelle le commentaire. Aujourd'hui, les technologies numériques favorisent de nouvelles approches, entre partition et interprétation, entre notation et annotation.

5. PRINCIPES GENERAUX DE REKALL

Rekall [16] est un logiciel qui permet de documenter les *digital performances*, en prenant en compte le processus de création, la réception et les différentes formes d'un spectacle. Il s'adresse à la fois aux artistes, aux techniciens, aux chercheurs et au grand public. Simple d'usage et rigoureux (en particulier dans les méthodes d'indexation et la gestion des métadonnées), Rekall permet de nombreux usages, au-delà des *digital performances* et des arts de la scène. Pour réaliser ce projet, des structures culturelles (le phénix scène nationale Valenciennes, Le Fresnoy, MA scène nationale Montbéliard), une société (Buzzing Light – Guillaume Marais et Guillaume Jacquemin) ainsi que des institutions (Pictanovo, Ministère de la Culture et de la Communication) se sont associés. Le projet a été initié et conçu par Clarisse Bardiot, en collaboration avec Buzzing Light et Thierry Coduys.

Rekall offre une vision synthétique de la quantité, de la qualité et de l'organisation des documents entre eux, tout en étant au plus près de la démarche propre à chaque artiste, à chaque compagnie. Il permet à la fois

de rendre compte des technologies utilisées dans le spectacle et d'en offrir une description pour éventuellement proposer une alternative avec d'autres composantes. Il nous semble en effet primordial de garder la trace la plus précise possible des composantes technologiques, parce qu'elles sont également porteuses de dimensions esthétiques et historiques, tout en offrant la possibilité de décrire les effets de ces mêmes composantes, dans la lignée de la réflexion sur les médias variables [5].

Le fonctionnement de Rekall s'articule essentiellement autour des documents de création : croquis de scénographies, commentaires audio, description d'éléments techniques, vidéos, textes, carnets de notes, conduites techniques, patches, captures d'écran de logiciels spécifiques, partitions, photographies, mails… Il permet également d'articuler plusieurs strates temporelles : celle du processus de création (éclairer par exemple les recherches menées pour tel aspect du spectacle), de la représentation elle-même (voire de ses différentes versions dans le cas d'un *work in progress*), et de sa réception (par exemple en ajoutant des commentaires audio de la compagnie sur son propre travail, ou bien de spectateurs, ou encore la revue de presse).

L'accumulation des documents est un élément clé pour l'efficacité de fonctionnement de Rekall. En effet, c'est en analysant ces documents, en les mettant en relation les uns avec les autres et en les plaçant dans des contextes soigneusement choisis (multidimensionnels, temporels ou non) que Rekall parvient peu à peu à révéler des caractéristiques spécifiques à une œuvre donnée.

Afin de recueillir tous ces documents de travail, il est essentiel que Rekall se trouve au cœur du processus de création. La majeure partie des documents doivent être naturellement implémentés sans représenter une charge de travail supplémentaire pour les artistes. C'est pourquoi une grande partie de Rekall est dédiée à l'organisation des documents de création pendant le processus créatif. Il permet à tous les protagonistes intervenant au cours du processus de création de travailler sur une plateforme commune et compartimentée.

Cette structure ouvre alors un spectre de possibilités analytiques extrêmement important. En partant du principe qu'un processus de création peut être analysé en grande partie par les documents de création collectés (devenus documents d'exploitation pour certains), Rekall se base sur les *métadonnées* présentes dans chacun de ces documents pour en extraire des informations cruciales (auteur, date de création, lieu de création, etc.) qui seront ensuite utilisées par les outils d'analyse et de représentation de l'information, pour révéler des comportements créatifs, des usages ou d'autres informations insoupçonnées.

Au vu de la masse d'informations que véhicule une œuvre, il apparaît évident qu'une solution d'export ciblé est nécessaire afin de n'inclure dans différents packages que les documents utiles à l'usage souhaité (pédagogie, exploitation, critique, etc.). Connaissant la nature de chaque document, Rekall permet de configurer simplement ces exports, qui auront également le mérite de valoriser l'œuvre (documents à jour, présentation soignée, etc.).

La préservation passive d'une œuvre d'art à composante technologique est dans la plupart des cas malheureusement irréalisable. Tout comme nous remplaçons ou renouvelons les éléments périssables essentiels à certaines œuvres d'art plastique, il est indispensable de préserver activement les œuvres d'art à composante technologique. Même si Rekall répond en partie au problème d'obsolescence des technologies (nomenclatures claires, mises à jour des outils embarqués, etc.), il est indispensable que certaines actions de préservation soient réalisées par l'utilisateur. Rekall simplifie cette préservation active en alertant par exemple lorsque la pérennité d'un document est mise en danger par la dernière mise à jour de son logiciel d'exploitation.

Les captures d'écran qui suivent sont issues d'une étude de cas en cours sur le projet *Re :Walden* créé par le metteur en scène Jean-François Peyret et qui prend différentes formes de 2009 à 2014 (workshops, installations, spectacles, concert) [15].

5.1. Projet et documents

Une œuvre dans sa globalité (création, répétitions, représentations, performances, itérations, régie, etc.) est structurée sous la forme d'un projet Rekall, constitué par l'ensemble des documents qui ont été nécessaires à la fabrication de l'œuvre. Les documents peuvent être de quatre natures différentes (Figure 5) :

- un fichier (patch, texte, vidéo, audio…),
- une URI (lien web, site internet, fichier online, ressource logicielle…),
- un acteur du projet (auteur, artiste, régisseur, réalisateur informatique musicale…),
- un *cue* ou marqueur (un timecode précis associé à une fonction dans l'œuvre).

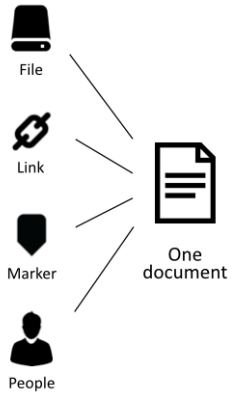


Figure 5. Polymorphisme des documents

Ces documents sont systématiquement transcodés dans le format le plus pérenne disponible et connu par le

logiciel : PDF pour les documents, H264 pour les vidéos, etc. Aussi, dans la mesure du possible, un aperçu du document (vignettes pour les images et PDF, pellicules pour les vidéos, formes d'ondes pour l'audio) est généré. Ces opérations sont effectuées à l'aide des outils *open source cross platform ffmpeg* [1] et *lame* [12].

5.2. Versions et métadonnées

Chaque document peut posséder plusieurs versions qui représentent le travail itératif d'un acteur du projet. À l'aide des mécanismes offerts par les systèmes d'exploitation, Rekall est capable d'analyser en temps réel les modifications des documents qu'il surveille, sans ralentissement pendant le travail. Sauf demande explicite de l'utilisateur, le logiciel ne crée pas de copie physique des versions des documents mais enregistre simplement le nom de la personne qui l'a modifié associée à un horodatage (Figure 6).

Figure 6. Versionning des documents

Dans le cas des documents de type *fichier*, Rekall identifie également les doublons à l'aide d'une fonction classique de hachage sur le contenu du fichier.

Pour chaque version d'un document, Rekall extrait l'ensemble des métadonnées disponibles à l'aide de l'outil *open source* cross platform *exiftool* [8]. Ce set de métadonnées est ventilé en catégories (informations relatives à l'audio, à la vidéo, au fichier, à l'auteur…) et sont stockées dans une table de hachage optimisée pour l'accès en lecture (Figure 7).

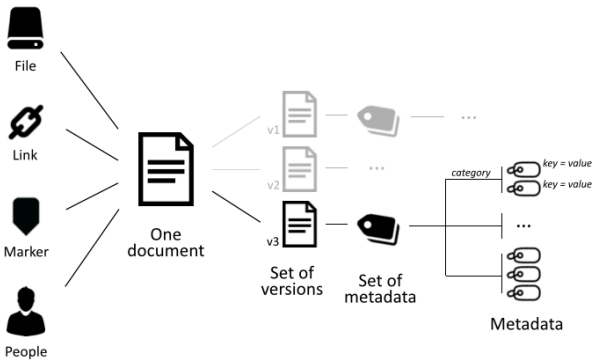


Figure 7. Processus d'analyse des documents

À l'ajout ou modification d'un document, Rekall enregistre aussi des métadonnées contextuelles telle que l'auteur de l'import ou de la modification, l'horodatage, les caractéristiques techniques de l'ordinateur et la géolocalisation. Ces informations seront très utiles pour la reprise de l'œuvre en cas d'incompatibilité technique *(« sur quelle version de Mac OS travaillait-on ? »)*, de perte de fichier (*« qui a fait la dernière mise à jour du fichier ? »*), de configuration technique oubliée (*« sur quelle machine tournait le patch ? »*), etc. (Figure 8).

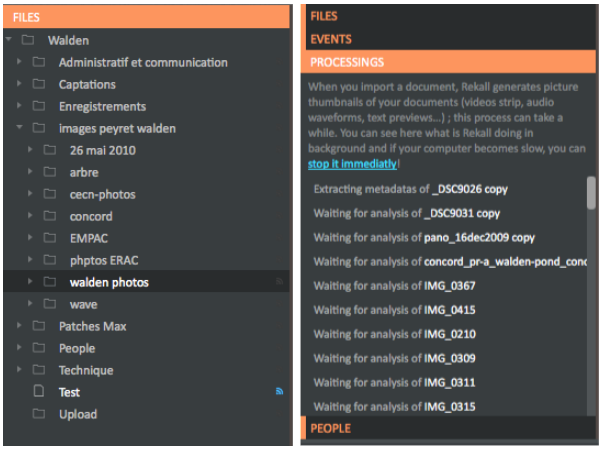


Figure 8. Organisation d'un projet avec ses documents et l'analyse automatique en tâche de fond

Dans la suite du travail de création avec Rekall, toutes les opérations effectuées par l'interface graphique (positionnement d'un élément dans la timeline, liens entre documents, etc.) sont stockées comme une métadonnée dans les documents (Figure 9).

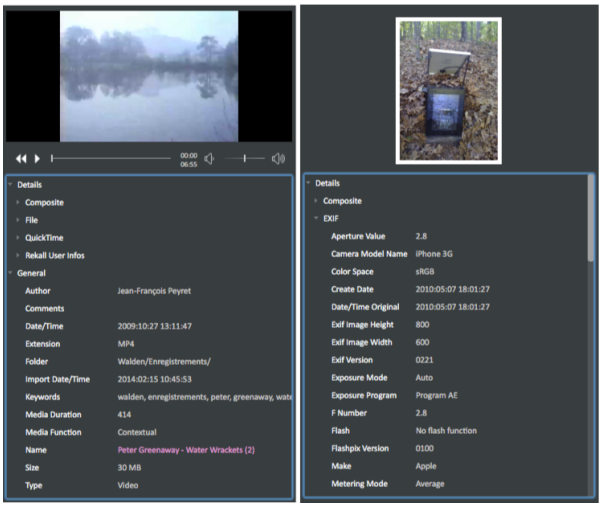


Figure 9. Exemple de métadonnées automatiquement extraites par Rekall

5.3. Data visualisation

Une fois l'ensemble des documents analysés et enrichis de leurs métadonnées, Rekall propose un mécanisme extrêmement simple mais puissant de représentation de l'information : chaque document est visualisé dans un espace 3D, sous la forme d'une barre dont la visibilité, la position, la couleur, le texte et l'enveloppe sont entièrement fonction des métadonnées du document qu'elle représente ; il s'agit du *mapping* des métadonnées sur des paramètres graphiques (Figure 10).

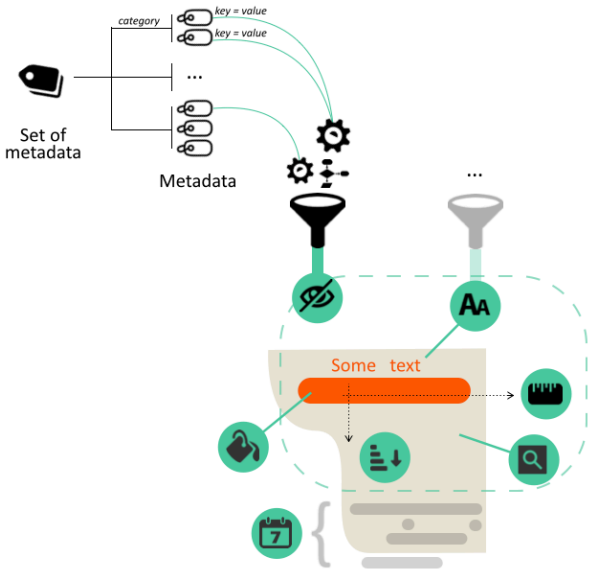


Figure 10. Principe de mapping des métadonnées sur la représentation graphique

Lorsqu'une métadonnée n'est pas une valeur numérique (texte ou date), Rekall crée des groupes afin de positionner les métadonnées dans une échelle de catégories (texte similaire ou intervalle de dates).

Pour chaque représentation graphique possible (position, couleur, texte…), une barre de filtrage permet de configurer finement le mapping : choix du critère (métadonnée) et activation/inhibition de certaines valeurs (Figure 11).

Figure 11. Barre de filtrage et de configuration du mapping des métadonnées

Au delà de la représentation isolée de chaque document, il est possible de constituer un groupe graphique de documents suivant les mêmes critères cités précédemment. On peut ainsi révéler l'évolution d'un mot-clef, d'un acteur du projet, d'un concept, etc. au fil du temps (Figure 12).

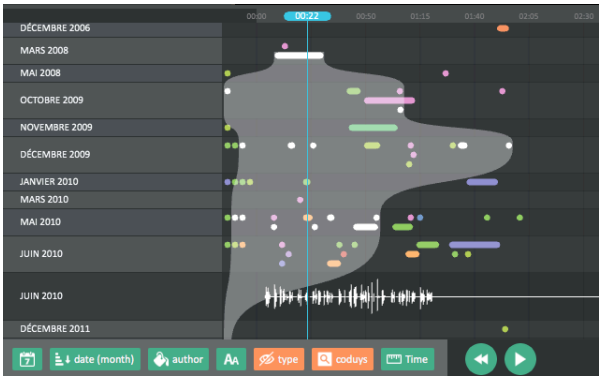


Figure 12. Groupe de documents possédant le mot-clef *Coduys*

Lorsqu'un fichier possède plusieurs versions ou doublons, ils apparaissent reliés par un fil, permettant d'identifier graphiquement les changements historiques sur un document (Figure 13).

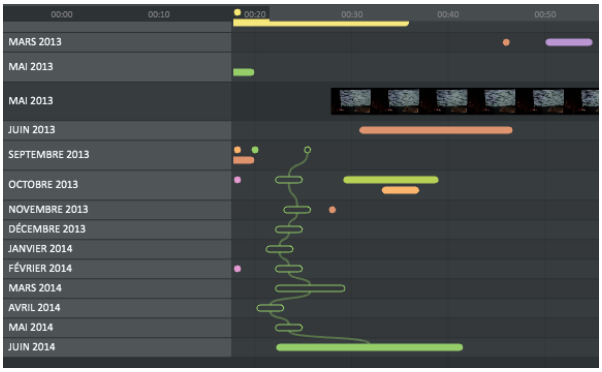


Figure 13. Évolution d'un document au fil du temps
Forme pleine = version actuelle
Formes vides = anciennes versions

5.4. Exemples de représentations

Quelques exemples de représentations utilisant des paramètres simples : la date, le type et l'auteur du document et son implication dans le déroulé de l'œuvre (timeline) (Figures 14 à 17).

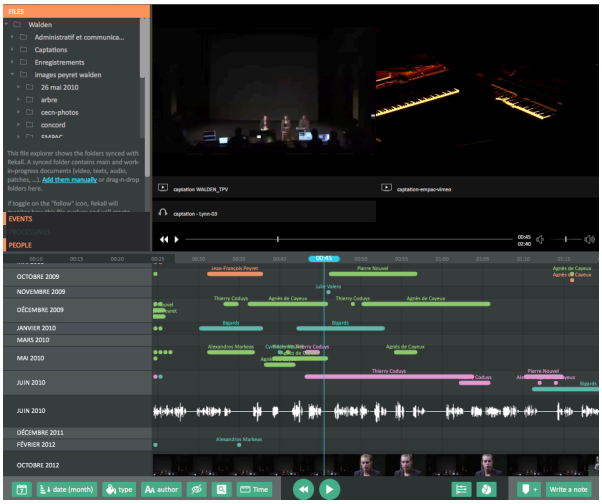


Figure 14. Étude chronologique des créations et modifications des documents de l'œuvre
axe vertical : mois du calendrier ; couleur : type de document
texte : auteur du document ; axe horizontal : timeline de l'œuvre

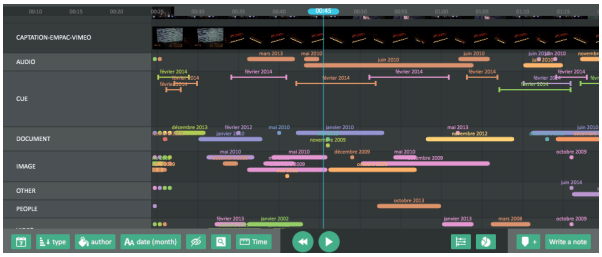


Figure 15. Étude des types de documents utilisés dans le déroulé de l'œuvre
axe vertical : type de document ; couleur : auteur du document
texte : mois du calendrier ; axe horizontal : timeline de l'œuvre

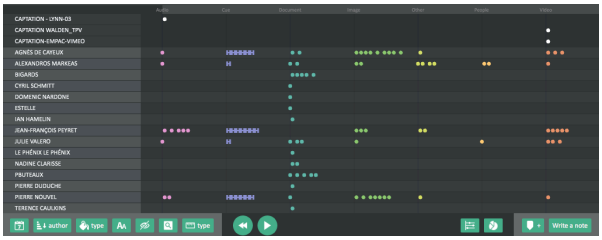


Figure 16. Étude des types de documents produits par les différents acteurs du projet
axe vertical : auteur
axe horizontal + couleur : type du document

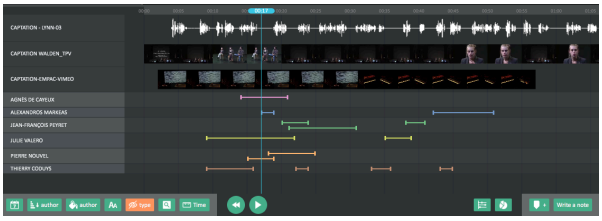


Figure 17. *Cues* des différents acteurs du projets
axe vertical + couleur : auteur
axe horizontal : timeline de l'œuvre
filtre : n'afficher que les documents de type *cue*

5.5. Régie

Cette dernière figure révèle que le logiciel peut gérer la régie générale du projet. Il s'agit alors d'utiliser Rekall comme un séquenceur capable d'envoyer en temps réel les *cues* (événement ponctuel ou début/fin d'un événement) en OpenSoundControl. À l'aide d'un simple filtrage, chaque acteur du projet peut utiliser Rekall pour visualiser uniquement sa régie (Figure 18).

Figure 18. *Cues* de Thierry Coduys

Une représentation graphique dédiée à la temporalité de l'œuvre est alors disponible et permet de visualiser en liste l'ensemble des documents et *cues* filtrés précédemment.

Un bouton spécial assorti d'un raccourci clavier permet à tout moment de placer un *cue* dans le projet (Figure 19).

Figure 19. Visualisation de la timeline en liste
À gauche sans filtre sur l'auteur. À droite, uniquement Thierry Coduys. Le chronomètre à gauche de l'item permet de d'anticiper l'apparition imminente du cue.

5.6. Documentaire interactif

À l'aide de toutes les représentations présentées dans ces dernières sections, Rekall constitue aussi un outil interactif pour les documentaires augmentés (Figure 20). Par exemple, en combinant la timeline de l'œuvre, la captation d'une performance et la représentation en liste (utilisée précédemment pour la régie), il est possible de

visualiser en temps réel les documents de création pendant le visionnage d'une œuvre, mais aussi de conserver un regard sur sa globalité.

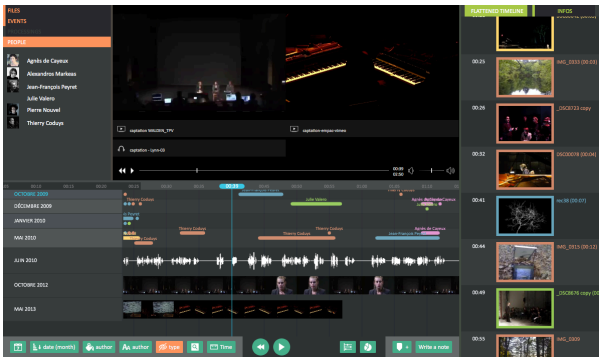


Figure 20. Documentaire augmenté

Rekall est capable de lire plusieurs médias simultanément, ce qui permet entre autres de comparer plusieurs versions d'un œuvre, plusieurs représentations ou encore d'ajouter des médias narratifs (commentaires de l'artiste, explication de l'interprète…). L'utilisateur peut à tout moment activer ou désactiver la vidéo et l'audio indépendamment, ajuster le volume et naviguer dans la timeline (Figure 21).

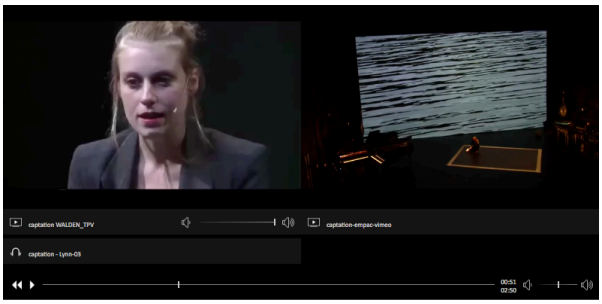


Figure 21. Visionnage simultané de deux représentations associées à une bande son supplémentaire

Enfin, un mode multi-moniteurs permet également un confort de travail et d'analyse en déportant la vidéo en plein écran sur un ou plusieurs écrans secondaires.

5.7. Choix techniques

Rekall est développé avec le *framework* de développement Qt. Ce choix a été fait pour les raisons suivantes :

- *framework open source*, compatible avec des modèles de licences types GPL ;
- *cross-platform*, qui peut s'exécuter sur les trois principaux systèmes d'exploitation Linux, Mac OS et Windows ;
- intégration native d'OpenGL, pour le rendu 2D et 3D de manière fluide et interactive ;
- interfaces graphiques riches avec une expérience utilisateur intuitive, élégante et efficiente.

Rekall est distribué sous licence GPL3 et son code source est maintenu sur un repository public [17].

6. CONCLUSION PROVISOIRE

Rekall est actuellement en version alpha. La bêta est prévue pour avril/mai 2014. La collaboration avec des équipes artistiques dans cette phase d'expérimentation est essentielle. En effet, le logiciel est conçu pour les artistes et leurs équipes techniques afin qu'ils soient à même de documenter leur propre processus de création et les œuvres créées. C'est pourquoi nous nous appuyons sur la collaboration de deux équipes artistiques, en théâtre (Jean-François Peyret [15]) et en danse (Mylène Benoit [3]), en résidence respectivement au Fresnoy et au phénix scène nationale Valenciennes. Ces équipes sont d'une part associées à la conception du logiciel et d'autre part les premiers utilisateurs. Des réunions de travail avec les différents intervenants (techniciens, régisseurs, metteur en scène, chorégraphe, éclairagiste, vidéaste) font partie du processus de conception et développement de Rekall, afin d'ajuster régulièrement le cahier des charges et les spécifications aux besoins des futurs utilisateurs. Plusieurs workshops avec différents types d'utilisateurs (artistes, techniciens, chercheurs) sont également prévus afin d'évaluer et éventuellement corriger certains aspects méthodologiques, comme le tracking des actions des différents contributeurs à une œuvre, ou encore le choix d'un modèle qui puisse s'adapter à de nombreuses œuvres. En effet, il nous semble que Rekall, conçu à l'origine pour des œuvres scéniques, peut également être utilisé pour des installations plastiques. Concernant les œuvres musicales, l'intégration d'un suivi de partition doit également être étudié. Enfin, nous évaluons le développement d'une version simplifiée *online* pour des usages simples ou des contextes légèrement différents, comme la critique artistique ou encore la phase de conception et d'écriture des œuvres.

7. REFERENCES

[1] Bardiot C. *9 Evenings, Theatre & Engineering*, site Internet de la Fondation Daniel Langlois, mai 2006, http://www.fondation-langlois.org/flash/f/index.php?NumPage=571 Consulté le 01/04/2014.

[2] Bardiot C. « Une autre mémoire : la chorégraphie des données. À propos des objets numériques développés par William Forsythe (*Improvisation Technologies, Synchronous objects* et *Motion Bank*) », in *Documenter, recréer… Mémoires et Transmissions des œuvres performatives et chorégraphiques contemporaines*, *Les Presses du réel*, à paraître.

[3] Benoit Mylène, site internet de la compagnie Contour Progressif, http://www.contour-progressif.net/ Consulté le 01/04/2014.

[4] Bonardi, A., Barthélémy J. « Le *Patch* comme document numérique : support de création et de constitution de connaissances pour les arts de la performance », *Le Document numérique dans le monde de la science et de la recherche, Actes du 10ème Colloque International sur le Document Numérique (CIDE),* INIST, Nancy, 2007, p. 163-174.

[5] Depocas A., Ippolito J., Jones C. (sous la direction de). *L'Approche des médias variables. La permanence par le changement*. Guggenheim Museum Publications et Fondation Daniel Langlois, 2003. Publié sur Internet : http://variablemedia.net Consulté le 01/04/2014.

[6] Digital Dance Archives, http://www.dance-archives.ac.uk/ Consulté le 01/04/2014.

[7] ECLAP, http://www.eclap.eu Consulté le 01/04/2014.

[8] ExifTool, http://sno.phy.queensu.ca/~phil/exiftool Consulté le 01/04/2014.

[9] FFMpeg, http://www.ffmpeg.org Consulté le 01/04/2014.

[10] Stephen G., « Conservation and Performance Art: Building the Performance Art Data Structure (PADS) », MA dissertation, 2008, http://www.incca.org/resources/38-documentation/255-graydissertation Consulté le 01/04/2014.

[11] *International Journal of Performance Arts & Digital Media*, « Choreographic documentation », vol. 9, n° 1, 2013.

[12] Lame, http://lame.sourceforge.net Consulté le 01/04/2014.

[13] Laurenson, P. « Authenticity, Change and Loss in the Conservation of Time-Based Media Installations », *Tate Papers*, n° 6, automne 2006, http://www.tate.org.uk/research/publications/tate-papers/authenticity-change-and-loss-conservation-time-based-media Consulté le 01/04/2014.

[14] *Performance Research*, « Digital Resources », 11:4, 2007.

[15] Peyret Jean-François, site de la compagnie, http://theatrefeuilleton2.net Consulté le 01/04/2014.

[16] Rekall, site internet, http://www.rekall.fr, Consulté le 01/04/2014.

[17] Repository de Rekall, https://www.github.com/Rekall/Rekall Consulté le 01/04/2014.

[18] Rinehart, R. « A System of *Formal Notation for* Scoring Works of Digital and Variable Media art », *Leonardo - Journal of the International Society for the Arts, Sciences and Technology*, The MIT Press, Volume 40, n° 2, 2007, p. 181-187.

DISSÉMINATION ET PRÉSERVATION DES MUSIQUES MIXTES : UNE RELATION MISE EN ŒUVRE

Guillaume Boutard
Université de Montréal
Centre interdisciplinaire de recherche en musique, médias et technologie
guillaume.boutard@mail.mcgill.ca

Fabrice Marandola
McGill University
Centre interdisciplinaire de recherche en musique, médias et technologie
fabrice.marandola@mcgill.ca

RÉSUMÉ

Le projet Documentation, Dissemination and Preservation of Compositions with Real-time Electronics (DiP-CoRE) a proposé d'articuler dissémination et préservation des œuvres de musique mixte au travers de la documentation des processus de production, prenant en compte les différentes expertises engagées, que ce soit au niveau de la composition, de l'interprétation, ou de l'ingénierie et plus fondamentalement encore de leurs articulations. Cet article détaille les aspects méthodologiques du projet, basés notamment sur des modèles théoriques en sciences de l'information, et les aspects empiriques, basés sur des conceptualisations inductives des processus de production et de création, et leur application pratique au travers d'exemples choisis collectés pendant le travail effectué par l'ensemble de percussion Sixtrum avec les compositeurs Robert Normandeau, Serge Provost, et Laurie Radford.

1. INTRODUCTION

Rappelons que l'utilisation de nouvelles technologies a mis le répertoire des musiques mixtes en danger du point de vue de sa pérennité. Cette problématique a été mise en avant dans de nombreux articles et dans un cadre musical qui ne se limite pas aux musiques mixtes (on prendra comme exemple [2], [17], ou encore [16]). Baudouin [1] reliait cette question, dans le cadre des musiques électroniques, à celle de la préservation d'une mémoire collective : « l'enjeu de sa conservation dépasse amplement celui des œuvres considérées de façon isolée, et concerne davantage une mémoire collective qu'il devient urgent, à mesure que le temps s'écoule, de préserver » (p. 1). Plus pragmatiquement, Tiffon [19] rappelait que pour les musiques mixtes en temps réel, « l'œuvre est dépendante d'une technologie tributaire d'une époque, ou même d'une technologie spécifiquement créée pour les desseins propres au compositeur » (p. 27).
C'est bien la problématique de l'idiosyncrasie des techniques, souvent présentée comme problème (e.g. [12]), qu'il convient de mettre en avant dans la recherche et tout spécialement dans sa relation à l'intelligibilité des œuvres, ce qui permettait à Baudouin d'avancer qu' « en préservant cette mémoire, menacée de disparition partielle à moyen terme, et en réunissant un maximum de sources, nous avons le sentiment d'avoir contribué à la conservation d'un patrimoine, conservation assurée non seulement au moyen du fonds, mais aussi par la mise en lumière de ses clés de lecture » .
La question de l'intelligibilité et de la dissémination avait notamment était mise en évidence par Bernardini et Vidolin [2], dans le cadres des musiques électroacoustiques, qui considéraient la nécessité de développer des « active communities of co-operating performers which will be conscious enough to share and document their experiences ». Mais cette question de l'intelligibilité ne peut être séparée des processus de production et des processus créatifs puisque comme le remarquait justement Born [4] dans le cadre de la production à l'Institut de Recherche et de Création Acoustique/Musique (IRCAM), "we have seen that programs are often developed over time through the collaborative imaginative labor of several authors. Because of this inherent temporal and social mediation, the resultant baroque totality is extremely difficult to decode after the event and is thus opaque to the reconstruction of its total logic – the necessary prerequisite for documenting it » (p. 276).
C'est depuis ce point de départ que s'est construit le projet Documentation, Dissemination and Preservation of Compositions with Real-time Electronics (DiP-CoRE), projet visant à développer des méthodes de préservation du répertoire centrées sur l'intelligibilité, tout en actant la dimension fondamentalement idiosyncratique de chaque processus créatif et de chaque production. C'est donc dans la mise en œuvre du rapport entre dissémination et préservation par la médiation de la documentation que le projet se place dans le domaine de la pérennisation du répertoire des musiques mixtes.

2. CONTEXTE DE PRODUCTION

Le projet DiP-CoRE s'est monté autour de la création par l'ensemble de percussion Sixtrum de trois œuvres mixtes de trois compositeurs de renommée internationale : Robert Normandeau, Serge Provost et Laurie Radford. L'ensemble Sixtrum composé de six percussionnistes—João Catalão, Julien Compagne, Julien Grégoire, Philip Hornsey, Kristie Ibrahim, Fabrice Marandola—était complété

par quatre chanteurs—Stéphanie Lessard (soprano), Marie-Annick Béliveau (alto), Michiel Schrey (ténor), Pierre-Étienne Bergeron (basse). Les concerts de création s'incrivaient dans la série organisée par le Centre Interdisciplinaire de Recherche en Musique, Médias et Technologie (CIRMMT), intitulée live@CIRMMT, et se sont déroulés les 15 et le 16 mai 2013 dans la salle du MMR de l'université McGill. Ces concerts ont été précédés d'une longue phase de production dont l'un des hauts points fut un atelier sur la spatialisation à l'université McGill, suivant une phase de travail entre compositeurs et instrumentistes et aboutissant à une performance des œuvres en l'état le 20 décembre 2012. Ces conditions de production, décrites par les trois compositeurs comme fondamentales, ont permis d'avoir accès à un niveau important de traces de l'activité collective de production d'une œuvre nécessaire à la mise en œuvre de nouvelles méthodes de documentation, de dissémination et par voie de conséquence de préservation du répertoire des musiques mixtes.
Les données brutes d'observation audio et vidéo fournissant la base à la partie méthodologique de la documentation discutée dans cet article ont été recueillies depuis l'atelier du 20 décembre 2012 jusqu'au deux concerts des 15 et le 16 mai 2013.

3. MÉTHODE

La préservation des musiques mixtes est une problématique interdisciplinaire et l'on peut la présenter de multiples façons, ainsi que les différentes tendances dans chaque axe, sans la trahir. Une possibilité est de distinguer les approches dans leur rapport à la technologie, c'est-à-dire entre des approches technocentriques (la technologie à la rescousse de la technologie—La technologie est ici comprise comme technique fondée sur le savoir scientifique, elle n'est donc pas entendue comme simple implémentation) et des approches sociotechniques (prise en compte des agencements entre humains et objets techniques). Une autre possibilité est de mettre en parallèle la vision traditionnelle de la préservation axée autour de l'organologie, la pratique et la partition, et la vision divergente axée autour de la documentation des œuvres et des processus créatifs dans leur singularité. C'est dans cette deuxième démarche que s'inscrit le projet DiP-CoRE sans toutefois considérer la première vision comme fondamentalement invalide dans le contexte des musiques mixtes, mais plutôt complémentaire.
Le point important est que chacune de ces visions est soustendue par un système de contraintes différentes. Si dans la vision traditionnelle les systèmes de contraintes que sont l'organologie, la pratique et la partition sont à l'origine d'écrits nombreux, ce n'est que plus récemment que les cadres de collectes de données en rapports avec les processus créatifs ont été l'objet de discussions méthodologiques. Ces discussions se sont développées en liaison avec certaines tendances en musicologie empirique, on prendra comme exemple le travail de théorisation de Theureau [18] qui nous présente ces contraintes méthodologiques dans le contexte de la technique du 'cours d'action'. Theureau les présente en rapport à plusieurs autres tendances dans les techniques d'enquêtes plus ou moins liées aux traces de l'activité, notamment le travail de Vermersch [20] sur les 'entretiens d'explicitation' et le travail de Clot [8] sur les 'autoconfrontation croisées'. Chacune de ces méthodologies se base sur des systèmes de contraintes différents qui requièrent notamment une implication plus ou moins importante du chercheur dans la direction de la production de données. Comment consigner une expérience, quels sont les partis pris ontologiques et épistémologiques ? Toutes ces questions sont héritées des disciplines mises à contribution, que ce soit l'ergonomie [18], la linguistique [7] ou la psychologie [8].
Le projet DiP-CoRE étant orienté sur cette consignation des expériences et de transmission entre experts, il était intéressant de s'inscrire principalement dans une approche permettant à l'expertise même d'être le cadre de référence, c'est-à-dire de fournir le système de contraintes. Comme le dit Goasdoué [11], dans un cadre réduit par rapport au projet DiP-CoRE, « [...] il n'appartient pas aux scientifiques de déterminer ce qu'est une bonne interprétation » (p. 91). Comment donc intégrer la pratique dans la méthodologie de collecte de données sans trahir l'expertise ? S'il faut se baser sur l'expertise pour consigner les connaissances en jeu et la tradition orale qui participe à la construction des objets que sont les musiques mixtes, alors il devient nécessaire d'établir des méthodes qui permettent de refaire surgir cette construction experte dans les méthodes de documentations sans réduire ces connaissances à un cadre formel défini a priori. Dans ce contexte, les autoconfrontations croisées proposées par Clot et leur corollaire dans un cadre organisationnel multi-expertise, c.-à-d. la méthode des 'dialogical mediated inquiries' proposée par Lorino et al. [13], permettaient au mieux de répondre aux objectifs du projet et à l'objet de recherche, le tout autour du cadre théorique fourni par ce que Boutard [5] nomme 'mixed methods digital curation', c'est-à-dire une façon d'envisager la conservation des objets numériques du point de vue pragmatique de la mise en commun de plusieurs méthodes aux postulats épistémologiques différents.
Le travail de Clot et Faïta [9] prend comme fondement la notion d'épaisseur de l'activité, c'est-à-dire le fait que « [...] l'activité n'est plus limitée à ce qui se fait. Ce qui ne s'est pas fait, ce qu'on voudrait faire, ce qu'il faudrait faire, ce qu'on aurait pu faire, ce qui est à refaire et même ce qu'on fait sans vouloir le faire est accueilli dans l'analyse de l'activité en éclairant ses conflits » (p. 35). Dans le cadre multi-expertise de la production des musiques mixtes tel que présenté notamment par Berweck [3], nous avons donc inclus dans le recueil de données non seulement les compositeurs et les instrumentistes, mais aussi les ingénieurs du son et réalisateurs en informatique musicale.
La segmentation des données était régie par le travail théorique de Boutard [5] et plus précisément les cadres conceptuels fournis par Boutard et Guastavino [6], cadres tou-

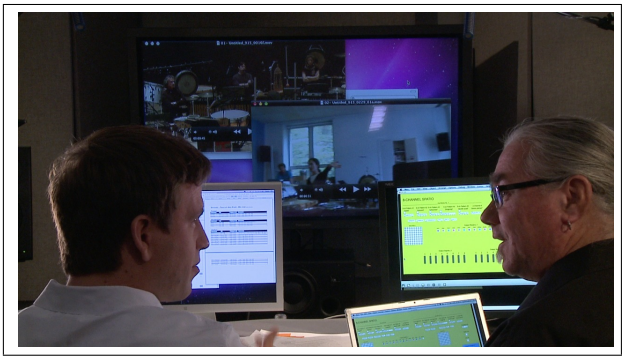


Figure 1. Dispositif d'entrevue du projet DiP-CoRE avec les traces de l'activité. Sur l'écran du fond, des extraits des captures vidéos pendant la production ; à droite le patch Max/MSP du compositeur Laurie Radford ; à gauche des documents textuels et graphiques récoltés pendant et après la production.

jours pris dans un processus d'adaptation aux nouveaux contextes. Cette segmentation a permis d'écarter les observations non pertinentes au processus de production. C'est ainsi que par la suite, ces données brutes d'observation segmentées ont été passées en revue et sélectionnées par un panel d'experts en fonction de leur utilité en temps qu'objet de documentation et/ou objet de médiation (c'est-à-dire permettant d'approfondir un point jugé pertinent par le panel) pour la phase suivante des entrevues. Ces documents vidéo, considérés comme objets de médiation, ainsi que les documents de travail fournis par les agents du processus de production ont fourni le matériel nécessaire à la constitution de deux types d'entrevues : d'une part les autoconfrontations croisées sur la base du travail en psychologie de l'activité de Clot [8] et leur application au contexte multi-expertise [13], et d'autre part des entrevues avec les compositeurs dans un dispositif semblable à celui de Donin et Theureau [10], c'est-à-dire de confrontation simple avec les traces d'activité (notamment les documents de travail, patches, et observations vidéo).

4. MISE EN ŒUVRE

Les données récoltées à la fin du projet comprennent donc : des observations vidéos enregistrées pendant les séances de répétitions, des documents de travails et esquisses, des entrevues, les enregistrements de concert. Le tout sera publié dans un DVD qui sera disséminé par le label *empreintes DIGITALes* [14] et comprendra également des enregistrements vidéos des trois œuvres, *Le rêve d'Ahmed* (Serge Provost), *Baobabs* (Robert Normandeau) et *The Body Loop* (Laurie Radford).

Quantitativement les données récoltées comprennent (les valeurs sont approximatives et divisées suivant le nombre de caméras utilisées à chaque étape) :

Traces vidéo

- Atelier de décembre 2012 : 2h20mn
- Répétitions Université de Montréal : 9h
- Répétitions Université McGill et performances : 7h30mn

Documents divers (comprenant notamment des photos non compressées de la disposition technique et instrumentale, des fichiers sons volumineux) : 4.5 GB.

Entrevues en fonction des œuvres (les valeurs sont approximatives et divisées suivant le nombre de caméras utilisées)

- Le rêve d'Ahmed (Serge Provost) : 2h50mn
- Baobabs (Robert Normandeau) : 1h45mn
- The Body Loop (Laurie Radford) : 1h10mn

Pour la production du DVD ont été sélectionné, en accord avec la méthodologie décrite et les contraintes d'espace, au niveau vidéo (valeurs approximatives) :

Le rêve d'Ahmed (Serge Provost) : 7mn d'observations vidéo et 43mn d'entrevues

Baobabs (Robert Normandeau) : 15mn d'observations vidéo et 15mn d'entrevues

The Body Loop (Laurie Radford) : 15mn d'observations vidéo et 20mn d'entrevues

Le dispositif de récolte de données par entrevues comprenait plusieurs écrans permettant d'afficher des documents de travail électronique, les patchs, et de rejouer les segments vidéo provenant des enregistrements des sessions de répétitions. Suite à la lecture des séquences vidéo il était demandé aux acteurs de la production sélectionnés de commenter l'activité affichée dans les séquences tout en s'aidant des autres documents présents si nécessaire. La figure 1 montre le compositeur Laurie Radford à droite et l'ingénieur du son Padraig Buttner-Schnirer à gauche, devant le dispositif d'entrevue comprenant comme traces de l'activité le patch Max/MSP (écran de droite), la liste des microphones (écran de gauche), et les enregistrements vidéo (écran central). Afin d'exemplifier le types de données récoltées nous prendrons quelques exemples liés à la taxonomie établie par Clot à propos de l'épaisseur de l'activité.

Sur la figure 2, l'instrumentiste Fabrice Marandola (à gauche), en compagnie du compositeur Robert Normandeau (à droite), utilise les documents fournis, ici la photographie de la disposition instrumentale sur scène, pour expliciter les choix fait en rapport avec la distance aux haut-parleurs :

– Fabrice Marandola (Percussionniste) : « Tu vois, si tu regardes là [FM montre du doigt l'image à l'écran, cf. figure 2] sur les images, il y a quand même du monde, il y a de la place de chaque coté. Et au début ce n'était pas ça... ».
– Robert Normandeau : « ... vous étiez jusqu'au bout ».
– FM : « Mais en même temps on avait beaucoup plus l'aile gauche, l'aile droite, donc l'effet stéréo [...] »

L'épaisseur de l'activité permet de rendre compte de ce qui aurait pu être fait par comparaison à ce qui a été réalisé :

– Robert Normandeau : « Oui en fait, c'est un rôle qui est destiné, au fonds, à des interprètes qui sont un

Figure 2. Interaction avec les documents fournis pour les entrevues. Le percussionniste Fabrice Marandola de l'ensemble Sixtrum commente l'évolution du placement des instrumentistes par rapport au système de diffusion.

peu à mi chemin entre des comédiens et des chanteurs. [...] Mais en même temps c'est vrai qu'il y a le travail de la projection de soi dans l'onomatopée. Et si on avait eu plus de temps, je vous aurais probablement indiqué plus de pistes en vous disant : 'voilà ça c'est un chat, ça c'est ceci, ça c'est cela'. Parce qu'il y a beaucoup de métaphores [...] ».

– Marie-Annick Béliveau (alto) : « Mais c'est là que les pistes audio aidaient beaucoup parce qu'il y avait déjà beaucoup de caractère, puis je trouve que dans *le Renard et la Rose* [nda : l'œuvre électroacoustique qui a servi de base à la transcription pour la nouvelle pièce de Robert Normandeau *Baobabs*] encore davantage [...] ».

Elle permet aussi de rendre compte de ce qui n'a pas été fait :

– Robert Normandeau : « Non, c'est vrai que le projet à l'origine c'était vraiment de mettre les consonnes aux percussions et les voyelles aux chanteurs mais ça ne marchait pas comme ça en fait [...] ».

Les documents de travail ont servi de support à l'explicitation du travail créatif des compositeurs dans la lignée du travail de Donin et Theureau, mais avec une visée documentaliste plutôt que musicologique. Ainsi le compositeur Serge Provost (figure 3) passe au travers de tout son patch Open Music pour expliquer sa démarche de transcription de matériel littéraire en partition, au travers notamment du code morse.

– Serge Provost : « Par exemple ici, j'ai fait deux librairies, une pour ce qui est des mots qui viennent de la déclaration universelle des droits de l'homme et l'autre, qui était pour l'autre instrument, qui est le mot liberté dans 25 langues différentes [...] ».

Les données récoltées ne se limitent pas au contexte spécifique de production des œuvres puisque les compositeurs font référence non seulement à leurs pairs mais aussi à leur propre parcours, quelquefois objectivé dans le patch Max-MSP.

Ainsi pour le compositeur Laurie Radford : « Some of these patches have been with me for ten years, or some

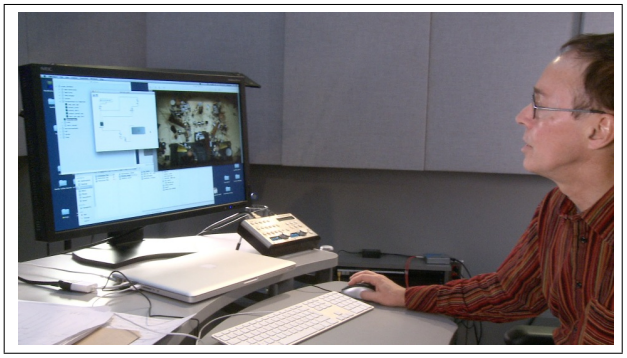


Figure 3. Le compositeur Serge Provost ouvrant son patch Open Music tout en le commentant.

of these sub-patches. And they get adjusted just a little bit in terms of their parameters, in terms of how many signals are coming in, how many signals are going out. For instance, the pitch shifter, I had the same pitch shifter for a number of years and then I started disliking the sound or the quality of the pitch shifter so I rebuilt it with a slightly different configuration. So it's like pitch shifter version two ».

Pour Serge Provost : « Il y a des outils de ce patch qui existaient déjà dans d'autres œuvres et que j'ai récupérés. Par exemple ici, le système de délai, d'harmonisation microtonale [...]. Bon, ce sont des outils qui font partie de ma boîte à outils où je vais reprendre tels ou tels éléments et les rebrancher ensemble afin de faire un réseau qui va correspondre aux besoins de la nouvelle pièce ».

Les compositeurs montrent non seulement le rapport du patch aux œuvres précédentes, à l'œuvre en cours, mais également aux œuvres futures dans les évolutions qu'ils prévoient déjà pour les projets en cours ou à venir.

Laurie Radford : « I think that probably what's going to happen with this [the output module] is that it's going to have another one [routing system] so that the manipulation of multiple signals becomes even more fluid and complex ».

C'est donc tout ce contexte, support de l'intelligibilité des œuvres en vue de sa re-performance et/ou migration, qui ressort du dispositif méthodologique de capture des données en relation avec le cadre conceptuel fournis par Boutard et Guastavino [6], c'est-à-dire en relation à l'organologie, à la transmission des connaissances, aux stratégies de production et bien sur à la composition et l'interprétation.

5. CONCLUSION

Le projet DiP-CoRE a permis de mettre en place une démarche méthodologique de travail pour la documentation et la dissémination des œuvres mixtes. Une fois cette base établie le dispositif peut être sensiblement réduit pour permettre son application à de plus simples dispositifs en fonction des moyens à disposition. Il reste que cette pratique, visant à la pérennisation du répertoire, est située

dans un contexte plus large qui inclut les disciplines des sciences de l'information, de l'archivistique, de l'ingénierie et tout spécifiquement dans leur relation à la conservation et la préservation des objets numériques. Le modèle général de préservation et de dissémination, conceptualisé par Boutard [5] et sur lequel se base cette recherche, requiert de pousser maintenant cette pratique à des dispositifs plus larges : dans la lignée de ce que réclamait déjà Moore [15], cela nécessite de s'appuyer les cadres théoriques et pratiques de préservation des objets numériques que fournissent communément les dépôts numériques, notamment dans le milieu universitaire. Par le modèle des *Mixed Methods Digital Curation* [5], nous avons maintenant les moyens de mettre en relation les pratiques documentaires, principalement qualitatives, établies par le projet DiP-CoRE et la théorie de la préservation des objets numériques, s'appuyant sur des modèles principalement quantitatifs. Pour cela il est nécessaire de mettre en place des plateformes de diffusion—dont les principes méthodologiques ont été mis en lumière par le projet DiP-CoRE—ancrées sur des dépôts numériques. Une volonté politique au niveau institutionnel est cependant nécessaire et implique sans doute, en plus des universités, des acteurs comme ceux identifiés par Berweck [3], notamment les éditeurs, dans un rapport à la fois collaboratif et fondé sur des moyens pragmatiques, qui bénéficierait à toute la communauté des musiques mixtes.

6. REMERCIEMENTS

Le projet DiP-CoRE est financé par le Conseil de recherches en sciences humaines (CRSH) du Canada : programme Connexion attribué aux deux auteurs. Le premier auteur est financé par une bourse postdoctorale du Fonds québécois de la recherche sur la société et la culture (FQRSC). Ce projet a reçu le soutien du Centre Interdisciplinaire de Recherche en Musique, Médias et Technologie.

7. REFERENCES

[1] Baudouin, O., « Les premières musiques synthétisées par ordinateur : préservation et exploitation d'une mémoire », *Actes des Journées d'informatique musicale*, Saint-Etienne, France, 2011.

[2] Bernardini, N. and A. Vidolin, « Sustainable live electro-acoustic music », *Proceedings of Sound and Music Computing 2005*, Salerno, Italy, 2005.

[3] Berweck, S., *It worked yesterday : On (re-)performing electroacoustic music*, Doctoral thesis, University of Huddersfield, 2012.

[4] Born, G., *Rationalizing Culture : IRCAM, Boulez, and the Institutionalization of the Musical Avant-Garde*, University of California Press, Berkeley and Los Angeles, California, 1995.

[5] Boutard, G., « Towards mixed methods digital curation : Facing specific adaptation in the artistic domain », *Archival Science*, in press.

[6] Boutard, G. and C. Guastavino. « Following gesture following : Grounding the documentation of a multi-agent creation process », *Computer Music Journal*, vol. 36 (4), 2012, p. 59–80.

[7] Cance, C., H. Genevois, and D. Dubois, « What is instrumentality in new digital musical devices ? a contribution from cognitive linguistics and psychology », *La Musique et ses instruments*, Delatour, Paris, 2013, p. 283–298.

[8] Clot, Y., *Travail et pouvoir d'agir*, Presses Universitaires de France, Paris, 2008.

[9] Clot, Y. and D. Faïta, « Genres et styles en analyse du travail : Concepts et méthodes », *Travailler*, vol. 4, 2000, p. 7–42.

[10] Donin, N. and J. Theureau, « Theoretical and methodological issues related to long term creative cognition : the case of musical composition », *Cognition, Technology & Work*, vol. 9 (4), 2007, p. 233–251.

[11] Goasdoué, D., « L'art du violon, aperçu historique d'une pratique raisonnée », *Technologies/Idéologies/Pratiques - Revue d'Anthropologie des connaissances*, vol. 14 (2), 2002, p. 69–93.

[12] Leroux, P., « ... phraser le monde : continuité, geste et énergie dans l'oeuvre musicale », *Circuit : musiques contemporaines*, vol. 21 (2), 2011, p. 29–48.

[13] Lorino, P., B. Tricard, and Y. Clot, « Research methods for non-representational approaches to organizational complexity : The dialogical mediated inquiry », *Organization Studies*, vol. 32 (6), 2011, p. 769–801.

[14] Marandola, F. and G. Boutard, « Des puces et des drums, DVD », *empreintes DIGITALes*, à venir.

[15] Moore, A., « History and archival : the pitfalls of storage », *Colloque Informatique musicale : utopies et réalités*, November 2009.

[16] Orcalli, A., « Réflexions sur la restitution des œuvres musicales électroniques », *Actes des Journées d'informatique musicale*, Saint-Etienne, France, 2011.

[17] Polfreman, R., D. Sheppard, and I. Dearden, « Time to re-wire ? problems and strategies for the maintenance of live electronics », *Organised Sound*, vol. 11 (3), 2006, p. 229–242.

[18] Theureau, J., « 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*, 4(2), 2010, p. 287–322.

[19] Tiffon, V., « Les musiques mixtes : entre pérennité et obsolescence », *Musurgia*, vol. 12 (3), 2005, p. 23–45.

[20] Vermersch, P., *L'entretien d'explicitation*. ESF, Issy-les-Moulineaux, 7 edition, 2011.

Journée Sonification
vendredi, 23 mai 2014

SONIFICATION AND ART

DOMINANTE DE LA JOURNEE (KEYNOTE)

Peter Sinclair
Locus Sonus ESAA
petesinc@nujus.net

After a brief definition of sonification I will discuss two creative approaches – that of Sound design and that of conceptual art. I will attempt to see how these differ and converge in their goals and methods particularly in the context of what we might call musical sonification. I will describe some examples of key sonification artworks and the aesthetic strategies they employ. Finally I will describe RoadMusic a project that I have been working on for some time, and the practice to which I apply my ideas concerning sonification.

After a brief definition of sonification I will discuss two creative approaches – that of Sound design and that of conceptual art. I will attempt to see how these differ and converge in their goals and methods particularly in the context of what we might call musical sonification. I will describe some examples of key sonification artworks and the aesthetic strategies they employ. Finally I will describe RoadMusic a project that I have been working on for some time, and the practice to which I apply my ideas concerning sonification.

It has been suggested in the past that sonification as a term should exclude artistic and musical usage of sound. I am referring here to an article by Sonification expert: Thomas Hermann that can be found in the 2008 ICAD proceedings. In a special edition of AI&Society dedicated to artistic sonification that I guest edited, several artists and composers contradicted this position and Hermann himself has revised his point of view since contribution to the review which he co-signs starts with: ‘Sonification today is an interdisciplinary practice ranging from scientific applications to sound art and composition.’

ALGORITHMIC ORCHESTRATION WITH CONTIMBRE

Thomas A. Hummel
conTimbre
Beethovenstr.30
D79100 Freiburg
th@contimbre.com

1. INTRODUCTION

conTimbre [1] is a database project for the sound of the contemporary orchestra. It has been published in 2012 and comprises recordings and documentation of more than 86.000 sounds for 158 orchestra instruments and more than 4.000 playing techniques. Within this, mor than 20.000 woodwind multiphonics are recorded and documented. The aim is manifold.

It offers a web browser based software called „conTimbre learn“, which allows the student of composition or instrumental musicians to learn about the playing possibilities of the contemporary orchestra. It includes a search function to search sounds by a series of paramteres.

The "conTimbre Orchestrator" is a MAX [2] based software which allows to simulate orchestral chord situations. The user can edit a virtual orchestra, select instruments, playing techniques and pitches (including microtones) or versions for a certain playing technique. Notation graphics including fingering schemes or special notations are presented. The user can ask for detailed text and photo documentation.

The conTimbre orchestrator can load orchestrations from disk which may be calculated through a CommonLisp interface.

The „conTimbre ePlayer“ is a MAX based sampler with which it is possible to play contemporary scores from commercial score editors by MIDI. It is also possible to play on the ePlayer by MAX messages or through OSC from controllers like Supercollider.

2. ALGORITHMIC ORCHESTRATION

The conTimbre database includes a CommonLisp library which creates orchestrations. The orchestrations can be loaded into the „conTimbre Orchestrator“.

The library runs on Mac and needs the installation of the freeware Steel Bank Common Lisp [3]. On startup, the whole conTimbre database metadata are loaded into RAM. These metadata include

- file names,
- instruments,
- playing techniques,
- comments,
- pitch information,
- spectral envelopes,
- attack times,
- partial series,
- and others.

3. ORCHESTRATION RULES

An important application for this library is the simulation of real orchestral situations. In real orchestral situation, orchestration rules have to be considered. Some orchestra instruments can play several notes at a time, some others not. Real musicians can change their playing technique within a performance only within limits.

For this reason, an orchestration rules module was developed. It divides orchestration rules into two different types.

3.1. Vertical rules

Vertical rules deal on restrictions or possibilities of real instruments within a static chord.

Empirically, the behaviour of instruments was analysed, and groups of instruments were built which obey the same rules.

3.1.1. The one-sound rule

This is the most simple rule and means, that an instrument can just play one sound at a time of this kind. This rule applies generally to woodwind and brass instruments.

3.1.2. The glockenspiel rule

This rule applies typically to the glockenspiel, but also to other instruments like piano and marimbaphone. It means, that the instrument can play a certain number of sounds at same time, but they should be of the same playing mode. The number of sounds to be played in parallel depends on the instrument, and also the pitch ambitus of the chord to be played. E.g., a marimbaphone can play up to 4 pitches at same time using a set of 4 beaters in two hands.

3.1.3. The strings rule

A special case is the rule for string instruments. String instruments playing ordinario or techniques similar to ordinario can realise two sounds on adjacent strings (double stops). When playing double stops, there is a restriction on the pitch choice. Stops positions should not be too different as the hand span may not be sufficient.

3.1.4. Detailed rules

For some instruments, more detailed rules are required. The glockenspiel rule needs to be precise on the pitch ambitus, as for example the piano chord cannot exceed a certain ambitus because of the size of the hand. In the case of the piano, this rule only applies to ordinario too. In the case of special playing techniques like guiro effects, only one guiro sound can be realised on the piano at same time.

3.2. Horizontal rules

Horizontal rules deal on restrictions within a temporal sequence. Generally, there is only one kind of this rule, ie. the mutual exclusion of two different playing technique without any time interval.

This kind of rule has to be examined for each instrument family individually. In the following, some examples are given.

3.2.1. brass instruments.

Different kinds of mutes in a sequence need a certain time interval to change the mute.

3.2.2. Percussion instrumentas

Different beaters cannot be changed without a pause. An exception is that different beaters can be hold in the hands.

3.2.3. Piano

playing modes with preparations on strings may not be mixed with playing modes without any preparation - at least for the same string.

3.2.4. All instruments

Playing techniques with a dismembered instrument cannot not be mixed with playing techniques using the normal instrument.

4. REFERENCES

[1] conTimbre database project. www.contimbre.com

[2] MAX/msp. www.cycling74.com

[3] Steel Bank CommonLisp. www.sbcl.org

CAGE: UNE LIBRAIRIE DE HAUT NIVEAU DÉDIÉE À LA COMPOSITION ASSISTÉE PAR ORDINATEUR DANS MAX

Andrea Agostini
HES-SO, Genève
and.agos@gmail.com

Éric Daubresse
HES-SO, Genève
eric.daubresse@hesge.ch

Daniele Ghisi
HES-SO, Genève
danieleghisi@gmail.com

RÉSUMÉ

Cet article est une introduction à *cage*, une librairie pour l'environnement Max [1] composée d'un certain nombre de modules de haut niveau pouvant être utilisés principalement pour la composition assistée par ordinateur. La librairie, actuellement en version alpha, contient un ensemble d'outils dédiés à plusieurs catégories de problèmes typiquement abordés par cette discipline : génération de notes, génération et traitement de profils mélodiques, processus symboliques inspirés par le traitement du signal audio, interpolations harmoniques et rythmiques, automates et L-systèmes, rendu audio, outils pour la *set theory*, outils pour la gestion de partitions. Ce projet, soutenu par la Haute école de musique de Genève [2], a principalement une vocation pédagogique : en effet, tous les modules de la librairie sont des abstractions, qui se prêtent très facilement à être analysées et modifiées.

1. INTRODUCTION

Dans cet article seront abordés certains des principaux concepts de la librairie *cage* [3] pour Max ; elle contient plusieurs modules de haut niveau pour la composition assistée par ordinateur (CAO). Elle est entièrement basée sur la librairie *bach : automated composer's helper*, développée par deux des auteurs [1, 3]. Comme pour *bach*, *cage* est principalement centrée autour de la notation symbolique dans le domaine du temps réel. Les objets de *cage* communiquent entre eux avec le mécanisme des *Lisp-like linked lists* (*llll*s) [2].

À la différence de *bach*, qui se compose d'un grand nombre d'objets et d'abstractions prenant en charge des opérations de bas niveau sur ces listes (p.e. : rotations, inversions, entrelacements...), ou bien des opérations très avancées mais cependant essentiellement basiques (p.e. : résolution de problèmes par contraintes, quantification rythmique...), les modules de *cage* accomplissent en général des taches de plus haut niveau, ayant une connotation plutôt compositionnelle que strictement technique (p.e. : génération de matériel mélodique ou calcul de modulations de fréquences symboliques).

Deux critères principaux ont motivé la conception et réalisation de la librairie.

Le premier est celui qui a été à la base de la création de *cage* : construire une librairie de modules prêts à l'utilisation pour créer des classes de processus assez universels dans la pratique de CAO. Une partie de cette librairie est donc directement inspirée par d'autres librairies déjà existantes dans quelques logiciels (notamment les librairies *Profile* [9] et *Esquisse* [6, 8] pour *Patchwork*, qui ont ensuite été portées dans *OpenMusic* [4]) ; parallèlement, un autre versant de la librairie trouve sa raison d'être dans des concepts issus du monde du temps réel (p.e. : *cage.granulate*, le moteur de granulation symbolique).

Le second critère est lié à la forte connotation pédagogique du projet : à la difference de *bach*, dont les fonctionnalités principales sont implémentées dans des objets compilés, tous les modules de la *cage* sont des abstractions, qui se prêtent très facilement à être analysées et modifiées. Il n'est donc pas compliqué, pour l'utilisateur qui souhaite apprendre à manipuler des données musicales, de copier, modifier ou ajuster des morceaux de patch pour ses propres besoins. Cette flexibilité des abstractions fait en sorte que, bien que les processus implémentés soient conçus pour fonctionner facilement avec une connaissance moyenne de Max, l'utilisateur plus avancé pourra non seulement partir de ces abstractions et en modifier le comportement selon son projet, mais également les intégrer dans son propre environnement de travail, que ce soit au studio ou dans le cadre d'une performance. Cette vocation pédagogique est complétée par le fait que la librairie sera entièrement documentée, avec des fichiers d'aide, des feuilles de référence et une collection de tutoriaux.

2. UNE APPROCHE TEMPS RÉEL À LA COMPOSITION ASSISTÉE

Le paradigme du temps-réel influence profondément la nature elle-même du processus compositionnel. Par exemple, les compositeurs qui opèrent dans le domaine de la musique électroacoustique ont souvent besoin que la machine réagisse immédiatement à tout changement de paramètres. De la même manière, les compositeurs qui composent avec des donnés symboliques pourraient souhaiter que la machine s'adapte dans les plus brefs délais

1 . *http ://cycling74.com*

2 . La librairie *cage* est développée au sein de la HEM avec le soutien du fonds stratégique de la HES-SO (Projet CPE-MUS12-12 : Développement d'outils destinés à l'enseignement, la composition et l'interprétation)

3 . *www.bachproject.net/cage*

à la nouvelle configuration. Le paradigme sous-jacent à *cage* est donc le même que celui qui a façonné la librairie *bach* : la création et la modification des données musicales n'est pas forcement réduite à une activité hors du temps, mais elle suit, en s'adaptant, le flux temporel du processus compositionnel (voir aussi [3, 5, 11]).

3. COMPOSITION DE LA LIBRAIRIE

La librairie se compose de plusieurs familles de modules qui sont regroupés en catégories.

3.1. Génération de hauteurs

Une première famille de modules s'occupe de la génération des hauteurs selon des critères différents : *cage.scale* et *cage.arpeggio* peuvent générer, respectivement, des échelles et des arpèges, dans un certain ambitus de hauteurs. La typologie des accords ou des échelles peut être donné soit par nom de type symbolique (p.e. **F#m**, **ReM**), soit avec un pattern de valeurs d'intervalles exprimées en midicents (p.e. **100 200 100 200 100 200 200 100**) [4]. Les dénominations des échelles et des accords peuvent aussi bien contenir des quarts et des huitièmes de tons (p.e. **Sol+M**, **Abv7**... [5]). *cage.harmser* génère des séries harmoniques à partir d'une note fondamentale, avec éventuellement un facteur de distortion harmonique.

D'autres modules opèrent en générant des hauteurs une par une : *cage.noterandom* génère des notes au hasard, à partir d'un réservoir donné, selon plusieurs poids de probabilités prédéfinis qui peuvent être soit ignorés, soit définis simplement grâce à *cage.weightbuilder* (voir Fig. 1) ; *cage.notewalk* génère un chemin aléatoire dans un réservoir donné, selon une liste de pas admissibles. Dans les deux cas, le résultat de l'opération est conçu pour être utilisé en combinaison avec *bach.transcribe*, qui va gérer la transcription symbolique en temps réel. Ainsi, dans les deux cas, l'élément choisi au hasard peut être validé *a posteriori* par l'utilisateur via un *lambda loop*. [6]

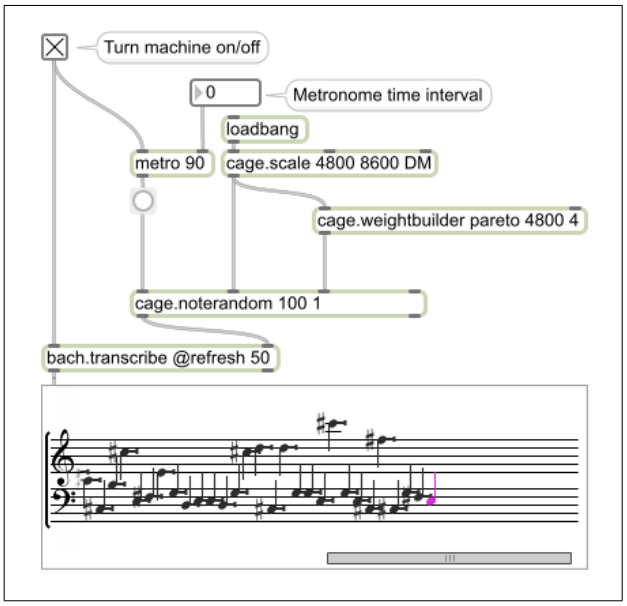


Figure 1. Génération en temps réel de notes tirées d'une échelle de ré majeur, avec des poids de probabilité donnés par une distribution de type 'pareto' avec 4800 midicents comme *pareto wall* et 4 comme valeur d'exposant.

3.2. Génération et traitement de profils mélodiques

Une famille de modules est spécifiquement dédiée à la génération et au traitement des profils mélodiques, d'une manière similaire à la librairie *Profiles* dans Patchwork et OpenMusic [9]. Une *breakpoint function*, par exemple dessinée dans un objet *function* ou un objet *bach.slot* [7] peut être convertie en une séquence de hauteurs (un profil mélodique) grâce à *cage.profile.gen*. Ce profil peut être modifié de plusieurs manières : compressé ou étiré (avec *cage.profile.stretch*), renversé (avec *cage.profile.mirror*), approximé par une grille harmonique ou par une échelle (avec *cage.profile.snap*), forcé dans une région de hauteurs (avec *cage.profile.rectify*), perturbé d'une façon aléatoire (avec *cage.profile.perturb*), ou filtré (avec *cage.profile.filter*). Dans ce dernier cas, le filtrage du profil est réalisé par l'application d'un filtre moyen, médian ou encore 'custom', défini par l'utilisateur via un *lambda loop* (voir aussi Fig. 2).

3.3. Processus d'inspiration électroacoustique

La librairie *cage* contient un groupe de modules dédiés à l'émulation symbolique de processus extraits du domaine de la synthèse sonore et du traitement du signal audio.

cage.freqshift est un outil qui permet de transposer des matériaux de façon linéaire sur l'axe des fréquences, à la

4 . *cage*, comme *bach*, adopte la convention de *Patchwork* et *OpenMusic* [4] d'exprimer les hauteurs et les intervaux en midicents, c'est-à-dire des centièmes de note MIDI.

5 . Par convention, **+** indique une altération de $+1/4$ de ton ; **-** ou **d** indique une altération de $-1/4$ de ton ; ˆ et **v** font la même chose pour les huitièmes de ton.

6 . Un *lambda loop* dans *bach*, et donc par extension dans *cage*, est une configuration de feedback symbolique : les objets qui le supportent ont une ou plusieurs sorties dédiées (*lambda outlets*) qui envoient des donnés devant être validées ou modifiées ; ces données sont élaborées dans une section du patch et la 'réponse' est ensuite re-injectée dans une entrée dédiée (*lambda inlet*) de l'objet de départ. Cette configuration est très souvent utilisée dans *bach* pour définir des comportements personnalisés pour certains éléments (p.e. : un critère d'ordonnancement d'une liste, ou un processus qui doit être appliqué à chaque élément d'une *llll*, et ainsi de suite). Le nom 'lambda' est une allusion au fait que cette configuration permet, en quelque sorte, de passer un morceau de patch comme pseudo-argument d'un objet. Ce n'est pas plus qu'une allusion : il ne s'agit en aucun cas de lambda-calcul, ni de fonctions interprétées.

7 . Dans *bach*, un *slot* est en général un conteneur de méta-données associées à une certaine note [1]. L'information contenue dans un *slot* peut avoir des formes différentes, par exemple : *breakpoint functions*, nombres et listes de nombres, filtres, texte, liste de fichiers à parcourir, matrices, trajectoires de spatialisation... *bach.slot* permet l'affichage et l'édition d'un *slot* sans qu'il ne soit associé à aucune note spécifique.

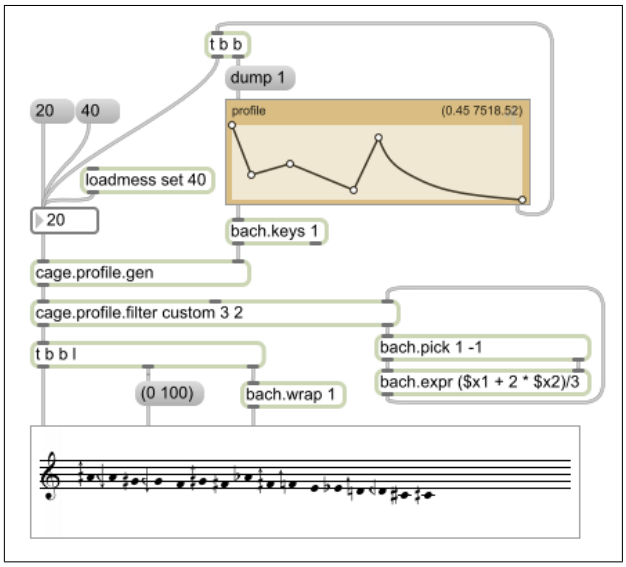


Figure 2. Un profil mélodique est construit à partir d'une fonction définie dans un *bach.slot*. Dans ce cas, la fonction affichée est échantillonnée en 20 points. Ensuite, ce profil est filtré par un processus exprimé à travers un *lambda loop*, qui fonctionne avec des fenêtres de trois notes ; pour chaque fenêtre, on substitue une moyenne du premier et dernier élément de la fenêtre, pondérée avec les poids $(1, 2)$. Ce processus de filtrage est répété deux fois. On remarque que, à cause du fenêtrage, le résultat contient quatre notes de moins que l'échantillonnage original.

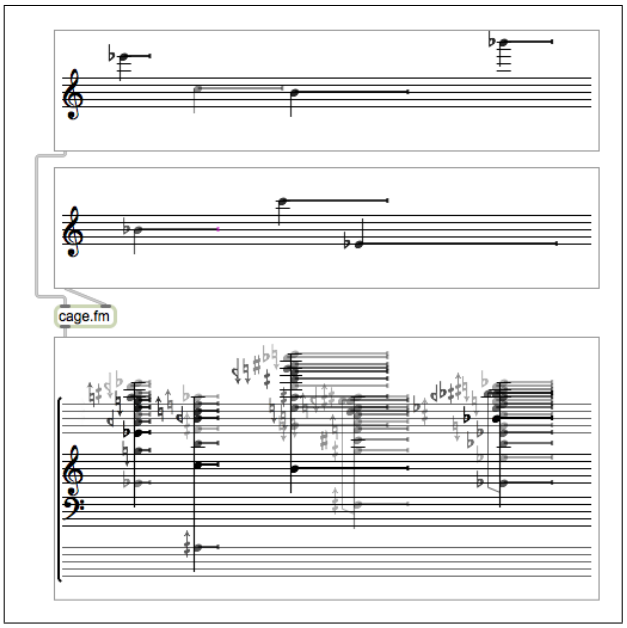


Figure 3. Un exemple de modulation en fréquence entre deux partitions, obtenue à travers l'abstraction *cage.fm*. La 'porteuse' et la 'modulante' sont en haut, le résultat en bas. La vélocité des notes (traitée dans ce cas comme l'amplitude des sinusoïdes correspondantes) est représentée par une échelle de gris.

manière du traitement audio du *frequency shifting*. En raison de la similarité des deux processus, on classifiera aussi *cage.pitchshift* dans cette même catégorie, quoique l'opération de *pitch shifting* sur la notation musicale soit, en fait, une simple transposition.

cage.rm et *cage.fm* sont dédiés respectivement à la modulation en anneaux et en fréquence. L'idée à la base de ces techniques, largement utilisées par les compositeurs associés a l'école spectrale, est la suivante : à partir de deux accords (l'un 'portant' et l'autre 'modulant') dont chaque note est considérée comme une sinusoïde simple, on calcule le spectre qu'on obtiendrait en modulant entre eux ces deux groupes de sinusoïdes. Cette opération nécessite d'effectuer des approximations et des compromis qui peuvent éloigner son résultat de celui obtenu par le même processus appliqué à des signaux audio : cependant, il s'agit d'une approche très efficace pour la génération de familles harmoniques potentiellement très riches, ceci à partir de matériaux simples, d'où leur intérêt compositionnel. Quoique l'inspiration immédiate de *cage.rm* et *cage.fm* provienne de la librairie *Esquisse* [6, 8] pour *OpenMusic*, leur paradigme de fonctionnement et certains détails relatifs aux calculs sont différents. En particulier, ces deux modules sont conçus pour travailler dans le temps, et peuvent ainsi accepter comme données d'entrée des accords simples, mais aussi des suites d'accords, afin de représenter dans le temps les variations des 'porteuses' et des 'modulantes'. Dans ce cas le processus restituera une nouvelle partition, qui tiendra compte de ces variations dans le temps (voir Fig. 3). En ce qui concerne le véritable calcul interne, les deux modules prennent en compte une estimation des oppositions de phase générées par la modulation, et donc permettent l'élimination de certaines fréquences, à la différence de la librairie *Esquisse*. Pour cette raison, les résultats d'un même processus dans les deux environnements peuvent être très différents.

cage.virtfun est un estimateur de la fréquence fondamentale virtuelle d'un accord, comme on la perçoit par exemple à l'issue d'un processus de *waveshaping*. L'implémentation est très simple : on parcourt la série subharmonique de la note la plus grave de l'accord, jusqu'à trouver une fréquence dont les harmoniques approximent toutes les notes du même accord avec un degré de tolérance établi. Il est aussi possible d'utiliser *cage.virtfun* à partir d'une séquence d'accords dans le temps : le résultat sera alors une séquence correspondant à la suite des fondamentales virtuelles de chaque accord.

cage.delay étend le principe de la ligne à retard avec réinjection dans le domaine symbolique. Il s'agit essentiellement d'un outil pour la création de boucles et de structures répétitives, qui permet d'altérer le matériau à chaque pas du processus à travers un *lambda loop*. Le temps de retard lui-même peut être changé d'une répétition à l'autre. Il n'y a pas de limitation à priori dans la richesse des processus auxquels les matériaux sont soumis dans le *lambda loop* : le résultat musical peut donc être beaucoup plus complexe qu'une simple itération.

cage.cascade$\sim$ et *cage.pitchfilter* étendent le principe

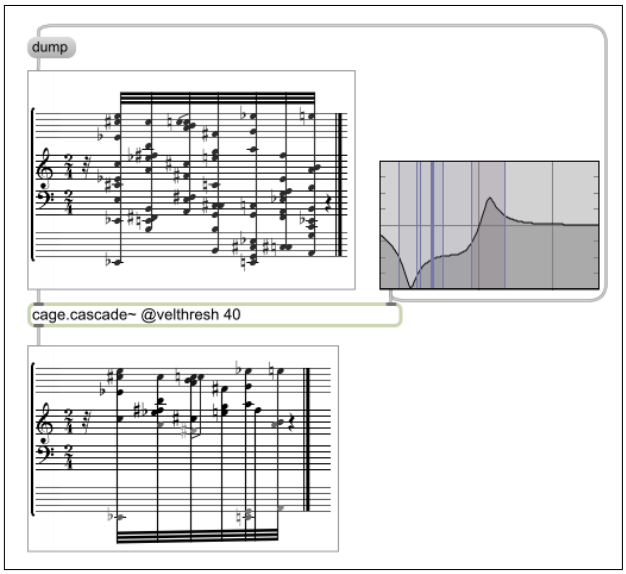


Figure 4. Un exemple de filtrage d'une partition avec l'objet *filtergraph∼* et *cage.cascade∼*. À chaque fois que le filtre est modifié par l'interface, le résultat est automatiquement mis à jour en temps réel.

du filtrage dans le domaine symbolique. Le premier applique à une partition une séquence de filtres avec deux pôles et deux zéros, de façon similaire aux objets Max *biquad∼* et *cascade∼* (voir fig. 4), en émulant la réponse en fréquence d'un véritable filtre numérique [8] . Le second opère par contre sur le domaine des hauteurs (et non pas des fréquences), en appliquant à une partition un filtre défini tout simplement par une *breakpoint function*, produite par exemple par un objet *function* ou *bach.slot*. Dans les deux cas, la vélocité MIDI de chaque note est modifiée en accord avec la réponse du filtre, et les notes dont la vélocité tombe au dessous d'un certain seuil sont éliminées. Il est aussi possible de définir des changements dans l'interpolation des filtres dans le temps (voir fig. 5).

cage.granulate est un moteur de granulation symbolique. Les paramètres de granulation sont les mêmes que dans les processus correspondant en musique électroacoustique, c'est à dire : l'intervalle de temps entre deux grains, la taille de chaque grain, le début et la fin de la zone de la partition d'où le grain doit être extrait. À partir de ces données, *cage.granulate* se charge des calculs et remplit en temps réel un *bach.roll* relié à sa sortie (voir fig. 6).

3.4. Interpolations harmoniques et rythmiques, formalisation de l'agogique

Le module *cage.chordinterp* opère un calcul d'interpolation harmonique linéaire dans un ensemble d'accords, en fonction de poids attribués pour chacun d'eux par l'utilisateur. De la même manière, on peut obtenir une interpolation rythmique entre un ensemble de figures à travers le module *cage.rhythminterp*.

Un cas particulier d'interpolation rythmique, ou plutôt une extension du même principe, est la formalisation de l'agogique : c'est-a-dire, l'écriture dans un système de notation proportionnelle (qui bien sûr pourra ensuite être quantifié) d'une structure musicale répétée en accelerando ou rallentando. Cette structure musicale peut aussi être une simple impulsion rythmique constante qui représente une grille pour des figures musicales non répétitives : pour cette raison, nous avons choisi de réduire le problème au cas d'une structure itérative. Nous avons donc identifié cinq paramètres qui caractérisent un accelerando ou un rallentando : durée de la première instance, durée de la dernière instance, relation entre deux instances consécutives, nombre d'instances, durée totale de la figuration. Evidemment ces paramètres ne sont pas indépendants : en général, à partir d'un ensemble de paramètres de départ, on peut calculer les autres et formaliser entièrement l'accelerando ou le rallentando. La librairie *cage* comprend donc un groupe de modules dédiés au calcul des paramètres manquants à partir des données disponibles (un module est utilisé pour chaque combinaison de données significatives en entrée), et un module 'central' calcule au final l'accelerando ou le rallentando.

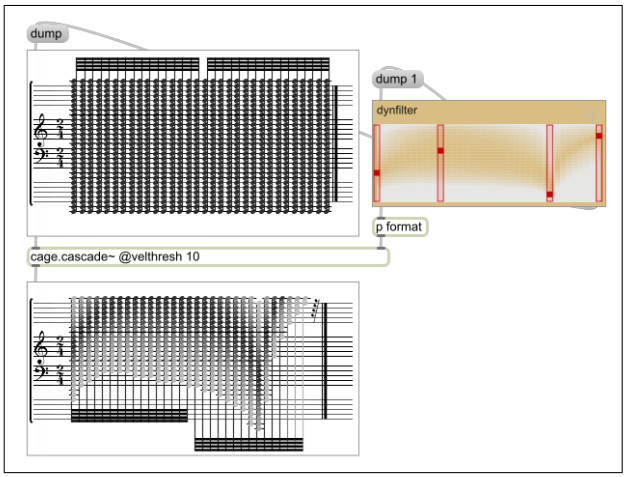


Figure 5. Un exemple de filtrage dynamique d'une partition obtenu avec *cage.cascade∼* piloté par un slot de type *dynfilter* dans l'objet *bach.slot*. À chaque fois que le filtre est modifié par l'interface, le résultat est automatiquement mis à jour en temps réel. Les quatre rectangles de couleur rouge dans le *bach.slot* à droite représentent quatre filtres passe-bande, dont les fréquences centrales sont représentées par les carrés rouges ; à différence de la Fig. 4, ici le résultat est donné par interpolation de ces filtres, et non pas par un unique filtre statique.

3.5. Automates et L-systèmes

Deux modules se consacrent aux systèmes génératifs et d'automates.

8 . Puisque pour calculer la réponse en fréquence d'un filtre avec deux pôles et deux zéros il est nécessaire de connaître la fréquence d'échantillonnage, l'abstraction *cage.cascade∼* fait partie des abstractions de DSP (*digital signal processing*), comme le ∼ indique, bien qu'elle agisse sur un contenu symbolique.

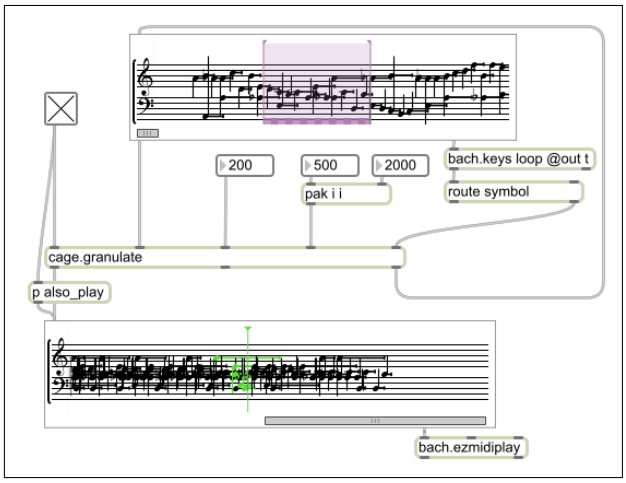


Figure 6. Un exemple de granulation d'une partition en temps réel ; les grains ont une distance de 200ms, et une taille comprise entre 500ms et 2000ms. La région de loop dans le *bach.roll* supérieur est la région d'où les grains sont extraits.

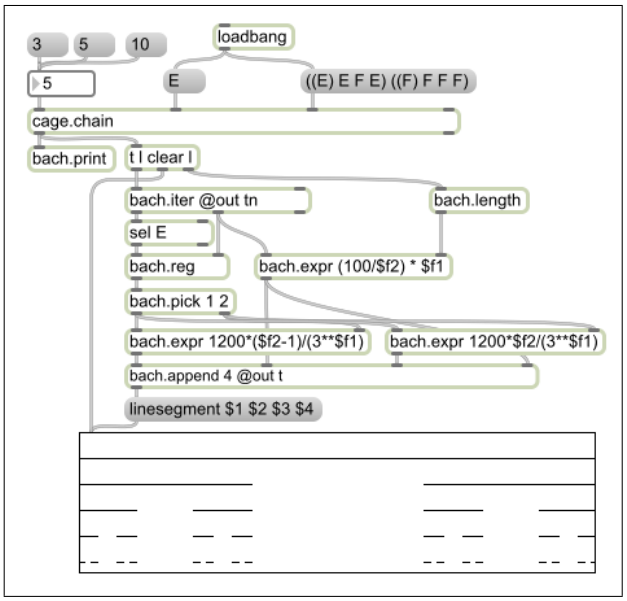


Figure 7. Un exemple de génération et d'affichage d'un comportement fractal, obtenu par une règle de substitution. La liste **E** est la liste initiale ; les règles de substitution sont : à **E** on substitue la séquence **E F E**, à **F** on substitue la séquence **F F F**. La première itération donne **E F E**, la deuxième donne **E F E F F F E F E**, et ainsi de suite. Dans l'affichage, on interprète **E** comme une ligne et **F** comme un espace vide. Ce patch génère cinq itérations qui approximent l'ensemble de Cantor. Toute la partie du patch située en dessous de *cage.chain* est dédiée uniquement à l'affichage.

cage.chain modélise des automates cellulaires en dimension 1, et de L-systèmes. Il opère des ré-écritures d'une liste donnée, selon un certain nombre de règles définies par l'utilisateur soit par message, soit via un *lambda loop*. Les substitutions peuvent intervenir sur des éléments simples (p.e. une certaine lettre ou note est substituée avec une liste de lettres ou notes), ou bien sur des suites d'éléments ayant une longueur fixée (p.e. à chaque couple d'éléments, qu'ils se présentent séparés ou superposés, on substitue un ou plusieurs autres éléments) ; dans ce dernier cas, *cage.chain* gérera le comportement aux limites de la grille en fonction des valeurs de certains attributs (*pad*, *align*). Avec ce module il est ainsi très simple de construire des automates cellulaires, ou des fractales obtenus par substitution (voir Fig. 7).

cage.life s'occupe d'automates cellulaires en dimension 2 (dont le plus fameux exemple est sans doute le 'jeu de la vie' de Conway). Les règles de ces automates sont définies grâce à un *lambda loop*. L'ordre des sous-matrices de substitution peut également être défini par l'utilisateur.

3.6. Rendu audio

Deux modules dans *cage* sont consacrés à la production d'un rendu audio de partitions *bach* : *cage.ezaddsynth*∼ (un moteur de synthèse par ondes sinusoïdales) et *cage.ezseq*∼ (un échantilloneur de fichiers sons). Les deux sont prêts à être utilisés très simplement, à la façon de *bach.ezmidiplay*, en les connectant simplement à la sortie dédiée 'playout' des objets de notation.

Le moteur de synthèse répond à la nécessité d'avoir un rendu audio qui ne soit pas basé sur une sortie MIDI. Cela est par exemple extrêmement utile quand on travaille avec des grilles microtonales non standard, ou quand on a besoin d'enveloppes liées aux notes. *cage.ezaddsynth*∼ peut prendre en compte une enveloppe d'amplitude et une enveloppe de panning (données comme slots). Il suffit ensuite de relier *cage.ezaddsynth*∼ à la sortie 'playout' des objets de notation pour avoir le rendu audio, qui prend en compte les slots pré-cités.

L'échantillonneur répond au besoin d'utiliser les objets *bach.roll* et *bach.score* comme des 'sequencers augmentés' : *cage.ezseq*∼ fournit une palette standard de slots, censés contenir pour chaque note l'information du nom du fichier, l'enveloppe d'amplitude, la vitesse de lecture, l'enveloppe de panning, le filtrage audio, et le point de départ dans la lecture du fichier (en millisecondes). Si ces informations ne sont pas toutes données, des valeurs par défaut sont utilisées. L'intérêt d'utiliser *cage.ezseq*∼ est aussi lié au fait que cet objet prend en charge la mise en mémoire ('preload') des fichiers audio, une fois qu'on lui donne un lien vers un dossier qui les contient. Ce module peut aussi s'occuper d'opérer sur chaque échantillon une transposition automatique sans altération temporelle, à l'aide de l'objet Max *gizmo*∼.

3.7. Outils de *set theory*

Certains modules dans *cage* sont dédiés à des représentations propres de la *set theory* : *cage.chroma2pcset* et *cage.pcset2chroma* opèrent des conversions entre

pitch class sets et vecteurs de chroma (voir [10]) ; *cage.chroma2centroid* et *cage.centroid2chroma* opèrent des conversions entre vecteurs de chroma et centroïdes spectraux. Le centroïde spectral d'un vecteur de chroma est obtenu via la transformée introduite par Harte et Sandler [7]. Dans le passage de chroma au centroïde, une partie de l'information originale est forcément perdue ; par conséquent, la conversion de *cage.centroid2chroma* n'est pas univoque : un seul vecteur de chroma, parmi tous ceux qui ont pour centroïde le vecteur introduit en input, est restitué.

3.8. Partitions

cage contient un groupe de modules pour le traitement global de partitions entières. Dans ce groupe, les outils d'usage le plus courants sont *cage.rollinterp*, qui opère une interpolation entre les partitions contenues dans deux objets *bach.roll*, en utilisant comme paramètre la courbe d'interpolation (ou bien une valeur fixe, dans le cas d'une interpolation statique), et *cage.envelopes*, qui représente une famille de fonctions synchronisées sur la durée totale d'une partition, et qui aident à modifier en temps réel la partition en rapport avec les valeurs des courbes à chaque instant.

4. ROADMAP

Au moment de l'écriture de ce texte, la librairie est encore dans sa phase de développement. Une version alpha publique sera disponible en mai 2014 : il est possible que toutes les fonctionnalités prévues ne soient pas encore implémentées ; la documentation ne sera pas complète. Cependant, la majorité des modules seront déjà utilisables. La première version complète de la librairie sera mise en ligne au mois de octobre 2014, à l'occasion d'une présentation officielle qui se déroulera à Genève. La librairie sera disponible sous le modèle "open source"et son téléchargement libre. À partir de l'année académique 2014-2015, *cage* sera objet d'enseignement dans les cours de composition et de musique électronique à l'Haute École de Musique de Genève et dans un ensemble d'institutions partenaires.

5. REMERCIEMENTS

cage est un projet de recherche mené au sein du centre de musique électroacoustique de la Haute Ecole de Musique de Genève, avec le soutien du domaine musique et arts de la scène de la Haute Ecole Spécialisée de Suisse occidentale. Le nom *cage*, qui dans le parallèle entre lettres et notes représente la fameuse expansion de *bach*, est aussi un acronyme en hommage à ce soutien : *composition assistée Genève*.

6. REFERENCES

[1] A. Agostini and D. Ghisi, "*bach* : an environment for computer-aided composition in Max," in *Proceedings of the International Computer Music Conference (ICMC 2012)*, Ljubljana, Slovenia, 2012, pp. 373–378.

[2] ——, "Gestures, events and symbols in the *bach* environment," in *Proceedings of the Journées d'Informatique Musicale*, Mons, Belgium, 2012, pp. 247–255.

[3] ——, "Real-time computer-aided composition with *bach*," *Contemporary Music Review*, no. 32 (1), pp. 41–48, 2013.

[4] G. Assayag and al., "Computer assisted composition at Ircam : From patchwork to OpenMusic," *Computer Music Journal*, no. 23 (3), pp. 59–72, 1999.

[5] A. Cont, "Modeling Musical Anticipation," Ph.D. dissertation, University of Paris 6 and University of California in San Diego, 2008.

[6] J. Fineberg, "Esquisse - library-reference manual (code de Tristan Murail, J. Duthen and C. Rueda)," Ircam, Paris, 1993.

[7] C. Harte, M. S, and M. Gasser, "Detecting harmonic change in musical audio," in *In Proceedings of Audio and Music Computing for Multimedia Workshop*, 2006.

[8] R. Hirs and B. G. editors, *Contemporary compositional techniques and OpenMusic*. Delatour/Ircam, 2009.

[9] M. Malt and J. B. Schilingi, "Profile - libreria per il controllo del profilo melodico per Patchwork," in *Proceedings of the XI Colloquio di Informatica Musicale (CIM)*, Bologna, Italia, 1995, pp. 237–238.

[10] M. Müller, *Information Retrieval for Music and Motion*. Springer Verlag, 2007.

[11] M. Puckette, "A divide between 'compositional' and 'performative' aspects of Pd," in *Proceedings of the First Internation Pd Convention*, Graz, Austria, 2004.

SYSTEMIC THINKING AND CIRCULAR FEEDBACK CHAINS IN A LIVE-ELETRONICS ALGORITHM FOR *GOSTO DE TERRA*

Daniel Puig

Universität der Künste (UdK-Berlin)
Universidade Federal do Rio de Janeiro (UFRJ)
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
danielpuig@me.com

RÉSUMÉ

La formalisation de certains présupposés de la pensée systémique dans un algorithme live-électronique à l'intérieur d'un patch de Max/MSP 6, fait partie de ma pièce *gosto de terra*, pour piano et live-électronique. Cela montre une possibilité de traiter des concepts comme *différence*, *information*, *numéro*, *quantité* et peut-être *pattern*, à la programmation dans un contexte musical qui intègre l'interprète, l'instrument, l'espace acoustique et l'algorithme dans sa conceptualisation. A partir des idées de Gregory Bateson sur ces concepts, le patch tente de recueillir des informations sur la performance, par des différences de second ordre, en comparant les différences entre les quantités dans le temps. Ceci est fait en comparant leur flux selon un seuil, qui établit un contrôle de "sensibilité". Le résultat de l'algorithme peut être utilisé pour déclencher d'autres algorithmes dans le patch, ce qui entraîne un fonctionnement du live-électronique suivant les caractéristiques prévues de la performance live.

ABSTRACT

The formalization of some presuppositions stemming from systemic thinking in a live-electronics algorithm within a Max/MSP 6 patch, is part of my piece *gosto de terra* for piano and live-electronics. It shows one possibility of dealing with concepts as *difference*, *information*, *number*, *quantity* and possibly *pattern* in programming within a musical context that integrates performer, instrument, acoustic space and the algorithm in its conceptualization. Following Gregory Bateson's ideas on these concepts, the patch tries to gather information about the performance, through second order differences, by comparing differences between quantities in time. This is done comparing their flow against a threshold, that sets a "sensibility" control. The result of the algorithm may be used to trigger other algorithms in the patch, resulting in a functioning of the live-electronics that follows the intended characteristics of the live performance.

1. INTRODUCTION

This text focuses on the formalization of some presuppositions in systemic thinking, in live-electronics algorithms within a Max/MSP 6 patch. They are part of my piece *gosto de terra*, for piano and live-electronics. The composition was finished in august 2013 and revised after the first performance, together with the pianist Adam Marks, to whom it was written for. It deals with different concepts included in the discussions presented in my doctoral dissertation.[1]

2. PRESUPPOSITIONS

The dissertation deals, among other discussions, with concepts stemming from Gregory Bateson's systemic thinking. In his book *Mind And Nature: a necessary unity* [1], he sets out to explain and exemplify a series of presuppositions that form the basis not only of further ideas developed in the text, but also of his whole systemic approach. The book was written by the end of his life, and according to his daughter Mary Catherine Bateson, co-author of some of his texts, it represents the real synthesis of his work and "the first of his books' composed to communicate with the nonspecialist reader" [3]. Ramage and Shipp summarize some of his main contributions as a systems thinker and note that the whole pattern of his work is still in the process of being understood and appreciated [8].

Bateson was concerned with *mental process* and how looking at it from a very different perspective could shed some light on different complex processes in nature. Apart from discussing his ideas in relation to musical composition with open forms, with it's consequences for the construction of graphical scores (those that use non-traditional musical notation) and the understanding of musical performance situations that imply improvisation by the performers, I have tried to implement them as the basis for interactive processes in live-electronics. This approach draws its perspective from *holism*, which in Bateson's definition is: "The tendency in nature to produce from the ordered grouping

[1] My doctoral research has been kindly supported by a scholarship from the German Academic Exchange Service (DAAD), that enabled an internship at the Universität der Künste (UdK-Berlin, under Prof. Dr. Dörte Schmidt). This has been a joint effort with the Brazilian Government, through my leave at Universidade Federal do Rio de Janeiro (UFRJ) and my doctorate at Universidade Federal do Estado do Rio de Janeiro (UNIRIO, under Prof. Dr. Carole Gubernikoff and Prof. Dr. Vania Dantas Leite).

of parts complex wholes with properties that are not present in or predictable from the separate parts" [3]. M. C. Bateson notes that Gregory frequently used the term and its adjective "holistic" to refer to modes of acting and observing that are attentive to holistic properties. These wholes have characteristics of self-organization, that are intimately linked to the presence of *circular feedback chains*, responsible for the dynamical self-regulation of the system. That is, different interacting components mutually affecting each other show emergent properties and characteristics that are sustained by their interaction [6]. The idea of *emergence*, in this sense, presupposes the understanding that that which emerges is: *irreversible*, can only be studied taking *time* into account and cannot be replicated; and *irreducible*, it resists a study through the reduction to its smaller parts. For Morin, "what is important in emergence is the fact that it is indeductible from the qualities of the parts, and thus irreducible; it appears only parting from the organization of the whole" [7]. Demo stresses that complex phenomena "produce modes of being that are always of becoming as well. They behave in a reconstructive way: they do not reproduce themselves linearly, they reconstruct themselves non-linearly" [5].[2]

For complexity studies, the whole is more than the sum of its parts, but it is, *at the same time*, less than the sum of its parts, whose qualities and properties can be inhibited by the organization of the whole. As Di Scipio puts it, "once the whole has emerged, it can reduce or inhibit the action of specific individual components. Take, for instance, a large-scale social organization: however complex, varied and efficient in its performance, the whole organization can never be as rich and complex as each single human being participating in it. It induces simplified, typified behaviors that are functional to the whole, but detrimental to the individual" [6].

Looking for complex behavior in the construction of the live-electronics algorithm in *gosto de terra*, I have focused in four of Bateson's presuppositions regarding different aspects of the functioning of circular feedback chains in mental process.

2.1. Difference and information

First, for Bateson "information consists of differences that make a difference" and difference "is a non-substantial phenomenon not located in space or time." He stresses that to produce news of difference, i.e., information, there must be a relationship between two different parts of a system, or the same part in two different moments, "such that the difference between them can be immanent in their mutual relationship" and the whole process be such that news of that difference can be represented inside an information-processing entity, such as a brain, an ecosystem or a computer [1]. To better clarify, it is important to understand that the difference that makes up information is a *class* of some sort of differences, immanent in the relationship of the involved parts. It is a second order difference, which can only make sense if regarded in its context. And that, for Bateson, "change" can be seen as a "difference which occurs across time" [2].

2.2. Number and quantity

Another presupposition that Bateson points out in *Mind and Nature* refers to the difference between number and quantity. What matters to him in this difference are not really their names, the way we describe these categories in words—number and quantity—, but the fact that the formal ideas behind them are immanent in the processes observed. For him, "number is of the world of pattern, gestalt, and digital computation; quantity is of the world of analogic and probabilistic computation" [1].

Drawing from examples coming from organic life to explain the concept, he refers to a flower that has five petals and many stamens. The number of petals will stay the same from one individual to another, but the number of stamens will vary enormously. Bateson tries to explain that it seems clear in biological terms that the pattern differences of smaller numbers, like three and five for example, are drastic and form even important taxonomic criteria. On the other hand, after a certain size of number, they become quantity, meaning that for the organism there is a different process going on for that part of its growing form: "numbers can conceivably be accurate because there is a discontinuity between each integer and the next. Between *two* and *three*, there is a jump. In the case of quantity, there is no such jump" [1]. He asserts that this difference is basic for theoretical thought in behavioral science, since it reveals two very different ways of conceiving the relation between parts of organism or between parts of processes in nature. For him, numbers are the product of counting and quantities of measurement, and therefore always approximate.

2.3. Digital and analogic

Extending that concept to the difference between digital and analogic systems, he writes: "digital systems more closely resemble systems containing number; whereas analogic systems seem to be dependent more on quantity" and goes on to clarify that in digital systems there is a discontinuity between "response" and "no response", yes and no, on and off [1], 1 and 0. Its parts function like a *switch*.

Looking at a switch from the point of view of the circuit, it does not exist when it is turned on, since in that case it is not different from the rest of the conducting wire. Similarly, when the switch is off, it also doesn't exist from the point of view of the circuit, since the conductors themselves only exist as conductors when the switch is on. "In other words, the switch is *not* except at the moments of its change of setting, and the concept 'switch' has thus a special relation to *time*. It is related to the notion 'change' rather than to the notion 'object'" [1].

[2] "(…) produzem modo de ser que são sempre também de vir a ser. Comportam-se de maneira reconstrutiva: não se reproduzem linearmente, reconstroem-se não linearmente."

2.4. Quantity and pattern

In trying to explain what he understood as pattern, Bateson points out the phenomenon of *moiré* among other examples, which can be explained by looking at Fig.1, where a third pattern *emerges* (to the right) from the superposition of the other two. Moiré phenomena are actually a classical example of emergence and to truly comprehend its implications, one has to try the superposition (e.g., with two sheets of transparent paper with the pattern to the center printed on them) and see how a slight change in the rotation angle can yield a drastic change in the resulting pattern.

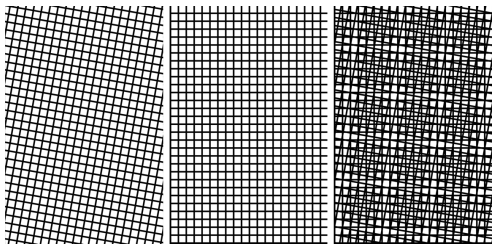


Figure 1. Moiré phenomenon: a third pattern emerges from the superposition of other two. (Source: Wikimedia Commons, Moiré_grid.svg, modified for the purposes of this text.)

For him: "First, any two patterns may, if appropriately combined, generate a third. Second, any two of these three patterns could serve as base for a description of the third. Third, the whole problem of defining what is meant by the word *pattern* can be approached through these phenomena." What triggered the solutions used in the constructed algorithm, was Bateson's observation that "*a ratio between two quantities* is already the beginning of pattern" [1]. So, if it would be possible to measure quantities in a digital system *and* gather information about differences in a given quantity in time, then quantities could be compared and a ratio between them may show an emergent pattern.

3. FORMALIZATION IN LIVE-ELECTRONICS

The patch[3] for *gosto de terra* tries to implement these ideas in a live-electronics setting, understanding it as part of a system.

3.1. The system, of parts and their interrelationships

Bateson clarifies that "the basic rule of systems theory is that, if you want to understand some phenomenon or appearance, you must consider that phenomenon within the context of all *completed* circuits which are relevant to it" and that a system "is any unit containing feedback structure and therefore competent to process information" [4]. Following the model drawn from systemic thinking, that a system is constituted by its parts *and* the interrelationships between them, the algorithm presupposes that this system is formed not only by the algorithm and its formalization in a Max/MSP patch, but also by the piano itself, with the sounding result of the interaction between the performer and the instrument, in a given acoustic space. The interactions between these four parts of the system (algorithm, performer, instrument and acoustic space), in turn, are dependent on the interface between: a given way of capturing digital data from the live performance and of returning the result of the algorithm to the acoustic space, i.e., to both the performer and the sound response of the instrument. In this case: a microphone is used to transduce typical aspects measured from sound (frequency and amplitude) and make them available as numbers to the digital system through FFT (Fast Fourier Transform); and loudspeakers that project sound back to the acoustic space, yielding possible sound responses of

Figure 2. User interface for the patch of *gosto de terra*.

[3] Fig.2 shows the user interface of the patch for *gosto de terra*, that can be obtained contacting the author. The score of the piece may also be obtained in the same way.

the instrument, by acoustically exciting vibration of the strings, but also responses of the performer that is encouraged to listen and respond musically to the sound result of all parts involved.

3.2. The "stable?" and "unstable?" algorithm: a jump between number and quantity

Typically, a digital system following the flow of data of a live performance in time returns a flow of numbers, but doesn't tell much about quantities in that flow. To bridge that gap, the algorithm compares the flow of numbers in *time*. Comparing a measurement of numbers in a given moment of the flow to the same measurement on a immediately subsequent moment, it returns equality (1) or inequality (0) of these measurements. This information is gathered for a relatively much longer time span and taken as a *set* of results. This set is then compared to a threshold, and the algorithm programmed to return the state of the set according to that threshold: if the quantity of equalities is above or below the threshold.

For example, analyzing the subpatch shown in Fig.3 from top to bottom, as this is the direction of the data flow in the patch, we have the following steps:

1) The flow of frequencies captured by the microphone, expressed as real numbers (floats), is analyzed and the result passed around every ten milliseconds (*speedlim* object). The *mean* value of about the last hundred values is rounded to a natural number.
2) Two subsequent results in time are fed through the *bucket* object to the "= =" (exactly equal to) object, which compares them and returns one (1) for an equality or zero (0) for an inequality. The second *zl stream* object gathers these results into a list.
3) The list of ones and zeros is shown in a *message box*, for a visualization of the set of results in time.
4) The quantity of ones in the set (*zl sum* object) is compared to the threshold through the ">" (greater than) object, which returns one (1) if it fulfills the condition or zero (0), if it doesn't.

In other words, the algorithm returns if frequencies are stable (not changing) in the performance, at any time. If instead of greater than (>) a smaller than (<) comparison to the threshold would be used, the same algorithm would return the information that the frequencies are unstable (changing in time). The result, one (1) or zero (0) according to the fulfillment or not of the given condition for the comparison to the threshold, is shown in a toggle object, that goes on and off (seen right below the first outlet of the patch, in Fig.3). This change of state triggers other algorithms in the live-electronics.

An information about the performance is gathered from a difference in quantity, according to a threshold. This difference is the result of a series of differences gathered from the flow of numbers in time, it is a second order difference, that tells something about the musical discourse, as it happens.

Since the accuracy of numbers does not matter much for the gathering of information about quantity, the flow goes through a series of approximations, so as to be useful for the comparison between numbers representing the flow of data in time. That is to say that the algorithm presupposes an amount of "coarseness" as necessary, or the jump to quantities wouldn't be possible from a flow of numbers. This, on the other hand, gives a certain "organic" characteristic to the result, that mirrors something like the "overall" characteristics of moments of the performance and not the accurate numeric details.

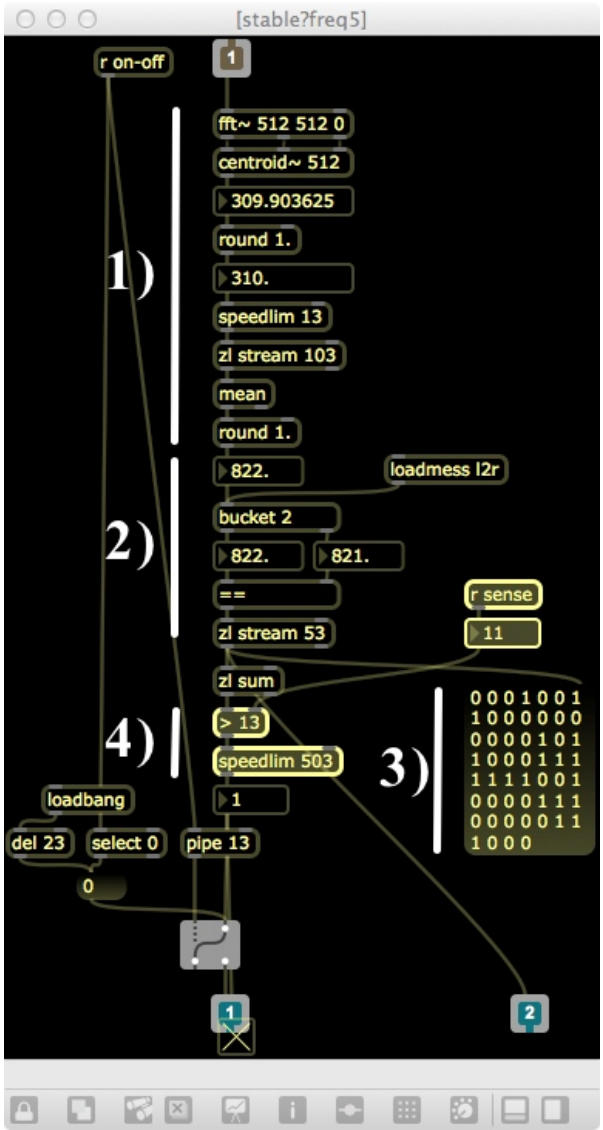


Figure 3. One instance of the subpatch "stable?" in the patch of *gosto de terra*: 1) data is treated; 2) compared in time; 3) visualized as a set of results; and 4) compared to a threshold.

A control of the number given as threshold can be seen as a control of the "sensibility" of the algorithm: a smaller number as threshold, makes the algorithm react to a lesser quantity of differences (more sensible), and a bigger, to a grater (less sensible). The value of the "sensibility" control, seen at the lefthand side of the user interface in Fig.2, is sent to the algorithm via the *receive* object named "r sense", highlighted in Fig.3.

3.3. Triggering other instances of the patch through the algorithm: the use of a third order difference

Following this, a second instance of the same algorithm is applied to the same flow of data in the patch (see Fig.2, "p stable?freq5" and "p stable?freq7"), but with a different time span. In Fig.3, the time span is given by the highlighted *speedlim* object with the argument 503 (five hundred and three), which means that the result of the comparison with the threshold, from the ">" (greater than) object, will pass only about every half a second (five hundred and three milliseconds). In the second instance mentioned here, this argument is set to 701 (seven hundred and one) milliseconds. The results of both instances are then coupled, and the big *toggle* object shown in Fig.2 —at the center of the patch, right above the window that shows a wave form, where a written information is given: "when stable: record"—, will only change its state if the value (1 or 0) of both instances coincide at any point in time. A third order difference is achieved through another comparison or ratio between quantities. The information about the stability or instability of frequencies in the performance is double-checked in different time spans. As indicated by the written information in the user interface, if both instances return an equal result, at any point in time, the recording is on, if unequal, the recording is off: the patch starts recording the performance when the algorithms detect any stable frequency that comes through the microphone for about half a second. The waveform of the recording is shown in the window below the toggle and, of course, changes during the performance.

The systematic use of prime numbers as time spans or number of elements in sets of numbers that refer to quantities, is a way of trying to avoid temporal constraints in the functioning of the algorithm (and, in fact, of the whole live-electronics system for *gosto de terra*) that would resemble an idea of time as isometric. The characteristic of prime numbers, i.e. their property of only being divided by themselves or one, increases the possibility that there will **not** be any co-incidence (literally understood) of events in time, reinforcing non-linearity —a characteristic also found in organic systems — and, in that way, the feeling of organicity implied above when referring to "coarseness".

3.4. Circular feedback chains

It is interesting to note, that the microphone is used in the system not only to record sound from the live performance, but also as a kind of sensor. It is used for its capacity to sensor differences in time, transduce them and make them available to the algorithm described.

The recorded excerpts of the performance are used in a special convolution process with the sound result of the piano, i.e., of what is being played live by the performer. When the recording is triggered by the algorithm, it will obviously record anything that comes through the microphone, feeding the convolution with chunks of sound that have stable frequencies, but that also carry with them any musical content sounding at that moment, including sympathetic vibration of the strings, sounds in the piano body, anything played by the performer, acoustic responses of the performance space and so on.

Circular feedback chains are created, where the live-electronics influences itself, but is also influenced by the performer's intentions, through listening and playing, responding to the whole sound space created. On the other hand, the performer has a certain control of the response of the algorithm through the performance itself. It feels possible to control the electronics by the very playing of the instrument. The patch helps the process, showing to the performer in written text, but also visually, what is happening in the algorithm.

In that way, one interesting result that may be the indication of the pattern that emerges from the compared differences in quantity, is the small window with vertical lines to the right side of the big *toggle* (Fig. 2). Every vertical line is the representation of a 1 (one) from the *set* of ones and zeros gathered by the algorithm in the *message box* linked to step 3) of the description of the algorithm in 3.2 and Fig.3. These lines move from right to left when the sound processing of the patch is on. When frequencies are stable, many "ones" are generated by the two instances of the algorithm, and more lines move in the small window, with more density in time, emulating a flux that gives a hint of when the recording will be triggered or not.

In the same way, the piano becomes active in the system when there are sympathetic vibrations of the strings, as does the acoustic space, resounding with its particular acoustic response what is fed to it by the performance. The piece adapts itself to all this factors without loosing a certain identity: there is a recognizable sound characteristic of the whole process, tied to what the score defines as the musical content to be sought, and dependent and independent, at the same time, of all parts of the system and their interrelationships.

4. CONCLUSIONS

The described algorithm is, of course, only one way of looking at the formalization of these presuppositions in systemic thinking. Dealing with concepts as difference, number and quantity in such a way, seems to point to the possibility of developing further formalizations that may be responsible for the emergence of complex patterns, specially if ratios between quantities could be formalized in such a way that they themselves yield second, third and maybe higher order information.

5. REFERENCES

[1] Bateson, G. *Mind And Nature: A Necessary Unity.* Hampton Press, Cresskill, 2002.

[2] Bateson, G. *Steps to an ecology of mind.* University of Chicago Press, Chicago, 2000.

[3] Bateson, G., M. C. Bateson, *Angels Fear: Towards an Epistemology of the Sacred.* Hampton Press, Cresskill, 2005.

[4] Bateson, G., R. E. Donaldson (Ed.), *A Sacred Unity: Further Steps to an Ecology of Mind.* Harper Collins, New York, 1991.

[5] Demo, P. *Complexidade e aprendizagem: a dinâmica não linear do conhecimento.* Atlas, São Paulo, 2011.

[6] Di Scipio, A. « Listening to Yourself through the Otherself: on *Background Noise Study* and other works. » *Organised Sound.* Cambridge University Press, Cambridge, vol.16, n.2, 2011.

[7] Morin, E. « Restricted Complexity, General Complexity. », *Worldviews, Science and Us, Philosophy and Complexity.* (Ed.) Gershenson, C., D. Aerts, B. Edmonds, World Scientific, London, 2007.

[8] Ramage, M., K. Shipp, *Systems Thinkers*. Springer, London, 2009.

BILAN ET PERSPECTIVES DE LA REVUE FRANCOPHONE D'INFORMATIQUE MUSICALE

TABLE RONDE

Anne Sèdes - coordination
MSH Paris Nord, AFIM.
mshpn-rfim@mshparisnord.fr

Sous l'égide de l'AFIM, la Revue francophone d'informatique musicale[1] publie des articles théoriques et expérimentaux originaux, des rapports de recherches et des travaux de synthèse en relation avec les domaines liés à l'informatique et à la création musicales. La revue est annuelle, voire bisannuelle et étudie tout au long de l'année les propositions de publication après soumission au comité de lecture. Une large place est donnée aux travaux des doctorants et jeunes chercheurs.

La revue choisit, sans exclusive, des thématiques émergeant des journées d'informatique musicale (JIM), ou des Sound and Music Computing (SMC), ou plus généralement de l'actualité scientifique et artistique.

Depuis 2010, trois numéros annuels sont parus, un quatrième numéro est en cours de préparation.

Le numéro 1 concernait la thématique de la mixité et du *live electronic* (actes d'une journée d'étude consacrée à ce sujet en mars 2011 à la MSH Paris Nord) ainsi que des travaux de jeunes chercheurs[2].

Le numéro 2 a donné une large place à la préservation des œuvres utilisant les technologies numériques, approfondissant ainsi la thématique principale des journées d'informatique musicale 2011 qui avaient eu lieu au Cierec à Saint-Étienne[3].

Le numéro 3 a abordé la mise en espace du son, la musicologie des pratiques électroacoustiques employant des dispositifs numériques, le geste sensible lié à la musique interactive dans une approche somatique ainsi que l'expérience de l'interprète créateur d'œuvres mixtes[4].

Les jim 2014 seront l'occasion de faire un premier bilan de la revue, après trois années d'existence, de présenter le projet de numéro pour l'année 2014, et puisque l'AFIM s'ouvre à la francophonie, suite aux JIM 2012 à Mons, et dans la perspective des JIM2015 à Montréal, de questionner l'usage de la langue française dans notre communauté pluridisciplinaire de chercheurs et créateurs.

Ce bilan se fera sous la forme d'une table ronde qui réunira :

Myriam Desainte-Catherine, Labri Bordeaux
Yann Orlarey, Grame, Lyon
Laurent Pottier, porteur du prochain numéro de la revue, CIEREC, St-Etienne.
Alain Bonardi, CICM, EA 1572, université de Paris 8.
Pierre Michaud, porteur du projet JIM 2015 à Montréal, LITEM, UDM.
Julien Rabin, coordinateur des JIM, GMEA, Albi.
Anne Sèdes, responsable de la publication (CICM,EA 1572, université de Paris 8, MSH Paris Nord)

[1] revues.mshparisnord.org/rfim /
[2] http://revues.mshparisnord.org/rfim/index.php?id=69
[3] http://revues.mshparisnord.org/rfim/index.php?id=183
[4] http://revues.mshparisnord.org/rfim/index.php?id=243

Annexes

SONIFICATION AND ART

DOMINANTE DE LA JOURNEE (KEYNOTE)

Peter Sinclair
Locus Sonus ESAA
petesinc@nujus.net

ABSTRACT

After a brief definition of sonification I will discuss two creative approaches – that of Sound design and that of conceptual art. I will attempt to see how these differ and converge in their goals and methods particularly in the context of what we might call musical sonification. I will describe some examples of key sonification artworks and the aesthetic strategies they employ. Finally I will describe *RoadMusic* a project that I have been working on for some time, and the practice to which I apply my ideas concerning sonification.
After a brief definition of sonification I will discuss two creative approaches – that of Sound design and that of conceptual art. I will attempt to see how these differ and converge in their goals and methods particularly in the context of what we might call musical sonification. I will describe some examples of key sonification artworks and the aesthetic strategies they employ. Finally I will describe *RoadMusic* a project that I have been working on for some time, and the practice to which I apply my ideas concerning sonification.
It has been suggested in the past that sonification as a term should exclude artistic and musical usage of sound. I am referring here to an article by Sonification expert: Thomas Hermann that can be found in the 2008 ICAD proceedings [10]. In a special edition of AI&Society dedicated to artistic sonification that I guest edited [12], several artists and composers contradicted this position and Hermann himself has revised his point of view since contribution to the review which he co-signs starts with: 'Sonification today is an interdisciplinary practice ranging from scientific applications to sound art and composition [9].'

1. INTRODUCTION

I do not wish to labour a point that has to a certain extent been resolved, however will serve as a starting point for this discussion of the aesthetics and semiotics of sonification. The definition of sonification most commonly cited is that proposed during the ICAD[1] conference in 1999: 'Sonification is the use of non-speech audio to convey information [15]' or in other words the mediation of (most commonly digital) data through sound.

Conveying information through non-verbal sound has a long history. Alerts and alarms ranging from the sounding of the post horn or the chiming of a clock to the warning given by a siren might be considered as precursors to sonification. We can recognise the traits of auditory perception that make it particularly well adapted to these types of signals: hearing is both omni-directional and always available; we cannot close our ears, even when sleeping. And a part from startle responses there are other primitive mechanisms at work, distinguishing variations in multiple streams, without demanding our conscious attention. There are clear advantages in exploiting these faculties in an environment where our visual attention is increasingly monopolized by electronic screens or other technical tasks such as driving.

A well-known example of sonification is the Geiger counter where the frequency of clicks designates the intensity of radioactivity. The nature of the clicking sound of the Geiger counter is the result of its pre-digital electronic circuits, today however, sonification almost inevitably involves choice: that of what sound or which variable parameters of sounds to map to otherwise silent data. Sonifications are pervading our environment and the necessity to distinguish between them is giving rise to complex and sophisticated techniques where analysis of function, and subsequently design, play an increasingly important role. To give another familiar example, a mobile telephone no longer 'rings' but notifies the user by playing a specifically chosen sound, which distinguishes his or her telephone from those of others. It probably plays different sounds for different types of alerts (or even to identify different callers) and beyond this, recent 'smart phones' provide carefully tailored feedback sounds for touch screen functions.

Different principles or techniques of sonification can be identified:

- *Audification* is the direct transposition or transduction of a signal into the audio domain. Audio-biofeedback is an example where sensors connected to a subject's muscles or skull capture electrical impulses that are amplified

[1] International Community for Audio Display

directly and played through a loud speaker as an audio signal. It is also a technique exploited by several artists (myself included) often because it is perceived as being as close as possible to the physical reality of captured phenomena.

- *Mapping Based Sonification* modifies parameters of a sound such as pitch or amplitude. An example is the pulse-oximeter that monitors a patient's blood oxygen saturation as pitch and pulse rate as tempo [20]This is also perhaps the most obvious method for creating musical sonifications since MIDI based, digital music systems are inherently adapted to accept a variable input driven by data.

- *Earcons* are usually short tones, combinations of tones or simple melodies an example might be the jingle preceding an announcement on the PA of a train station.

- *Spearcons* are time compressed speech samples, which can be played at speeds where they are no longer recognizable as words. They represent the advantage of being non-arbitrary, easy to create and easy to learn.

- *Auditory Icons* have a symbolic relation to an action they represent, examples are to be found providing feedback on personal computers, for example: the sound of crumpled paper falling in a waste paper bin used to indicate that a file has been moved to the trash folder.

If we consider these different techniques we might also place them on a scale that goes from direct (almost unadulterated) information through abstracted information to highly symbolized or metaphorical information. Paul Vickers in his 2006 paper *Sonification Absraite/Sonification Concrète: An Aesthetic Perspective Space For Classifying Auditory Displays In the Ars Musica Domain* [24] suggests that we might consider these from the position of electroacoustic music as a slider between Schaeffer / Chion's "reduced listening" and "causal listening". That is to say that within an audification we listen to the purely acoustical properties of the sound and glean information from them. Whereas in the case of an auditory icon we associate the symbolic nature of the sound with a function –mapping based sonification probably fits somewhere between these two. Vickers suggests that musical knowledge is an advantage in sonification design because in a sense it is what composers do all the time.

I would add to this a thought –that comes from friend and fellow composer and sonifier Peter Gena– which is that until very recently music was an affair of data mediation before being one of sound [7]– sound only started to be used in its own right as musical material with the advent of recording and electronically produced sounds. Until then –at least for a long chapter in the history of western music– a composer would right notes (data) to be interpreted by musicians.

2. AUDITORY SCENE ANALYSIS

An approach to sound and musical perception that I have found valuable for sonification design – specifically when there is a situation where multiple sources of information are being sonified- is the Auditory Scene Analysis theory as proposed by Albert Bregman [4]. This considers that there is a primitive aspect of audition, a sorting level so to speak, which is prior to and independent of cultural influences on listening. It is not however the simple reflex/startle response that I evoked above, rather it is a relatively complex mechanism or collection of mechanisms, which have evolved in response to our environment and possibly to our neurological system. It is an intermediary located between the capture of sound through the basilar membrane and the construction of significance through schematic memory and culture. Bregman does not deny the existence of higher 'from the top down' mechanisms, he maintains however that they are built with and on top of a primitive segregation, that he calls 'Auditory Scene Analysis'.

It would be too long here to go into the details of ASA but the basic question to which it provides a response is: How does the ear identify coherent auditory objects from the cross section of incoming sound, the simultaneous mash-up of frequencies? How from the immediate incoming vibrations does our mind distinguish those parts of the frequency spectrum belonging to one source from those belonging to another? If it might seem obvious to us that different sound qualities belong to a same source, from the cognitive point of view it is a complex problem. Each sound we perceive is made up of a multitude of frequency components spread across the spectral range of our hearing, so although we know that the first step in auditory analysis (the cochlea) breaks up the incoming signal into frequency bands, it is unlikely that we are able to identify a sound through this mechanism alone. Bregman demonstrates how principals of gestalt construction apply to sound by using auditory illusions that show how we can be "tricked" into associating elements of the audio spectrum with one another. ASA does not deal with sound objects or even sound sources but rather with 'streams' which can be continuous sounds or repeating sound events (such as notes or footsteps). Bregman studies in great detail the characteristics that tend to group elements such as closeness in time and pitch and those that separate such as non synchronous onset or timbral incoherencies. The interesting thing here is that ASA can provide a method to compose multiple sounds (pitched, musical sounds or other) and predict their interaction (ie what will be considered as separate and what will be grouped as a

same source) even when the sounds are variable and the precise score or mix is unknown in advance. Since ASA mechanisms are pre-cultural using them as a basis for organizing sonification is arguably less risky than a more traditional musical approach that might be misinterpreted by part of the population and since it allows for variation it is possibly richer and less annoying than auditory icons. This last point is important – I argue against the use of recorded samples in sonification applications since (personally) I find repeating sounds rapidly become annoying and if they have the advantage of being unambiguous they are of limited scope in terms of the information they can carry. This might seem to be an argument in favour of audification as a technique and to a certain extent it is – however several non-treated (noisy) audifications might become difficult to distinguish and at the very least require a lot of training or experience to become interpretable.

In the recent symposium I organized on audio mobility Gaëtan Parseihian, researcher at CNRS-LMA, described his research into *The process of sonification design in the case of guiding tasks* [17]. The processes use parameter-based sonification and synthesis to provide feedback information gained via visual capture for (among others) blind people. Gaëtan apologetically explained that blind people where reticent to use the system since a/ in encumbered their (normal) audio perception and b/ they found they the sounds un-subtle and disagreeable. I must admit that I sympathize, for instance I found the sound attached to the reversing sensor on the car I was driving last week, efficiently understandable but insultingly oversimplified and disagreeable in its texture. I would argue that while there is a case to be made for the clear transmission of information through sonification and that it is important to be able to distinguish between different sound sources, it is arguably not always useful to oversimplify and over filter incoming data, and that in certain cases at least alternatives to auditory icons, earcons or rudimentary mapping based sonification can be preferable. There are undoubtedly different ways to approach this problem, I will return to the solutions that I have adopted later.

3. ARTISTIC PROJECTS

Up until this point I have been discussing questions related to sonification design; that is to say the art of crafting sounds for a certain purpose or to mediate certain data in an appropriate way. However there is a different approach possible to the art of sonification, which involves considering the holistic artistic approach that evolves when projects, rather than responding to a design problem, adopt sonification as the means to artistic ends. In these situations, often it is the source of the data itself that becomes significant and the relationship between that source, the situation and the artists mediation that makes the art work. In many cases, these are in a conceptual continuum with the ideas and techniques developed by composers of the mid twentieth century who sought the emancipation of (their) music in regards to (their) expression. I am thinking here in particular of John Cage –his inclusion of the everyday into the compositional process and his definition of experimental music as a being of a kind which the composer discovers at the same time as the other auditors [5]– but also of Stockhausen, Xenakis and the serialist composers who gave *process* a dominant role in composition. Moving on in time I would also include Murray Schafer [19] and others involved in soundscape listening and sound installation, which tend to shift the focus away from the composer's imaginary projection and towards a decision making process that includes real-time and real-place. Digital technology makes this emancipation all the more feasible since it can mediate the everyday, render audible the imperceptible or anchor artistic form in real-time and in real-place.

3.1. Beyond Expression

There is a recurring idea in sonification artworks, that by externalising some of the 'expression' in the work; by allocating a responsibility to data, there is a shift of sensibility away from the individual artist or musician, creator of the work, and towards the environment being sonified; the artist adopts a different position. This is a trend, which can be noted in much digital art, where artists see themselves as an element in (as opposed to the author of) a process. There is also a strong ecological presence in sound art, manifested through practices such as field recording and sound walking considered as being less dominant or imposing forms than traditional composing and in some cases sonification can be seen as a continuation of this idea.

Andrea Polli for example uses sonification to render public significant but normally imperceptible data. Her intentions are overtly political; she compares sonification to soundscape listening and sound-walking, which she considers as essentially socially engaged activities [18]. For Polli, audification can readily be considered as an extension to soundscape, since it is the simple translation of non-audible vibration into audible vibration. In comparison, mapping-based sonification (more specifically 'geosonification'[2]) potentially poses other problems, since the translation involves higher levels of human intervention and therein a danger of over-simplification; an inevitable subjectivity appears through the choices involved in the mapping process.

[2] Geosonification: the sonification of data from the natural world inspired by the soundscape.

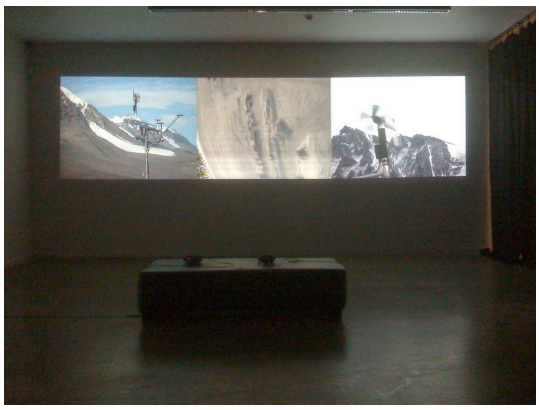


Figure 1. Andrea Polli, *Sonic Antarctica*, 2008

Sound and music are readily associated with cosmology and holistic thinking. The idea that they are representative of the higher (and lower) order of things is sporadically recurrent in Western Philosophy and musicology from Pythagoras onwards. The universality of sound is also largely reflected in non-western philosophical traditions such as Sufism, as this quote from *The Sufi Teaching of Hazrat Inayat Khan* illustrates:

> Since all things are made by the power of sound, of vibration, so every thing is made by a portion thereof, and man can create his world by the same power. Among all aspects of knowledge the knowledge of sound is supreme, for all aspects of knowledge depend upon the knowing of the form, except that of sound, which is beyond all form. (Khan 1996)

That sound and music are appropriate forms to vehicle information that is otherwise imperceptible because too vast to perceive, is a dominant concept in several examples of artistic sonification. Lorella Abenavoli sonifies the vibrations of the earth, compressing them in time in such a way as to render them audible. With her installation *Le souffle de la Terre* Abenavoli invites us to 'drop in and listen to the earth' (Abenavoli 2004) Marty Quinn has worked with NASA using data from solar storms as a source (Quinn 2011), and Richard Kroland-Martinet, Solvi Ystad & Mitsuko Aramaki, sonify cosmic particles —invisible but constantly present in our environment (Aramaki, et al. 2009).
On the other end of the spatial scale Victoria Vesna has collaborated with nano scientist James Gimzewski, creating sonifications of the metamorphosis of a butterfly. Originally stemming from an audification technique used by Gimzewski to display scientific data (the evolution of yeast cells), the Californian artist/scientist couple present this perception of the infinitely small as a public installation entitled *Blue Morph*, which reflects their holistic philosophy:

> As many speculative ideas in the West circulate around ideas of energetic approach to matter in general, particularly the body and mind, alternative medicine and other Eastern philosophies are thriving. …We have investigated these ideas from the sounds of cells to the concept and realization of the Blue Morph installation at the Integratron[3] [23].

Nicolas Reeves, son of the famous French/Canadian astrophysicist and ecologist Hubert Reeves, applies his training in architecture and physics to the domain of media arts. His installation *Cloud Harp* sonifies by reflecting laser beams off cloud cover, using a technology similar to that of a cd player [22]. The artist qualifies his work as 'Keplerian' (in reference to the German Astronomer, who incorporated cosmology into universal mathematics and revisited Pythagoras' *Music Of The Spheres* in his 1619 publication *Harmonices Mundi* [13]). For Reeves, 'Cloud Harp' reflects a social political and ecological position that advocates the integration of human technologies into the greater order of things. The work can be said to 'listen' the environment rather than imposing itself on that environment as such, we can hear an echo of the influence of Murray Schaffer [19].

Another example of sonification, that evokes both scale and in a certain sense politics, is Jens Brand's *Global Player*. Presented as a commercial brand, *Global Player* has a dedicated website [3] that uses spoof advertising to present a genuine technique that consists of sonifying the relief of the earths' surface, from data supplied by orbiting satellites. His advertising slogan is 'Brand – We Play The world– What Are You Playing?' The website offers two products the GP4 'an exclusive, top-notch Hi-Fi product which plays the earth as a disc' and the *G-POD* a portable version of the *Global Player* which looks (suspiciously) like an apple *iPod*.

3.2. Audience Reception

When outside of the domain of pure sound design the perception of sonifications is not necessarily that obvious. If we hear the sound of the earths vibrations compressed in time is it possible to perceive them as such, intuitively, without being informed of their origin by an oral or written explication? If we hear music generated from the pattern of DNA (Gena, DNA Music 1995) is this perceivable in itself? I would say that it probably isn't and in most cases of artistic sonifcation that I have encountered the source of the data is given to the audience through some other means than the sonification itself. The source of the data becomes a signifier or perhaps something akin to the program in program music[4].

[4] Program music appeared in the nineteenth century and consisted of a symphony accompanied by a text that can vary in complexity from a simple title (for example *Tragic Overture* as opposed to *Symphony n°...*) to a text several pages long,

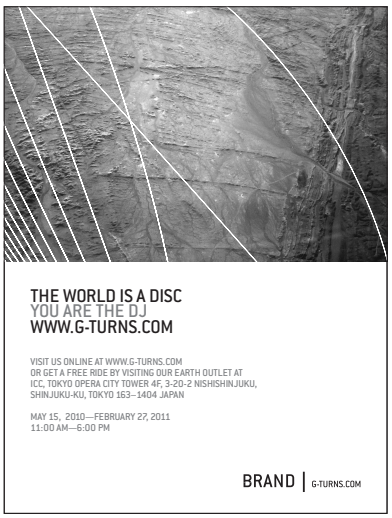


Figure 2. Jens Brand, Brand - finest Global Players since 2004.

If understanding the provenance of the data is integral to appreciation of the artistic proposition, how does the artist make this information available? Alternatively, if the provenance of data is not integral to appreciation, how can we consider that the data is significant? Does attaching a created sound to a source remove it from the realm of 'pure' music?

Take London-based composer John Eacott's work *Floodtide* [6]. I witnessed a performance of this piece in July 2010. The performance took place on a plaza in front of the Southbank Centre in Central London. In the place of music stands, the musicians had flat screen displays showing a 'score' generated by a computer in real-time. The program generating the scores was driven by tide-flow data gathered by a sensor plunged into the Thames below the performing musicians. Strategically positioned posters explained the process with the aid of visuals and a large screen indicated tide flow in knots. The music was agreeable, engaging and minimal in nature. I would place it aesthetically in the school of repetitive music. The performance spanned the six hours of the tidal turn. Although I did not experience the whole duration of the piece I happily spent two hours, half paying attention to the music, half taking in the surroundings. If there is no direct 'comprehension' of the state of the tide through *Flood Tide*, the ensemble –I understand by this the geographical situation plus the generated score plus the human presence of the performers plus the information provided by the posters– 'works' as a whole. The knowledge that the flow of the water next to us was participating in the performance introduced an extra dimension –an extension to the scale of the piece.

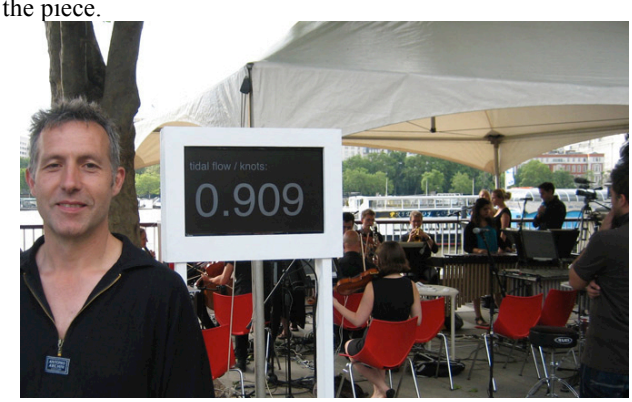


Figure 3. J.Eacott *Floodtide*, London, July 7 2010.

John Eacott considers that data derived from the flow of tide is more meaningful than data generated by stochastic computer processes, but this is not his primary motivation: he willingly talks of his family background in show business and the importance he attaches to spectacle. It would seem that it is this aspect that has pre-eminence over a more esoteric significance that might be suggested by his choice of elements. In the interview we conducted with Eacott he suggested that if today it is necessary to explain to an audience that sonification is taking place, in the future the audience might automatically come to expect that something is being sonified. Perhaps, as John Eacott suggests, some day real time sonification will have become conventional to the point where audiences will be expecting the music to be data-driven. If this becomes the case, it will undoubtedly change the audiences' receptivity to this kind of work but it seems that the data keeps its 'program' status as a signifier, 'what is being sonified tonight? – Ah ok we are listening to …' although it might shift progressively to something which one might guess at without the necessity to consult a poster.

So is artistic sonification necessarily conceptually based? I would venture that when the sonification is intrinsically related to a person's phenomenology then perhaps not.

Christina Kubisch is an international sound artist who has also taken an interest in the invisible and inaudible dimension of electromagnetic waves. Her *Electrical Walks* project is probably among the best known and documented of sonification art works. Publicly presented for the first time in 2003, it predates this in forms that are more experimental[5]. In her description of *Electrical*

pertaining to the composers' intentions (for example Hector Berlioz's *Symphonie Fantastique*). According to Peter Kivy this evolution can be traced to the influence of German philosopher G.W.F. Hegel '…who decreed, at the time the status of music as a fine art was being debated, that absolute music could not be a fine art without a content and could not have a content without a text (Kivy, 2002, p. 192).'

[5] In the Catalogue dedicated to Kubish's works entitled 'Electrical drawings' Christophe Metzger mentions a 1993 experiment where Christina Kubisch and composer Alvin Lucier 'scanned the streets of Tokyo with these modified headphones' [12].

Walks [11] Kubisch explains how, originally unsought after parasites that appeared in her installations[6] became the genesis of this work.

Figure 4. Christina Kubisch, Electrical Walks Nancy June 6 2011 (photo C. Kubisch).

As the quantity intensity and diversity of these parasites increased along with the proliferation of electronic apparatus, she became aware of the artistic potential of visiting this invisible and normally inaudible universe. She constructed special headphones equipped with induction coils, which allow the auditor to hear the electromagnetic waves emitted by different devices:

> Light systems, transformers, anti-theft security devices, surveillance cameras, cell phones, computers, elevators, streetcar cables, antennae, navigation systems, automated teller machines, neon advertising, electric devices, etc. [11]

Members of the visiting public are invited to don these headphones (see Figure 4-3) and discover the hitherto unperceived but ubiquitous and pervasive dimension of electromagnetic waves. This work is a notable exception to the idea that data is necessarily exposed as the conceptual basis of the artwork. I would argue that in this case the artwork is potentially self explicit and autonomous. Christina Kubisch offers maps to her public that indicate spots of particular interest, however, I propose that if one where to wear the headphones without prior knowledge of their usage, one would rapidly comprehend their functionality and ultimately be in a position to appreciate the artistic intention. For this to be the case though, the user must move around and arguably it is this mobility that generates the actual content of the piece.

This idea that if mobilized, a sonification can potentially "make sense" by itself is the basis of my project RoadMusic which I will now describe in more detail.

4. ROADMUSIC

The current version of *RoadMusic* runs on a dedicated mini PC or (since about a month ago) on an adroid telephone in both cases the device is that is fixed to the interior of a car's windscreen (like a GPS) and connects to the car's stereo via a standard audio input. The program uses sensors and a camera.

Figure 5. *RoadMusic* running under *Android* (Photo F. Hartmann).

4.1. How it works

The sensors gather information about movement (accelerometers on x, y, and z axes): vibrations from the car body and the road surface; and on a larger scale bends, bumps, accelerations and braking. The camera gathers information about the visual field: moving objects and colour.

Possibly the single most important (sonic and conceptual) feature of the *RoadMusic* program, is that at the outset there is no pre-recorded audio in the system[7]. It does not use playlists or samples; rather audio synthesis is based upon the incoming raw data –the forms of the road and the cars movements. This data is written into wavetables[8] and read at audio speeds, meaning that the sounds evolve as data updates.

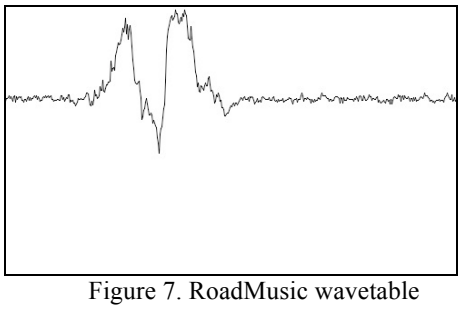


Figure 7. RoadMusic wavetable

This might immediately conjure up the idea something very noisy in the auditory imagination of the reader, however since this is just the starting point of the synthesis (the wavetable is then read by oscillators and

[6] Earlier installations use induction headphones to capture deliberately transmitted sounds as the visitor navigates in a designated space.

[7] There is an exception to this: a recorded voice which gives feedback information about user navigation.

[8] Indexed memory or array of values used in computer music.

numerous other processes are applied to it), it essentially effects the sound on a micro-sonic scale, modifying timbre or texture. The influence of this on the end result is subtle and not necessarily consciously quantifiable to the untrained ear but it eliminates the monotonous nature of many synthesised sounds and importantly *is* the audio 'imprint' of the journey itself.

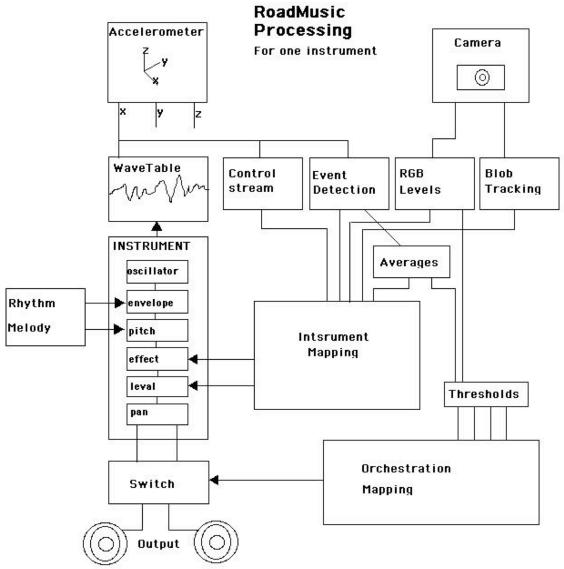


Figure 6. Schema of *RoadMusic* analysis and audio processing

4.1.1. *Instruments Or Streams*

At the next level of complexity up from the wavetables described above, audio synthesis is accomplished by a number of modules generating what will be described later in this thesis as multiple streams[9] –separate audio identities, which for the moment are most easily thought of as instruments. Each of these 'instruments' has basic parameters that can be varied in real-time such as pitch, amplitude and pan and some have more distinctive qualities such as frequency spectrum (tone filter), echo, delay, and 'Doppler'[10] effects that can simulate physical space or movement depending on the instrument.

4.1.2. *Data Analysis And Mapping*

Data from the accelerometers is processed in different ways:

- Each data stream is rescaled and smoothed so that it can be used as a continuous controller by any parameter of any instrument. For example, the varying force of acceleration and deceleration or g-force as the car goes round bends or over bumps, can be mapped to change pitch, level or harmonic spectrum.
- Events are detected within these same streams by measuring difference against time, so it is possible to discern a bump, a bend, an acceleration or a deceleration. These events are used to trigger sounds directly, to introduce or remove notes from melodies, or to switch signal processes on or off.
- These events are also used to calculate statistics about the road –measures of bumpiness, bendiness, stops and starts– that in turn produce new streams of data (moving frame averages). Like the rescaled data, these can be mapped to any parameter of any instrument, causing slower variations that mediate the drive on a macroscopic scale.
- A last level of analysis applies threshold values to the statistical data, producing a new set of events which are typically used to orchestrate the ensemble – switching different instruments on and off according to the type of road, for example: straight, some curves, winding; flat, bumpy, very bumpy.

4.1.3. *The Landscape*

A camera captures the visual field of the road ahead. This capture is analysed in two ways:

- Blob tracking is used to distinguish large moving objects, most often cars in the opposite lane. An object detected in the video frame is translated as coordinates: horizontal and vertical position and size. As with data from accelerometers, these can be mapped to sound parameters. In practice, they are employed to create the impression that a sound is moving correlation with an object outside the car by using psycho-acoustic cues (phase difference, amplitude and Doppler shift).
- Colour of the overall scene is analysed by extracting varying RGB (red, green, blue) component levels, typically used to vary harmonic elements in an instrument. Here too an event is extracted when there is a change in the dominant colour of the landscape and used to bring about changes in the music.

4.1.4. *Global Inactive*

A last but important detection is the non-event: the program provides a signal when no event has occurred for a few seconds, typically when the vehicle marks a

[9] See Chapter 3.3 : *Auditory Scene Analysis*.

[10] Phenomena whereby, when a sound source is moving rapidly towards or away from an auditor (for example a siren or train horn) the perceived pitch is modified.

stop (there is a mechanism which prevents this from recurring more than once per minute to cater for traffic jams and suchlike). This is used to define parameters of a new piece, for example, to re-generate melodic and rhythmical patterns. The compositional process, once the actual code for instruments has been completed, consists of deciding which input parameters are to be routed to which instrument parameters.

RoadMusic's aim is not one of (stricto senso) informing the driver about the road. However there is an intention to create an audio environment (although musical in its form) that puts the driver and passengers in contact with the situation of driving, the road, the car and the environment. Where it might be argued 'normal' listening in a car tends to a large extend to exclude these. Modern (in particular electric or hybrid) cars are isolated from the environment they traverse (even with the windows open at anything beyond minimal speeds turbulence will cover any sounds in the landscape) and car manufacturers do there best to eliminate and mechanical noise. Thus the car HiFi, and has become the by-default audio source, and in most cases unless giving traffic information it is disassociated from the situation. So in a sense RoadMusic applies design principals to a problem solving process. To this end I use the principals of Auditory Scene Analysis described earlier, in order to compose through routines or programs where the absolute result is unknown (the sound of RoadMusic is never the same). By thinking of the Auditory scene in terms of streams with identities and characteristics that separate or approach them it becomes possible to create compositions without knowing where they will start or end and what they will encounter in between. In this I also rejoin with John Cage's experimental music since the composer discovers the piece as it occurs and I would propose that the fact that all the music is generated from audification of the sensor data gives a live quality to the experience (something akin to acoustic sound). Finally I am also aligned to a certain extent with the cosmological camp at least in the sense that one of the aspects of this approach that interests me, is that rather than creating a fixed art work or a performance, the process is one of cybernetic inclusion. By this I mean that the driver the car the program and the environment play together and the limits of such a system are arguably unlimitedly extendable.

5. BIBLIOGRAPHY

[1] Abenavoli, Lorella. "Le Souffle de la terre, étude nº 5." Fondation Daniel Langlois. http://www.fondation-langlois.org/html/f/page.php?NumPage=268. 2004.

[2] Aramaki, Mitsuko, Charles Gondre, Richard Kronland-Martinet, Thierry Voinier, and Solvi Ystad. "Imagine the Sounds: An Intuitive Control of an Impact Sound Synthesizer." Edited by S. Ystad et al. Auditory Display: 6th International Symposium, CMMR/ICAD 2009. Berlin: Springer , 2009. 408-421.

[3] Brand, Jens. "GP4." http://www.g-turns.com/. 2004.

[4] Bregman, Albert S. Auditory Scene Analysis - The Perceptual Organization of Sound. The MIT Press, 1994.

[5] Cage, John. Silence lectures & writings. London: Marion Boyars, 1971.

[6] Eacott, John. "Flood Tide." http://www.informal.org/. 2009.

[7] Gena, Peter. "Apropos Sonification: a broad view of data as music and sound - Sonification - What Where How Why." AI&Society (Springer), 2011.

[8] Gena, Peter. "DNA Music." http://www.petergena.com/DNAmus.html. 1995.

[9] Grond, Florian, and Thomas Hermann. "Aesthetic Strategies in Sonification." AI & Society (springer), 2011.

[10] Hermann, Thomas. "Taxonomy and Definitions For Sonification And Auditory Display." Proceedings of the 14th International Conference on Auditory Display, Paris, France June 24 - 27, 2008. Paris, 2008.

[11] Kubisch, Christina. "Works with Electromagnetic Induction." Christina Kubisch. http://www.christinakubisch.de/english/klangundlicht_frs.htm (accessed december 10, 2011).

[12] Kubisch, Christina. "Electrical Walks." Kunsthalle Bremen. Christina Kubisch, Electrical Drawings, Works 1974-2008. Kehrer Verlag Heidelberg, 2008.

[13] Kepler, Johannes. Harmonices mundi (The Harmony of the Worlds). 1619.

[14] Khan, Hazrat-Inyat. The Mysticism of Sound and Music. Shambhala, 1996.

[15] Kramer, G et al. "Sonification report: Status of the field and research agenda." ICAD 1997. ICAD, 1997.

[16] Quinn, Marty. ""WALK ON THE SUN" - An Interactive Image Sonification Exibit." AI&Society (Springer), 2011.

[17] Parsehian, Gaetan. "Locusonus symposium #8 Audio Mobility." Locusonus symposium #8 Audio Mobility. Locusonus, 2014.

[18] Polli, Andrea. "Soundscape, Sonification and Sound Activism ." AI& Society - Sonification - What Where How Why (Springer), 2011.

[19] Schafer, R Murray. The Tuning of The World. New York: Alfred Knopf, 1977.

[20] Sinclair, Peter. "Living With Alarms." AI & Society - Sonification How Why What Where (Springer), 2011.

[21] Sinclair, Peter, ed. "Sonification - What Where How Why." AI & Society (Springer) 27, no. 2 (05 2012).

[22] Reeves, Nicolas. “La Harpe à Nuages (Cloud Harp).” Fondation Langlois. http://www.fondation-langlois.org/html/e/page.php?NumPage=101. 1997-2000.
[23] Vesna, Victoria. “Vibration Matters: Collective Blue Morph effect.” AI& Society - Sonification - What Where How Why (Springer), 2011.
[24] Vickers, Paul. “Sonification Abstraite/Sonification Concrete: An 'Aesthetic Perspective Space' For Classifying Auditory Displays in the Ars Musica Domain.” Proceedings of the 12th International Conference on Auditory Display. london, 2006.