LE PROJET Q-LIVE

Pierre Michaud
Université de Montréal
p.michaud@umontreal.ca

Olivier Bélanger
Université de Montréal
olivier.belanger@umontreal.ca

Lucas Paris
Université de Montréal
lucas.paris@umontreal.ca

RÉSUMÉ

Cet article présentera l'état des recherches et du développement de *Q-Live* : un nouveau logiciel pour la musique mixte développé à la Faculté de musique de l'Université de Montréal depuis l'été 2014. Ce projet a comme buts principaux :

1. De proposer une solution au problème d'obsolescence technologique en musique mixte ;
2. De faciliter le développement d'une pédagogie heuristique de la musique mixte pour les interprètes et les compositeurs.

Q-Live est un logiciel *open source* conçu par Pierre Michaud qui sera développé sur trois ans par Olivier Bélanger et Lucas Paris grâce au soutien du Fonds de recherche du Québec sur la société et la culture (FRQSC). Le logiciel sera disponible pour OSX, Windows et Linux.

1. GÉNÈSE DE Q-LIVE : REFLEXIONS PRÉLIMINAIRES

La musique mixte[1] existe depuis trois quarts de siècle[2] et soulève toujours de nombreuses questions. Les possibilités d'interactions entre un interprète et un ordinateur sur scène se sont démultipliées depuis la fin des années 1980 [4], en grande partie grâce aux logiciels d'algorithmie musicale (*Max, PureData, SuperCollider*, etc.). Ces logiciels de programmation graphique ou textuelle font désormais partie des cursus de composition et d'informatique musicale dans de nombreux pays. Nous pensons que ce changement de paradigme vers le numérique a permis un certain décloisonnement de la profession de compositeur en y greffant les métiers de réalisateur en informatique musicale, de soudeur ou de luthier dans le cas du *physical computing*, etc. Les vertus pédagogiques d'un tel enseignement sont inestimables.

Cependant, nous constatons également que ce changement a apporté son lot de nouveaux problèmes. Parmi les plus importants mentionnons : l'obsolescence technologique (plus particulièrement des logiciels), et l'investissement en temps nécessaire à la maîtrise des rudiments de programmation musicale. *Q-Live* compte justement aborder ces deux problématiques comme prémisses de départ pour l'élaboration de la conception logicielle.

Après 25 ans de création avec ces outils de programmation graphique, existe-t-il un savoir faire, des stratégies et des méthodes pour l'élaboration de la composante informatique ? Serait-ce possible d'imaginer un autre paradigme pour l'élaboration et la mise en forme de la composante électronique des œuvres mixtes ? Serait-ce possible de sauvegarder automatiquement les données nécessaires à la reconstitution complète de la composante électronique d'une œuvre mixte ?

1.1. La question de la pérennité et le niveau neutre

Commençons par le commencement. *Q-Live* est un logiciel conçu, d'abord et avant tout, pour la musique mixte avec partition. Le problème de pérennité ne se pose pas tant pour le volet instrumental de cette discipline si nous restons dans le paradigme des partitions imprimées sur papier. En proposant une certaine représentation graphique d'une certaine pratique musicale, Guido D'Arezzo[3] a donné aux compositeurs le pouvoir de transmission d'une intention musicale aux interprètes. Cette écriture, comme toute écriture, est dépendante d'une pédagogie qui lui est adaptée. Cette écriture, comme toute écriture, a inspiré un corpus théorique qui définit son organisation, ses comportements, son intégration, sa transmission. Nous pouvons aussi affirmer, sans l'ombre d'un doute, que la pérennité d'une écriture est directement dépendante de la transmission et de l'utilisation courante de celle-ci.
Dans la tripartition sémiologique de Nattiez [3], la

[1] Pour cet article, nous entendons comme *musique mixte* (au singulier), une œuvre musicale intégrant simultanément une composante électroacoustique (fichiers sonores, traitements en temps réel) et une composante acoustique (interprétation sur scène), le tout accompagné d'une partition. Nous reconnaissons qu'il existe, bien évidemment, d'autres formes de musiques mixtes (art performatif, robotique, musiques visuelles, etc.), mais nous nous intéressons particulièrement à la problématique de pérennité des œuvres mixtes avec partitions.

[2] En considérant *Imaginary Landscape No.1* (1939) de John Cage (1912-1992) comme une des premières œuvres mixtes dans l'histoire de la musique.

[3] Système de notation sur portée inventé par Guido d'Arezzo (992-1050), célèbre pour sa contribution à la transmission et à la préservation d'une intention musicale.

partition musicale est considérée comme étant le niveau neutre, ou l'ensemble des configurations du texte musical. Les deux autres niveaux sont :

1. Le poïétique, ou niveau de la création. Niveau lié à l'expérience du compositeur, à son inspiration, aux processus de création.
2. L'esthésique, ou niveau du ressenti, de la perception.

Force est de constater que la venue des musiques électroacoustiques, le siècle dernier, bouleverse cette théorie, le niveau neutre étant beaucoup plus difficile à définir. Peut-on, à l'écoute seule d'une œuvre acousmatique, par exemple, saisir l'ensemble des configurations du texte musical ?

Or, dans la musique mixte avec partition, le niveau neutre est bel et bien présent, mais est incomplet dans la majorité des cas. Comme le démontre l'exemple plus bas (Figure 1.), la partie interprétée de la composante électronique se résume à appuyer sur la touche *Mi* grave d'un clavier MIDI. Cette touche active une réverbération sur la note chantée par le soprano (*Sol#*) et déclenche un fichier sonore avec un texte enregistré (« warum ich behaupte,… »). Les chiffres en haut de la portée décrivent la spatialisation sonore. Quelle est la durée de la réverbération ? Quel est le type de réverbération utilisé ? La partition ne nous dévoile pas ces détails. La description morphologique sert de repère pour la chanteuse et pour le chef.

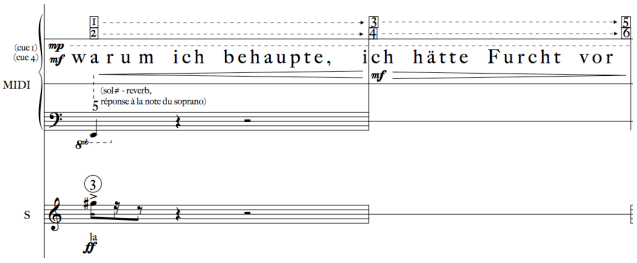

Figure 1. Extrait de *Constat d'une métamorphose (2006/2008)* de Pierre Michaud pour soprano, 13 musiciens et électronique [2].

La partie MIDI écrite dans la partition et la description morphologique pourraient être considérées comme étant suffisamment complètes dans le contexte d'une partition musicale de musique instrumentale ou vocale. La partition sert l'interprète, mais transmet-elle l'intention musicale du compositeur ? Pas complètement, d'où le problème.

Pour l'instant, la seule « partition » permettant de reconstituer une composante électronique dans une œuvre mixte est le *patch* de concert (avec l'aide des indications dans la partition musicale). Les standards actuellement dans la pratique pour l'élaboration de ces *patches* sont les environnements de programmation graphique *Max* ou *PureData* (Figure 2.).

Au niveau poïétique, ces environnements de travail peuvent présenter un avantage, même s'il s'agit d'environnements de relativement bas niveau comparativement aux logiciels de type séquenceur ou d'édition musicale. Le compositeur peut, par exemple, comprendre par une étape de formalisation le fonctionnement de certains traitements ou créer des prototypes à condition d'être formé en informatique musicale et en traitement de signal.

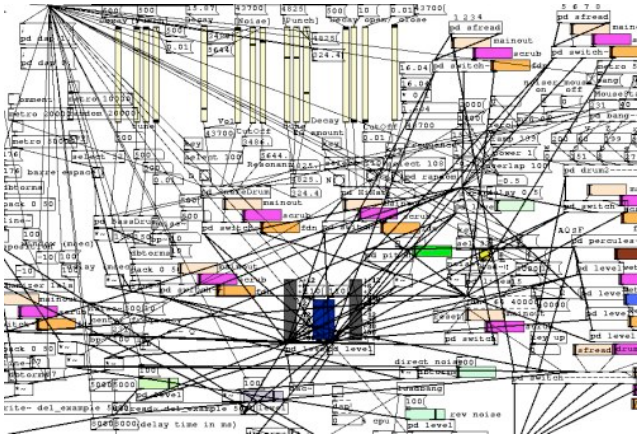

Figure 2. Exemple d'un patch PureData.

Ceci dit, en considérant la personne qui devra, un jour ou l'autre, reconstituer la programmation dans une œuvre mixte, cette approche est loin d'être idéale ou, du moins, semble incomplète du point de vue du niveau neutre.

Est-il possible de reproduire à l'identique le résultat sonore d'un traitement ou d'un processus présent dans une œuvre s'il est programmé dans un autre environnement ? Dans l'éventualité d'un ou plusieurs objets externes (ou internes) *Max* désuets, comment les reconstituer si l'ensemble des configurations du « texte » de programmation est incomplet ou difficilement retraçable ? Il y aura donc une part de traduction de l'intention musicale du compositeur (ou du RIM dans certains cas) en ce qui a trait à la composante électronique d'une œuvre qui doit être migrée.

Considérons donc l'acte de migration technologique[4], comme un acte de traduction.
Cet interprète-informaticien sera, plus souvent que non, un

[4] Migration technologique inévitable en musique mixte considérant la courte durée de vie des versions de logiciels et des systèmes opérationnels.

compositeur ou un Réalisateur en informatique musicale (RIM) selon le milieu et le contexte culturel ou socio-économique.

En ce qui concerne le Québec, et pour la majeure partie de l'Amérique du Nord et de l'Amérique du Sud, le RIM n'a pas le statut que nous retrouvons en France [7]. Nous pouvons donc sous-entendre, compte tenu des formations qui sont actuellement offertes dans nos universités et nos conservatoires, et compte tenu du financement disponible dans le cadre des programmes de commandes d'œuvres financés par les Conseil des arts du Canada ou du Conseil des arts et des lettres du Québec, que cet interprète-informaticien risque d'être un compositeur de formation. À quoi pourrait donc ressembler une « partition » qui lui permettrait de reconstituer la composante électronique obsolète d'une œuvre ? Quelles sont les informations qui devraient s'y trouver ?

2. DESCRIPTION DE Q-LIVE

2.1. *Q-Live* – prémisses de départ

Le logiciel développé dans le cadre de ce projet doit répondre à un certain nombre d'exigences. Il doit être simple d'utilisation, c'est-à-dire offrir une courbe d'apprentissage rapide, tout en étant le plus flexible possible, afin de répondre aux besoins de mise en forme et de standardisation de la notation électronique. Ce logiciel n'est pas destiné à des programmeurs ou à des informaticiens, mais bien aux compositeurs et aux interprètes, dans le but de leur offrir un environnement de travail unifié. Pendant la création de l'œuvre, le compositeur doit avoir la pensée libre de toute composante technique afin de se consacrer pleinement à l'écriture de la musique, instrumentale et électronique. Nous considérons donc que le logiciel doit être une aide et un support au créateur et non un fardeau dont il faut sans cesse maîtriser les rudiments. De même, pour l'interprète, le logiciel se doit d'être suffisamment simple d'intégration pour être utilisé sur scène sans le recours à un technicien.

Plusieurs raisons ont guidé le choix de l'environnement de développement pour l'écriture de *Q-Live*. Premièrement, afin d'assurer une longue vie au logiciel, il est impératif de travailler avec un langage de programmation mature, largement utilisé et en développement actif. Ainsi, malgré l'évolution des technologies, il est raisonnable de penser qu'un tel langage sera supporté encore plusieurs années. Ensuite, dans le but d'offrir ces nouvelles fonctionnalités au plus grand nombre possible, l'environnement doit fonctionner sous tous les systèmes d'exploitation majeurs (Mac OS, Windows, Linux). En éliminant le clivage en fonction de l'ordinateur disponible, nous augmentons la portabilité et la diffusion des œuvres musicales. Finalement, comme la pérennité d'un logiciel est en partie soumise à son accessibilité et à la possibilité d'être amélioré, ou adapté, par des tiers, la licence de déploiement assurera sa gratuité et la disponibilité de son code source. Les composantes utilisées pour le développement de l'application seront donc :

1 – Le langage de programmation Python. Deuxième langage le plus utilisé à ce jour, il est multiplateformes, gratuit et possède une immense bibliothèque de librairies permettant le développement en tout genre.

2 – La librairie *wxPython*. Librairie dédiée à la création d'interface graphique. Elle offre l'avantage que le même code fonctionnera de façon native sous tous les systèmes. L'aspect visuel du logiciel s'adaptera donc aux styles en place sur le système où il sera utilisé.

3 – Le moteur audio *pyo*. Environnement de synthèse et de traitement du son dédié au langage Python, *pyo* est développé ici même à l'Université de Montréal. Le module offre tous les outils nécessaires à la création musicale, du traitement de signal à la composition algorithmique, en passant par la gestion temporelle des évènements.

2.2. Pyo

La création du module Python *pyo*, par Olivier Bélanger, a débuté suite à une réflexion issue de ses expériences de programmeur d'applications audio. Après plusieurs années à développer des logiciels sonores alliant le langage Python pour la gestion de l'interface et *Csound* comme moteur audio [5] parmi lesquels figure *Ounk* [1] l'ancêtre de *pyo*, le besoin d'un environnement de programmation unifié devint manifeste.

L'idée de départ était d'éliminer la séparation existante entre l'engin audio et l'interface de contrôle. Un environnement de développement qui intégrerait à la fois le moteur audio et le langage de programmation permettrait de créer des interactions plus sophistiquées entre les structures de contrôle et les processus sonores. Un langage de programmation générale, tel que Python, offre une panoplie de fonctionnalités, des mathématiques complexes aux librairies d'interface graphique en passant par la programmation de base de données, qui sont généralement absentes dans les langages de programmation dédiés au son. Dans le cadre du développement d'un logiciel autonome, ces fonctionnalités peuvent parfois s'avérer cruciales. Et si, parmi ces spécialités, séparées en modules distincts, il en existait une qui soit dédié au traitement de signal ?

Le module *pyo* permet de créer le pont entre le langage servant à construire la structure du logiciel, c'est-à-dire les

contrôles de paramètres, et la génération des algorithmes sonores. En évoluant à l'intérieur du même environnement de programmation, la communication entre les éléments d'interface, ou les structures génératives, et les processus audio est grandement simplifiée puisque les objets ont un accès direct les uns aux autres. Le moteur audio devient donc un module spécialisé parmi des centaines d'autres modules disponibles, chacun étant optimisé pour effectuer une tâche particulière.

Les composantes audio de *pyo* étant des objets Python standards, la communication avec une interface graphique peut s'effectuer sans intermédiaire car les deux modules existent dans le même espace mémoire d'une application unique. Il n'est plus nécessaire d'utiliser un canal de transmission de données externe, tel que le protocole de communication OSC [6] pour transmettre les données de l'interface au moteur audio. L'exemple suivant (Figure 3) illustre l'utilisation de la librairie *wxPython* pour créer une interface graphique (Figure 4) et communiquer les données de contrôle aux objets audio. Comme le serveur et le synthétiseur audio sont créés à l'intérieur de la même classe que le bouton et le potentiomètre, les fonctions appelées par ces derniers n'ont rien d'autre à faire que de transférer directement les valeurs aux variables *self.server* et *self.synth* pour opérer des changements dynamiques du résultat sonore.

```
import wx
from pyo import *

class MyFrame(wx.Frame):
  def __init__(self, parent, title, pos, size):
    wx.Frame.__init__(self, parent, -1, title, pos, size)

    self.server = Server().boot()
    self.synth = SuperSaw(freq=[330,331], mul=.2).out()

    audio = wx.ToggleButton(self, -1, "on / off", (90,10))
    audio.Bind(wx.EVT_TOGGLEBUTTON, self.handleAudio)
    label = wx.StaticText(self, -1, "Freq", (13,62))
    fr = wx.Slider(self,-1,330,50,600,(50,45),(200,-1),
                   wx.SL_LABELS)
    fr.Bind(wx.EVT_SLIDER, self.changeFreq)

    self.Show()

  def handleAudio(self, evt):
    if evt.GetInt() == 1: self.server.start()
    else: self.server.stop()

  def changeFreq(self, evt):
    self.synth.freq = [evt.GetInt(), evt.GetInt()+1]

app = wx.App(False)
frame = MyFrame(None, 'SuperSaw', (25,25), (260,120))
app.MainLoop()
```

Figure 2. Exemple de code.

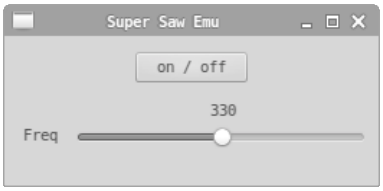

Figure 3. Interface graphique.

Q-Live étant construit à l'aide d'une structure modulaire, toutes les composantes audio du programme seront créées de façon dynamique, lorsqu'un effet est ajouté sur une piste par exemple. Le module de traitement de signal ainsi que l'élément d'interface graphique qui lui est rattaché seront donc créés de pair et automatiquement connectés. L'aspect modulaire du logiciel permettra notamment à l'utilisateur d'ajouter ses propres algorithmes de traitement à l'application en insérant des scripts Python dans la chaîne d'effets sonores d'une piste. De nos jours, le *scripting* est régulièrement utilisé pour modifier le comportement d'un logiciel (personnalisation des raccourcis, automatisation des tâches) mais rarement pour créer des processus audio de façon dynamique. La combinaison Python-*pyo* simplifie grandement le *scripting* audio.

2.3. Q-Live GUI

À partir des prémisses de départ de *Q-Live*, nous avons constitué une architecture logicielle et une interface graphique. Grâce à l'efficacité du langage Python et à l'expertise acquise à la faculté de musique de l'Université de Montréal en création d'interfaces graphiques avec wxPython pour les logiciels *Cecilia5* et *Soundgrain*, le développement d'un prototype fonctionnel a pu se faire en trois mois.

La partie centrale du logiciel est la section d'effets audionumériques. Celle-ci permet de charger des modules d'effets dans un tableau, d'accéder à une fenêtre flottante contenant les paramètres d'un effet et des options d'automations pour ces paramètres (*breakpoint function*, modulation par suivi d'amplitude, contrôle MIDI et OSC), de créer des connexions entre les effets, et de choisir des entrées audio.

Ensuite vient la section de gestion des événements. Cette section *cues* permet la création et la sélection de nouveaux *cues*. Après avoir sélectionné un événement on peut modifier l'ensemble des paramètres de la section d'effet et ils seront automatiquement enregistrés dans ce *cue*. À la sélection d'un autre *cue* une interpolation a lieu entre les anciennes et les nouvelles valeurs des paramètres. La durée de cette interpolation peut être ajustée de manière indépendante pour chaque module d'effet et pour chaque paramètre.

L'interface principale est complétée par une section mixeur. Cette section permet d'ajuster les signaux d'entrée avant qu'ils ne parviennent à la section d'effet. Il sera également possible d'appliquer des traitements dans cette section tels que le filtrage, l'égalisation et la compression qui ne seront pas enregistrés dans les *cues*. Les deux exemples suivants (Figures 5 et 6) démontrent la structure du GUI en phase de développement.

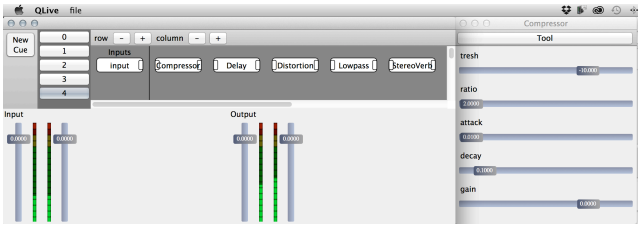

Figure 5. Capture d'écran de *Q-Live* (Décembre 2014).

Figure 6. Capture d'écran de *Q-Live* (Mai 2015).

Finalement, on pourra accéder à une deuxième interface pleinement dédiée à la sauvegarde d'archive. Celle-ci permettra au compositeur de compléter l'archive avec des fichiers qui ne sont pas inclus dans le logiciel, tels que des enregistrements vidéo d'une performance, une partition, ou des commentaires. Cette partie de l'interface sera présentée comme un formulaire et pourra aussi être utilisée indépendamment du reste de *Q-Live*. Même si un utilisateur utilise un autre environnement audionumérique, il peut tout de même profiter de ce formulaire qui générera une archive dans un format standard.

2.4. Le format de sauvegarde automatique de Q-Live

Concernant l'obsolescence technologique, nous avons cru bon d'intégrer une sauvegarde d'archive efficace, complètement indépendante du logiciel. Cette sauvegarde comprendra une description générée automatiquement en format .txt des effets utilisés (type d'effet, algorithmes si nécessaire) et une description des évènements déclenchés dans une œuvre mixte (paramètres, automations, descriptions). Le choix d'utiliser la librairie *pyo* développée par Olivier Bélanger s'est imposé.

Malgré la complexité apparente de la programmation graphique, un *patch* de concert peut se résumer, plus souvent que non, comme étant une organisation d'évènements (traitements, fichiers sonores, automations, processus, etc.) à déclencher physiquement (contrôleur MIDI, clavier d'ordinateur, etc.) ou automatiquement dans le temps (suivi de hauteur, suivi d'amplitude, *timeline*, etc.). Les effets audionumériques sont eux aussi, plus souvent que non, limités aux catégories génériques existantes (réverbérations, filtres, *harmonizers*, granulation, déphasages, délais, etc.). Quels seraient donc les éléments de base nécessaires pour la reconstitution d'une œuvre mixte ?

2.4.1. Légende des effets audionumériques

En premier lieu, il serait importun d'intégrer au document les descriptifs des effets audionumériques utilisés dans l'œuvre de manière automatique. Compte tenu des catégories génériques d'effets existants[5] et de l'information librement accessible pour une reconstitution de ces effets, nous croyons qu'une description textuelle suffirait dans la majorité des cas. La description des effets devrait inclure l'ensemble des paramètres ainsi que leur ambitus. Dans le cas d'effets non conventionnels, un exemple de l'algorithme pourrait être inclus.

2.4.2. Description des cues

La mise en forme invoquée dans la description du GUI nous permet d'envisager quelle forme prendra le document. Une description des évènements comprendrait une liste des réglages des paramètres et des instructions d'automation en format de listes.

2.4.3. Liste des fichiers sonores

Un dossier sera créé avec l'ensemble des fichiers sonores d'une pièce. *Q-Live* assignera automatiquement un préfixe standardisé aux noms des fichiers sonores pour assurer l'ordre des déclenchements.

2.4.4. Captures d'écran

Nous croyons l'interface graphique de *Q-Live* suffisamment claire pour instaurer une fonction de captures d'écrans automatiques de chacun des événements. Cette information visuelle sera complémentaire à la description des évènements et permettra de visualiser rapidement le *data flow*.

2.4.5. Enregistrements d'archive

Il sera possible de sauvegarder une performance à même *Q-Live* et de l'intégrer à la sauvegarde d'archive. Il sera possible d'enregistrer la performance de l'interprète (ou des interprètes) dans un fichier séparé et de l'œuvre, dans

[5] Par exemple : Délais, distorsions, filtres, dynamique, *EQ*, spatialisation, modulations, *pitch*, réverbérations, etc.

son entièreté (performance de l'interprète avec traitements et « bandes »), dans un autre fichier sonore.

2.4.6. Commentaires du compositeur

Finalement, un petit éditeur de texte sera intégré au logiciel pour que le compositeur puisse ajouter des commentaires qui faciliteront la migration de l'électronique de son œuvre.

L'arborescence des dossiers créés par *Q-Live* (Figure 7) et le contenu de ces dossiers pourront bien évidemment être modifiés par le compositeur s'il souhaite ajouter d'autres informations.

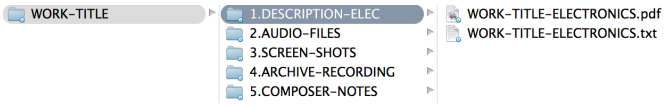

Figure 7. Proposition d'arborescence de dossiers pour la sauvegarde d'archive dans *Q-Live*

3. CONCLUSION

Sans vouloir prétendre solutionner tous les problèmes de la musique mixte, *Q-Live* propose une approche complémentaire à l'algorithmie musicale. Cette approche est particulièrement adaptée à l'écosystème de la Faculté de musique de l'Université de Montréal où nous retrouvons compositeurs, interprètes et musicologues oeuvrant à la fois en création, en recherche-création et en recherche. Le logiciel *Q-Live* sera intégré, par exemple, dans le séminaire de Pierre Michaud intitulé « Composer et interpréter la musique mixte » et dans les cours de composition mixte dès 2015-2016. Le temps nous dira si l'interface graphique et le *workflow* simplifié répondront réellement aux besoins des compositeurs et des interprètes et si l'information compilée dans le format de sauvegarde automatique sera suffisante pour assurer la migration d'œuvres mixtes créées en *Q-Live*.

4. RÉFÉRENCES

[1] Bélanger O. « Ounk - an audio scripting environment for signal processing and music composition. » Dans les actes de la *International computer music conference (ICMC)*, Belfast, Ireland, 2008, p. 399-402.

[2] Michaud, P., *Constat d'une métamorphose*. Centre de musique canadienne, Montréal, 2008.

[3] Molino, J. et Nattiez, J.-J., *Musicologie générale et sémiologie*. Christian Bourgeois Éditeur, Paris, 1987.

[4] Puckette, M. « The Patcher », Dans *ICMC Proceedings* (1988), http://msp.ucsd.edu/Publications/icmc88.pdf

[5] Vercoe B. L., Ellis D. « Real-time csound: Software synthesis with sensing and control. » Dans les actes de la *International computer music conference (ICMC)*, Glasgow, 1990, p. 209-211.

[6] Wright M., Freed A., Momeni A. « Opensound control: State of the art 2003. » Dans les actes de la *Conference on new interfaces for musical expression (NIME-03)*. Montréal, Canada, 2003.

[7] Zattra, L., « Les origines du nom de RIM (réalisateur en informatique musicale)», Dans les *Actes des Journées d'informatique musicale 2013*, http://www.mshparisnord.fr/JIM2013/actes/jim2013_14.pdf