UNE INTERFACE GRAPHIQUE DE MANIPULATION D'UNITES MODULAIRES DANS SUPERCOLLIDER

Frédéric Dufeu
Université Rennes 2
frederic.dufeu@gmail.com

RÉSUMÉ

La modularité caractérise une part importante des lutheries développées à partir des nouvelles technologies. Parmi d'autres langages courants d'informatique musicale, SuperCollider de James McCartney est largement adapté à la création d'unités de génération et de traitement de signal, ainsi qu'à leur assemblage selon une approche modulaire dynamique. Pourtant, lorsque des unités élémentaires sont prédéfinies tandis que leurs interconnexions sont laissées ouvertes, l'établissement d'une configuration particulière implique un important travail de codage, qui rend l'expérimentation peu flexible. Cet article présente un programme permettant, d'une part, de décrire sous forme de code des modules autonomes et, d'autre part, de les interconnecter de manière efficace et pratiquement immédiate à partir d'une interface graphique programmée en OpenGL dans Max/MSP/Jitter. À partir de types d'unités définis par l'utilisateur, cette interface permet de créer un nombre indéfini d'instances et de les interconnecter selon n'importe quelle configuration, sans aucune limitation liée à leurs caractéristiques techniques. Moyennant une préparation des modules et leur intégration au programme général, ce programme favorise l'exploration intuitive des ressources musicales et sonores de SuperCollider.

1. INTRODUCTION

La notion de modularité accompagne dans une large mesure l'émergence et les développements d'outils technologiques destinés à la création et à l'expression musicale. Dans un article présentant les nouvelles lutheries selon une approche historique, Jean-Claude Risset parle de « boîte à outils logicielle » pour désigner les programmes Music n de Max Mathews. À partir de Music III (1959), l'utilisateur

> « est laissé libre de décider du type de synthèse sonore qu'il veut mettre en œuvre : il choisit des modules dont chacun correspond à une production ou une transformation sonore […], et il les assemble à loisir, comme s'il "patchait" un synthétiseur modulaire […]. »[1]

Si la libre interconnexion de générateurs ou traitements autonomes permet au musicien de composer son instrument[2], différents niveaux d'ouverture doivent être distingués selon l'environnement considéré. En premier lieu, un synthétiseur modulaire matériel n'offre qu'un nombre limité de types d'unités (oscillateur, filtre, générateur d'enveloppe, amplificateur) et chacun de ces types est lui-même limité en nombre d'instances[3]. Leur extension par des unités supplémentaires existantes ou par la fabrication de nouveaux types de modules est possible, mais lourde et coûteuse. Avec les logiciels classiques de programmation musicale[4], l'implémentation de générateurs ou traitements inédits n'a de limite que la capacité du développeur à exprimer sa conception musicale ou sonore à travers le langage ou l'environnement choisi et pour un type de module, la quantité d'unités fonctionnant simultanément est contrainte par les performances de l'ordinateur hôte, mais leur création est aisément opérable. Ensuite, et au-delà de la flexibilité relative à la création de modules, les interconnexions de ces derniers ne sont pas toujours entièrement ouvertes : un système donné peut ne se prêter qu'à une partie de l'ensemble des connexions

[1] [15], p. 23. Risset précise immédiatement : « Contrairement à ce que pensent beaucoup, la conception de Mathews ne copie pas celle des synthétiseurs : elle a au contraire inspiré les dispositifs de Moog et Buchla, réalisés analogiquement en tirant parti de la commande par tension… Mais seulement à partir de 1964, alors que Music III a été écrit en 1959. En fait, la conception modulaire de Mathews a marqué la plupart des programmes de synthèse – comme Music 360, Music 11, CMusic, Csound – et des synthétiseurs analogiques ou numériques – comme ARP, DX7, 4A, 4B, 4C, 4X, Syter – aussi bien que des langages de simulation de circuits électroniques, et plus tard un langage de créations d'interactions temps réel comme Max. » *Ibidem*.

[2] Dans un article datant de 1999, Marcelo Wanderley et Philippe Depalle avançaient la notion d'« instrument virtuel ou composé » et écrivaient : « En désolidarisant cause et effet, l'incursion de l'électricité dans la lutherie a profondément modifié la nature du jeu des instruments de musique. Et bien qu'une prodigieuse diversification des moyens de production sonore en ait résulté, ce changement de nature n'a pas encore engendré de stratégies de contrôle permettant de développer la même subtilité et la même ampleur de jeu que celle que l'on trouve pour les instruments traditionnels. » [19], p. 145. Bruno Bossis emploie également l'expression « instrument composé » dans son article « Écriture instrumentale, écriture de l'instrument » : « contrairement à l'instrument traditionnel, l'"instrument composé" n'est pas entièrement défini avant le travail d'écriture par le musicien. Le compositeur prend ainsi en charge une partie du métier de luthier. » [4], p. 120.

[3] Par exemple, l'ARP2600 comprend trois *VCOs* (*Voltage-controlled oscillators* : oscillateurs commandés en tension).

[4] Comme Csound, Max/MSP, PureData ou SuperCollider.

théoriquement possibles. Enfin, l'établissement d'une situation modulaire n'est pas toujours propice à un certain idéal de temps réel. Devoir débrancher et rebrancher un cordon ou une broche empêche le passage instantané d'une configuration à une autre. Par ailleurs, certains logiciels ne permettent pas de modifier le trajet du signal sans interrompre le calcul audionumérique[5].

Livré sous une première version par James McCartney en mars 1996[6], SuperCollider se définit comme « un environnement et un langage de programmation pour la synthèse audio en temps réel et la composition algorithmique »[7]. En plus de la qualité de son rendu sonore, les caractéristiques de ce logiciel libre[8] en font un environnement particulièrement favorable à une approche modulaire dynamique pour la synthèse, la lecture, l'enregistrement et le traitement de signal : comme le décrit son premier auteur,

> « alors même que la synthèse fonctionne, des modules peuvent être créés, détruits et interconnectés différemment, des mémoires d'échantillons être allouées et réaffectées. Des traitements peuvent être créés et connectés dynamiquement au sein d'un trajet de signal existant […]. »[9]

Pour autant, SuperCollider est un langage d'informatique musicale général et la mise en œuvre d'un dispositif inspiré par la modularité des nouvelles lutheries n'en constitue qu'une approche particulière. Si un certain nombre de classes intégrées à la distribution publique et de bibliothèques tierces facilitent la création en cours de jeu de structures musicales ou d'entités audionumériques selon une approche de *live coding*[10], il n'existe à notre connaissance aucun développement permettant à la fois de définir dans SuperCollider une grande variété de types de modules et de les intégrer pleinement dans un environnement ergonomique de création et de manipulation d'interconnexions[11].

L'objectif du projet que nous présentons dans cet article est de favoriser l'exploration des *unit generators*[12] de SuperCollider en permettant à l'utilisateur de les interconnecter librement, et sans explicitement coder les opérations habituellement associées à cette tâche, notamment en termes d'allocation des bus véhiculant les signaux. Étant donné un ensemble de types de modules audionumériques prédéfinis mais extensibles, il s'agit de pouvoir composer aussi simplement que possible des graphes reliant les unités concernées entre elles. L'enjeu est de disposer d'un outil flexible pour l'expérimentation et la composition audionumérique, aussi bien approprié au travail de studio qu'à la performance en temps réel. Actuellement en cours de développement par l'auteur, ce programme est déjà fonctionnel sous forme de prototype et appelle, au-delà de ses améliorations locales, plusieurs perspectives que nous évoquerons en conclusion. Auparavant, notre présentation s'articulera en trois parties : dans un premier temps, seront exposées les fonctions de SuperCollider permettant d'établir sous forme de code des entités audionumériques et leurs interconnexions. Nous décrirons ensuite une interface graphique utilisateur simple implémentée dans Max, par laquelle doivent être opérées les manipulations de ces entités et du graphe les organisant entre elles. Enfin, seront présentés les éléments du programme permettant de lier adéquatement l'interface graphique au serveur de synthèse de SuperCollider.

2. IMPLEMENTATION D'UNITES AUDIONUMERIQUES INTERCONNECTEES DANS SUPERCOLLIDER

2.1. Création d'unités avec les classes SynthDef et Synth

Dans sa version 3[13], SuperCollider repose sur une architecture client-serveur. Le serveur, scsynth, écrit en C++, est le programme dédié à la synthèse et au traitement audionumérique[14]. Il reçoit ses commandes par réseau depuis un ou plusieurs clients, le plus couramment celui constituant l'autre versant de SuperCollider : l'application sclang, l'interpréteur dans lequel le code est écrit et exécuté.

> « Typiquement, l'interpréteur traduit le code, écrit en langage SuperCollider, en messages OSC[15] destinés au serveur. L'utilisateur écrit en quelque sorte de la poésie en SuperCollider, qui est ensuite

[5] Jusqu'à récemment, il n'était pas possible dans Max de créer ou détruire un objet ou un cordon MSP sans interruption du calcul audio ; les modifications de configuration modulaire nécessitaient la mise en œuvre de systèmes matriciels. Cette limitation a été supprimée dans la version 6 de Max, commercialisée en octobre 2011 [11].

[6] Pour une description de la genèse de SuperCollider, voir [12].

[7] « *[An] environment and programming language for real time audio synthesis and algorithmic composition.* » [17] Notre traduction.

[8] Depuis 2001. SuperCollider est actuellement publié sous licence GNU GPL.

[9] « *While synthesis is running, new modules can be created, destroyed, and re-patched, and sample buffers can be created and reallocated. Effects processes can be created and patched into a signal flow dynamically […].* » [13], p. 64. Notre traduction.

[10] Comme la bibliothèque *Just-in-Time Programming* de Julian Rohrhuber et Alberto de Campo [16]. Sur la pratique générale du *live coding*, voir [2].

[11] Le projet approchant le plus cette intention est *BabaKoto* d'Andrea Valle [18]. S'il fournit un ensemble de classes offrant une approche modulaire supportée par une interface graphique efficace, chaque module est équivalent à une instance unique de la classe Synth, ce qui, comme on le verra dans la suite de cet article, est restrictif vis-à-vis de notre objectif tendant à une plus grande généralité.

[12] Héritée des programmes Music n de Max Mathews [10], l'expression *unit generator*, désignant une unité de génération, de lecture, d'enregistrement ou de traitement de signal, est reprise dans plusieurs environnements d'informatique musicale, dont SuperCollider avec la classe UGen.

[13] Le programme décrit dans cet article repose sur la plus récente version stable pour Mac OSX, Linux ou Windows au 29 février 2012 : SuperCollider 3.4.4 (publication en juillet 2011).

[14] Pour une description complète de l'implémentation du serveur audio de SuperCollider, voir [1].

[15] *OpenSoundControl*.

paraphrasée en prose OSC par l'interpréteur sclang, elle-même envoyée au serveur [scsynth]. »[16]

Les *unit generators* de SuperCollider sont représentés au sein de fonctions (*UGen graphs*) passées en arguments d'instances de la classe SynthDef. La figure 1 montre l'implémentation d'un SynthDef décrivant deux sinusoïdes modulées en anneau.

```
SynthDef.new(\ringmod, { arg freq1, freq2;
	var oscil1, oscil2, output;
	oscil1 = SinOsc.ar(freq1);
	oscil2 = SinOsc.ar(freq2);
	output = oscil1 * oscil2;
	Out.ar(0, output);
}).add;
```

Figure 1. SynthDef « ringmod », deux sinusoïdes et un modulateur en anneau

Lorsque le code de la figure 1 est exécuté, le graphe d'*unit generators* du SynthDef est compilé et envoyé au serveur. Il peut alors servir de référence à une ou plusieurs instances de la classe Synth, qui commandent au serveur l'allocation d'un *node* pour le calcul audionumérique et la production sonore correspondante (figure 2).

```
// Création de trois nodes et déclenchement de la
production sonore correspondant au SynthDef
« ringmod »
x = Synth.new(\ringmod, [\freq1, 200, \freq2, 800]);
y = Synth.new(\ringmod, [\freq1, 500, \freq2, 1500]);
z = Synth.new(\ringmod, [\freq1, 100, \freq2, 1000]);

// Destruction des trois nodes
x.free; y.free; z.free;
```

Figure 2. Trois instances de Synth se référant au SynthDef « ringmod »

2.2. Allocation de *buffers*

Certains *unit generators* peuvent faire appel à des mémoires d'échantillons. Celles-ci, représentées par la classe Buffer, sont allouées indépendamment de la déclaration du SynthDef, et sont passées en arguments du Synth correspondant (figure 3).

```
// Allocation de deux échantillons dans deux objets
Buffer
s = Server.default;
a = Buffer.read(s, "échantillon1");
b = Buffer.read(s, "échantillon2");

// Déclaration du SynthDef
SynthDef.new(\lecteur, { arg bufnum;
	var output;
	output = PlayBuf.ar(1, bufnum,
BufRateScale.kr(bufnum));
	Out.ar(0, output);
}).add;

// Création des objets Synth
	// Lecture du premier échantillon
	y = Synth.new(\lecteur, [\bufnum, a]);
	// Lecture du second échantillon
	z = Synth.new(\lecteur, [\bufnum, b]);

// Destruction des objets Synth et Buffer
y.free; z.free; a.free; b.free;
```

Figure 3. Lecteur d'échantillon simple : objets SynthDef, Buffer et Synth

2.3. *Nodes* statiques et dynamiques

Dans les deux exemples précédents, le modulateur en anneau et le lecteur d'échantillon, les *nodes* ont un point commun : ils sont instanciés et libérés par commandes explicites (respectivement les messages « new » et « free » passés aux objets Synth). L'une des caractéristiques de SuperCollider ayant fortement contribué à motiver notre projet est la possibilité de se référer à un SynthDef par la création non d'un objet Synth, mais d'un objet Pbind, qui permet d'associer des motifs (*patterns*) de durées à des motifs de valeurs pour les arguments du SynthDef. Considérons le SynthDef déclaré à la figure 4, représentant une sinusoïde à laquelle est appliquée une enveloppe triangulaire :

```
SynthDef.new(\sinusgrain, { arg freq, amp;
	var envelope, output;
	envelope = Env.new([0, 1, 0], [0.1, 0.1]);
	envelope = EnvGen.kr(envelope, gate: 1,
doneAction: 2);
	output = SinOsc.ar(freq, mul: envelope*amp);
	Out.ar(0, output);
}).add;
```

Figure 4. SynthDef « sinusgrain », sinusoïde et enveloppe triangulaire

L'objet Pbind, à la figure 5, permet de décrire une séquence d'événements (*events*) appelant l'instrument défini par le SynthDef « sinusgrain », et dont les valeurs paramétriques respectent les *patterns* correspondant à ses arguments (fréquence et amplitude). Envoyer le message « play » à cet objet Pbind génère, à partir de grains sinusoïdaux[17], une séquence dont les six premières mesures sont données en représentations musicales solfégiques à la figure 6.

```
p = Pbind(\instrument, \sinusgrain,
	\dur, Pseq([0.5, 0.25, 0.25], inf),
	\freq, Pseq([60, 62, 63, 65].midicps, inf),
	\amp, Pseq([0.5, 0.1, 0.1, 0.5, 0.1], inf)
);

q = p.play;
q.stop;
```

Figure 5. Pbind décrivant une séquence d'événements créant des *nodes* dont la référence est le SynthDef « sinusgrain »

[16] « *Typically, the interpreter translates the code in SuperCollider language in OSC messages for the server. The user writes poetry (so to speak) in the SuperCollider language which is then paraphrased in OSC prose by the sclang interpreter, to be sent to the server.* » [5] Notre traduction.

[17] Grâce à l'argument doneAction de valeur 2 passé à l'objet EnvGen dans le SynthDef « sinusgrain » (figure 4), les *nodes* instanciés par le serveur à chaque nouvel événement de la séquence s'autodétruisent dès que l'enveloppe a terminé sa course, évitant ainsi une accumulation de modules audio et une surcharge du serveur.

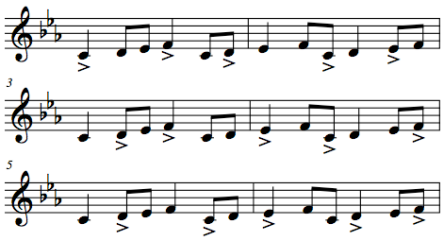

Figure 6. Début de la séquence solfégique correspondant à la description du Pbind de la figure 5

Selon James Harkins, « les *patterns* sont parmi les éléments les plus puissants du langage SuperCollider »[18]. En effet, ils sont associés à un vaste ensemble de classes permettant de composer des combinaisons de séquences complexes, formulées explicitement ou construites algorithmiquement, et de les appliquer à tous les paramètres des unités audionumériques disponibles. Intégrés à l'approche modulaire que nous présentons, ils autorisent également des interconnexions dynamiques : de la même manière que l'exemple représenté aux figures 5 et 6 met en œuvre des séquences affectant les paramètres de durées, de fréquences et d'amplitudes, il est possible d'écrire des *patterns* pour les appliquer aux entrées-sorties d'un module donné, ou aux appels de *buffers* déclarés indépendamment. Dans le cadre de notre programme, nous distinguons deux grandes catégories de *nodes*, tous deux référencés par la classe SynthDef : les *nodes* statiques, qui sont explicitement créés et détruits et sont permanents entre ces deux opérations, et les *nodes* dynamiques, dont l'allocation repose sur le recours à des *patterns* et pour lesquels la structure du SynthDef permet une destruction automatique, par exemple lorsque l'enveloppe d'amplitude atteint son dernier point chronologique (*self-freeing nodes*).

2.4. Bus et interconnexions d'unités

Les classes In et Out permettent à un *node* de recevoir et d'envoyer des signaux en communiquant avec d'autres unités ou avec des interfaces matérielles d'entrée-sortie par l'intermédiaire de bus. La figure 7 montre la création d'un bus par lequel est véhiculé le signal d'une sinusoïde vers le paramètre de fréquence d'une autre sinusoïde.

```
SynthDef.new(\sinus, { arg out, freq;
	var output;
	output = SinOsc.ar(freq);
	Out.ar(out, output);
}).add;

SynthDef.new(\fm, { arg in, freq, modindex;
	var input, output;
	input = In.ar(in, 1); // entrée 1 canal
	output = SinOsc.ar(freq+(input*modindex));
	Out.ar(0, output);
}).add;
```

[18] « *Patterns are one of the most powerful elements of the SuperCollider language* ». [8] Notre traduction.

```
// Allocation d'un bus audio 1 canal
s = Server.default;
a = Bus.audio(s, 1);

// Création et connexion des objets Synth
y = Synth.new(\sinus, [\out, a, \freq, 300]);
z = Synth.after(y, \fm, [\in, a, \freq, 200,
\modindex, 100]);
```

Figure 7. Fréquence de modulation et porteuse connectées par un objet Bus (variable a)

En plus de l'allocation d'un bus, la connexion de plusieurs objets Synth doit être opérée avec une attention particulière à l'ordre d'exécution des *nodes* sur le serveur. Tim Blechmann explique que

> « lorsque des objets Synth sont créés dans scsynth, ils sont ajoutés à une position donnée dans le graphe des *nodes*. SuperCollider dispose de groupes, qui sont des listes concaténées de *nodes*, chaque *node* pouvant être soit un Synth, soit un autre groupe [objet de la classe Group] ; ainsi est établie une structure de données en arbre, avec un groupe pour racine. La position [d'un *node*] peut être spécifiée avec un *node* de référence [dans notre exemple, y] et un argument de relation [ici, "after"], qui peut être "avant" ou "après" le *node* de référence, ou "en tête" ou "à la queue" d'un groupe de référence. »[19]

À partir des quelques notions fondamentales de SuperCollider introduites à travers cette première partie, il est visible que de larges configurations d'unités statiques ou dynamiques, faisant ou non appel à des *buffers*, sont aisées à établir sous forme de code. Pourtant, si la création d'une configuration spécifique, comme dans le cadre d'une composition, se conçoit naturellement par l'écriture explicite de tous les éléments à prendre en considération (*nodes*, bus, *buffers*, *patterns*, ordre d'exécution), il semble fastidieux de disposer d'un certain nombre de modules audionumériques préalablement déclarés et de devoir coder chaque nouvelle interconnexion. La figure 8 montre une situation initiale dans laquelle quatre modules quelconques sont connectés en série entre l'entrée et la sortie physiques de l'ordinateur hôte. Pour insérer un cinquième module dans le trajet de signal, trois opérations doivent être explicitement formulées : la création d'un bus supplémentaire, la création du Synth correspondant au module, et la modification du bus d'entrée pour l'un des modules existants.

```
// Situation initiale
var hardwareinput = 2, hardwareoutput = 0; // Index
semi-arbitraires pour les entrées-sorties physiques
a = Bus.audio(s, 1); b = Bus.audio(s, 1);
c = Bus.audio(s, 1);
```

[19] « *When synths are created on the SuperCollider server, they are added at a certain position in the node graph. SuperCollider has the notion of groups, which are linked lists of nodes, where each node member can be either a synth or another group, effectively modeling a tree data structure with a group as its root. The position can be specified with one reference node and one relational argument, which can be 'before' or 'after' a reference node or at 'head' or 'tail' of a reference group.* » [3], p. 10. Notre traduction.

```
w = Synth.new(\module1, [\in, hardwareinput,
\out, a]);
x = Synth.after(w, \module2, [\in, a, \out, b]);
y = Synth.after(x, \module3, [\in, b, \out, c]);
z = Synth.after(z, \module4,
[\in, c, \out, hardwareoutput]);

// Insertion d'un nouveau module entre x et y
d = Bus.audio(s, 1);
v = Synth.after(x, \module5, [\in, b, \out, d]);
y.set(\in, d);
```

Figure 8. Insertion d'un cinquième module dans une série préalablement établie

Même avec ce contexte et cette manipulation tous deux extrêmement simples, trois opérations doivent être formulées textuellement. Avec une interface graphique objet-cordon comme celles de Max ou PureData, il est envisageable de considérer les opérations suivantes : premièrement, choisir un type de module (« module5 »), deuxièmement, sélectionner le cordon reliant les modules x et y (respectivement « module2 » et « module3 »), troisièmement, choix de l'opération d'insertion. Les créations et affectations de bus sont alors transparentes pour l'utilisateur. Comme on le verra en troisième partie de cet article, de nombreuses manipulations du graphe de modules peuvent appeler un nombre plus important de procédures à exécuter, alourdissant les opérations de codage si une approche textuelle est retenue, mais n'occasionnant qu'un surcoût minime ou inexistant pour l'action à même une interface graphique simple. Nous présentons un premier prototype de celle-ci dans la partie suivante.

3. INTERFACE GRAPHIQUE MODULAIRE EN MAX/MSP/JITTER

Bien que « SuperCollider fournisse différents ensembles de fonctions d'interface graphique utilisateur, ainsi qu'une syntaxe pour écrire de manière transparente du code indépendant de la plateforme »[20], notre choix pour programmer un environnement graphique de manipulation de modules s'est orienté sur Max pour deux raisons. D'une part, la bibliothèque d'objets Jitter comprend des fonctions OpenGL permettant de développer simplement une représentation du graphe de modules selon des cubes reliés par des cordons. D'autre part, les objets Max de la famille *pattr* permettent une gestion efficace des *presets* de paramètres pour les modules individuels.

3.1. Représentation du graphe synoptique de modules en OpenGL

Dans notre prototype actuel, le graphe modulaire est représenté de manière synoptique. À chaque unité modulaire correspond un cube dont la couleur indique le type. L'ordre de disposition des cubes correspond à l'ordre d'exécution des unités dans scsynth, de gauche à droite puis de haut en bas (tableau 1).

0	1	2
3	4	5
6	7	8

Tableau 1. Ordre relatif d'exécution des modules dans scsynth en fonction de leur répartition sur le plan de l'interface graphique (0 : tête de groupe, 8 : queue de groupe).

La figure 9 montre un exemple de graphe reliant six modules de quatre types différents : trois oscillateurs pouvant recevoir un signal modulant en entrée, un lecteur d'échantillon, un mélangeur, un module de transmission vers le convertisseur digital-analogique. La sortie du premier oscillateur est dirigée à la fois vers l'entrée du deuxième oscillateur et vers sa propre entrée.

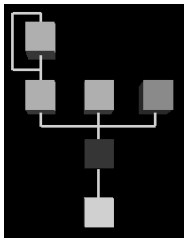

Figure 9. Graphe de six modules : oscillateurs, lecteur d'échantillon, mélangeur, et sortie

Cette représentation est réalisée, à partir de commandes OpenGL envoyées à un objet jit.gl.sketch, sur une fenêtre jit.window : ce dernier objet permet de connaître les coordonnées du curseur de souris et de construire les commandes correspondant aux actions appropriées, comme modifier la couleur des cubes ou des cordons sélectionnés, créer, déplacer ou supprimer un ou plusieurs objets.

3.2. Panneaux des types de modules et des paramètres d'unité

En plus de la fenêtre du graphe général apparaissent toujours au moins trois autres panneaux. L'un permet de sélectionner un type de module (figure 10), le type sélectionné définissant celui de la prochaine unité créée. L'autre est le panneau des paramètres du type correspondant (figure 11). Ainsi, lorsqu'une unité est créée sur le graphe, son type et ses valeurs de paramètres sont déjà déterminés. Le panneau des paramètres étant identique pour plusieurs unités d'un même type, un numéro d'identifiant permet de connaître l'unité à laquelle il est attaché. Enfin, un troisième panneau permet à l'utilisateur de basculer entre deux modes de construction graphique du graphe. Dans le premier mode, chaque nouvelle action sur l'interface commande immédiatement le calcul de modification du graphe audio dans scsynth. Dans le second mode, l'utilisateur opère toutes les modifications graphiques qu'il souhaite sans effet immédiat, avant de déclencher manuellement la mise à jour audio (figure 12).

[20] « *SuperCollider provides for using different gui kits, and also provides syntax for transparently writing kit and platform independent code.* » [7] Notre traduction.

Figure 10. Choix des types de modules et type sélectionné[21]

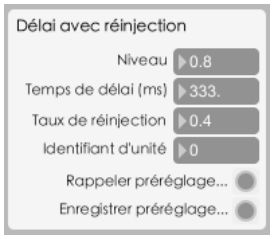

Figure 11. Panneau des paramètres du type « Délai avec réinjection »

Figure 12. Panneau de choix du mode de mise à jour du graphe audio

Dans le premier mode, les modifications sont audibles au fur et à mesure des actions simples (création, déplacement ou suppression d'une unité ou d'un cordon unique). Le second mode permet d'effectuer des transitions directes entre deux configurations modulaires très différenciées (création, déplacement et suppression par lots importants d'unités et de connections).

3.3. Actions de l'utilisateur

Dans l'état actuel du prototype, les actions implémentées se font à partir de la combinaison classique trackpad-clavier ou souris-clavier[22] :
- clic (cube) : sélectionner une unité ;
- clic (cordon) : sélectionner une connexion ;
- clic (vide) : tout désélectionner ;
- cmd+clic (cube ou cordon) : sélectionner une unité ou un cordon sans désactiver les sélections existantes ;
- backspace : supprimer les unités et liaisons sélectionnées ;
- double-clic (espace vide) : créer une unité ;
- clic+glisser-déposer (cube vers espace vide) : déplacer une unité ;
- shift+clic+glisser-déposer (cube vers cube) : connecter deux unités.

Ces actions basiques permettent de modifier le graphe selon les opérations simples implémentées dans SuperCollider : création, déplacement, suppression des unités et des connexions. Les développements futurs de notre interface seront l'objet d'une recherche approfondie sur les possibilités ergonomiques d'actions sur des ensembles d'objets, ainsi que sur des interfaces d'accès gestuel plus appropriées que le binôme clavier-souris.

La représentation du graphe étant synoptique, les cordons n'informent que de l'existence d'une liaison entre une unité et une autre, sans précision sur les index d'entrées-sorties. Lorsque l'action utilisateur consiste à relier un module à un autre et que ceux-ci disposent de plusieurs entrées ou sorties, un panneau permet d'afficher l'index d'entrée ou de sortie visée[23]. À l'aide des flèches gauche et droite du clavier alphanumérique, l'utilisateur peut commander la connexion aux autres entrées-sorties des modules concernés (figure 13).

Figure 13. Panneau de choix des index d'entrées-sorties pour une nouvelle liaison

À terme, l'environnement Max devra permettre un ensemble d'actions spécifiables non graphiquement. Actuellement, une seule fonction existe : générer un graphe d'unités et d'interconnexions aléatoirement[24] (figure 14).

3.4. Sauvegarde et rappel des configurations modulaires

Afin de ne pas devoir reconstruire manuellement un graphe donné depuis une fenêtre vide, un dispositif reposant sur des objets *coll* permet de sauvegarder et de rappeler des configurations modulaires sous forme de fichiers texte[25]. Par ailleurs, pour chaque type d'unité, le panneau des paramètres comprend des boutons de sauvegarde et de rappel de préréglages, qui ouvrent une boîte de dialogue donnant accès aux fonctions avancées d'objets *pattrstorage*[26].

[21] Les modules présentés ici sont donnés à titre de démonstration. Nous exposons en troisième partie les bases de données permettant d'étendre les types de modules disponibles.

[22] Les commandes listées ici sont appliquées à Mac OSX, mais sont généralisables aux autres systèmes d'exploitation.

[23] Par défaut, il s'agit de l'index 0 (première entrée ou première sortie).

[24] À l'origine, cette fonction a été implémentée à des fins de tests. Toutefois, elle nous paraît pouvoir être retenue pour l'expérimentation musicale, et appelle une réflexion sur la mise en œuvre d'autres opérations à caractère aléatoire ou algorithmique.

[25] Au format .txt.

[26] Les informations de préréglages sont importées et exportées au format .xml.

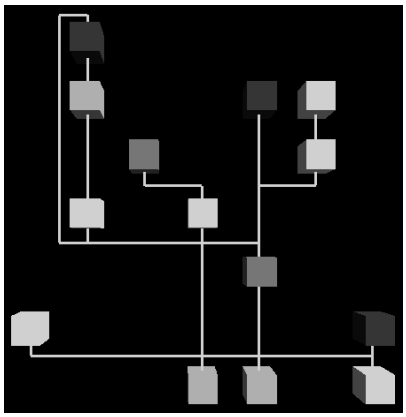

Figure 14. Un graphe généré aléatoirement, sans action utilisateur à même l'interface graphique

Lorsqu'une modification est opérée dans l'environnement Max, aussi bien pour la modification du graphe modulaire que pour l'ajustement paramétrique des unités, les informations adéquates sont transmises à SuperCollider par le biais d'objets udpsend (dans Max) et OSCresponder (dans SuperCollider). Dans la quatrième partie de cet article, nous présentons les opérations par lesquelles SuperCollider adapte son graphe audio en fonction des commandes reçues depuis l'interface graphique.

4. MISE EN CORRESPONDANCE DE L'INTERFACE GRAPHIQUE ET DE SCSYNTH

Dans un article présentant un environnement de simulation d'un synthétiseur modulaire implémenté dans Csound, Eric Lyon décrit son projet en ces termes :

> « La méthode de base de la synthèse analogique consiste à interconnecter des modules autonomes qui génèrent ou modifient des signaux électroniques, pour établir des configurations ayant des qualités sonores particulières. Ce paradigme est adopté dans Csound, où les modules sonores sont remplacés par des *unit generators*, et les cordons de raccordement par des variables de signal. Ce projet a deux parties. D'abord, nous écrirons un langage très simple qui traduit les spécifications de *patch* en orchestres Csound. Ensuite, nous écrirons un programme qui génère algorithmiquement les spécifications de *patch*, automatisant le processus de design sonore. »[27]

En nous inspirant de cette démarche, nous avons écrit dans SuperCollider un programme s'articulant en deux grandes parties. La première contient les bases de données renseignant sur les caractéristiques des unités à intégrer dans notre environnement modulaire. La seconde consiste en un algorithme devant, en fonction des informations reçues depuis Max, modifier le graphe audio généré par le serveur scsynth de SuperCollider.

4.1. Bases de données permanentes dans SuperCollider

4.1.1. Déclaration des objets SynthDef

La programme écrit dans SuperCollider commence par la déclaration des objets SynthDef et leur envoi au serveur, sous la même forme que dans les exemples décrits dans la première partie de cet article. Les instances de SynthDef sont réparties en trois grandes catégories. La première catégorie est celle des objets SynthDef principaux : il s'agit des ensembles d'*unit generators* que l'utilisateur souhaite employer directement à des fins musicales (générateurs, traitements, échantillonneurs). La deuxième catégorie est celle des objets SynthDef de connexion. Comme il le sera précisé au paragraphe 4.2, les connexions entre unités ne peuvent pas toujours être opérées directement : par exemple, envoyer un signal depuis une sortie mono vers une entrée stéréo suppose une conversion, effectuée par un objet Synth. Enfin, la troisième catégorie est celle des objets SynthDef de *bypass*[28]. Pour un module donné, il importe en effet de décrire explicitement le trajet du signal reliant les entrées et les sorties lorsque le traitement est contourné[29]. Pour celles des unités susceptibles d'être désactivées sans être détruites, un SynthDef de *bypass* doit donc être déclaré.

4.1.2. Données relatives aux objets SynthDef

La déclaration des objets SynthDef est suivie de la déclaration d'informations leur étant relatives et nécessaires au fonctionnement de l'ensemble du programme. Pour les SynthDef principaux, ces informations sont les suivantes : nom du SynthDef, statut statique (*false*) ou dynamique (*true*), nombre de *buffers* devant être référencés par le Synth, caractéristiques des entrées-sorties (taux audio ou contrôle[30] et nombre de canaux), nom des arguments (figure 15).

[27] « *The basic method of analog synthesis is to patch together self-contained modules that generate or modify electronic signals into configurations with particular sound qualities. This paradigm is adopted in Csound where sound modules are replaced with unit generators, and patch cords are replaced by signal variables. This project has two parts. First we will write a very simple language that translates patch specifications into Csound orchestras. Second, we will write a program that algorithmically generates patch specifications, automating the sound design process.* » [9], p. 629. Notre traduction.

[28] Nous employons le terme anglais, plutôt que sa traduction « contournement », pratiquement jamais rencontrée.

[29] Par exemple, pour un traitement disposant de deux entrées et de quatre sorties, il peut être souhaitable, en mode *bypass*, de router les deux entrées vers les deux premières sorties, les deux autres sorties ne délivrant alors aucun signal.

[30] Comme le précise Joshua Parmenter, « SuperCollider calcule la sortie [d'*unit generators*] par blocs de valeurs (les échantillons) aussi bien pour les signaux à taux audio (*audio rate : ar*) que pour les signaux à taux contrôle (*control rate : kr*) […]. La taille de bloc par défaut est 64 échantillons, avec un UGen à taux audio calculant 64 échantillons et un UGen à taux contrôle retournant une seul valeur pour chaque bloc. » (« *SuperCollider computes this output in blocks of values (called samples) for both audio rates (ar) and control rate (kr) signals […]. The default block size is 64 samples, with an ar UGen computing 64 samples and a kr UGen returning a single value for each block.* » [14], p. 56. Notre traduction.

```
mainSynthDefData = [
[ \monofm, false, 0,
      [ [[\audio, 1]], [[\audio, 1]] ],
      [\in0, \out0, \freq, \modindex, \amp] ],
[ \stereofm, false, 0,
      [ [[\audio, 2]], [[\audio, 2]] ],
      [\in0, \out0, \freq, \modindex, \amp] ],
[ \eventmonofm, true, 0,
      [ [[\audio, 1]], [[\audio, 1]] ],
      [\in0, \out0, \freq, \modindex, \amp] ],
[ \controlmonofm, false, 0,
      [ [[\control, 1]], [[\control, 1]] ],
      [\in0, \out0, \freq, \modindex, \amp] ],
[ \sampleplayer, true, 1,
      [ [[\audio, 1]], [[\audio, 1]] ],
      [\in0, \out0, \bufnum, \loop, \amp] ]
];
```

Figure 15. Exemple de base de données mainSynthDefData

Les bases de données relatives aux objets SynthDef de connexion et aux objets SynthDef de *bypass* ont pour information le nom de SynthDef et le nom des arguments.

4.1.3. Données relatives aux types d'unités modulaires

À chaque cube de notre interface graphique correspond ce que nous appelons ici une « unité ». Pour une flexibilité optimale, aucun *a priori* n'est donné à la structure d'un type d'unité : elle peut aussi bien correspondre à un SynthDef simple qu'à un ensemble d'objets SynthDef statiques ou dynamiques. En fonction de ses intentions, l'utilisateur peut définir un type d'unité comme étant un oscillateur simple ou une table de mixage virtuelle comprenant des objets SynthDef d'équalisation paramétrique, de compression et d'autres traitements. Avant de déclarer un type donné, il importe de savoir que sa structure interne n'est pas modulable (y compris en termes de nombres de canaux pour les entrées-sorties). Par exemple, si un simulateur de DX7 doit être établi, il conviendra, afin de pouvoir configurer les modulantes et les porteuses selon les différents algorithmes de l'instrument original, de déclarer non pas un type « DX7 », mais un type « oscillateur avec entrée pour modulation », dont on instanciera six unités sur le graphe, selon l'une des trente-deux interconnexions définies par Yamaha. Lorsqu'un type d'unité est défini, l'utilisateur peut, depuis Max, en créer autant d'instances qu'il le souhaite[31]. Les données définissant un type d'unité sont structurées sous la forme d'un *array* d'informations : le graphe des objets SynthDef principaux, définissant les *nodes* à relier et leurs connexions internes à l'unité, les sorties vers l'extérieur de l'unité, le nom du SynthDef de *bypass* à utiliser, l'ensemble des *buffers* utilisés par l'unité, l'ensemble des *patterns* utilisés par l'unité. Dans l'exemple de la figure 16, le premier type est composé d'un seul *node*, ayant pour référence le SynthDef « monofm », dont la base de données mainSynthDefData, à la figure 15, nous informe qu'il est statique (deuxième élément : *false*).

```
userTypeData = [
[ // Premier type
      [ // Graphe des objets SynthDef principaux et
de leurs entrées
            [ \monofm, [[[-1, 0]]] ] ],
      [ [ 0, 0 ] ],// sorties
      \bypass1,     // bypasser
      nil,          // buffers
      nil           // patterns ],
[ // Deuxième type
      [ // Graphe des objets SynthDef principaux
            [ \sampleplayer, [[[-1, 0]]] ] ],
      [ [ 0, 0 ] ],// sorties
      nil,          // bypasser
      buffers[0],   // buffers
      patterns[0]   // patterns ],
[ // Troisième type
      [ // Graphe des objets SynthDef principaux
            [ \monofm, [[[-1, 0]]] ],
            [ \monofm, [[[-1, 1]]] ],
            [ \eventmonofm, [[[0, 0], [1, 0]]] ] ],
      [ [ 2, 0, 0 ], [ 2, 0, 1 ] ], // sorties
      \bypass3,                    // bypasser
      nil,                         // buffers
      patterns[1]                  // patterns ]];
```

Figure 16. Base de données de trois types d'unités

L'*array* [[[-1, 0]]] lui étant associé signifie que sa première entrée correspond à la première entrée de l'unité elle-même. L'*array* suivant (« sorties ») indique que la première sortie de l'unité correspond à la première sortie du premier Synth du graph ([0, 0]). Ensuite, sont indiqués le nom du SynthDef de *bypass* pour l'unité (« bypass1 »), et les deux derniers éléments de l'*array* d'unité indiquent que celle-ci n'utilise aucun *buffer* ni aucun *pattern* (nil, nil).

Le troisième type est composé de deux objets Synth statiques (« monofm ») et d'un *node* dynamique (« eventmonofm »). Les *arrays* d'entrées et de sorties indiquent les connexions suivantes : la première entrée du premier « monosynth » correspond à la première entrée de l'unité ([[[-1, 0]]]), la première entrée du second « monosynth » correspond à la seconde entrée de l'unité ([[[-1, 1]]]). Le SynthDef « eventmonofm » disposant d'une seule entrée, mais étant dynamique, cette entrée sera, en fonction de la séquence du *pattern* utilisé, liée à la première sortie du premier « monosynth » ([0, 0]) ou à la première sortie du second « monosynth » ([1, 0]). De la même manière, en fonction de la même séquence, sa première et unique sortie sera envoyée à la première sortie de l'unité ([2, 0, 0]) ou à la seconde sortie de l'unité ([2, 0, 1]). La figure 17 récapitule la structure d'une unité de type 3.

À partir des bases de données relatives aux objets SynthDef (mainSynthDefData) et aux types d'unités définies par l'utilisateur (userTypeData), est développée une autre base de données (typeData), qui collecte toutes les informations nécessaires à la bonne gestion du graphe audio lors des manipulations graphiques.

[31] Dans la limite des ressources de l'ordinateur hôte.

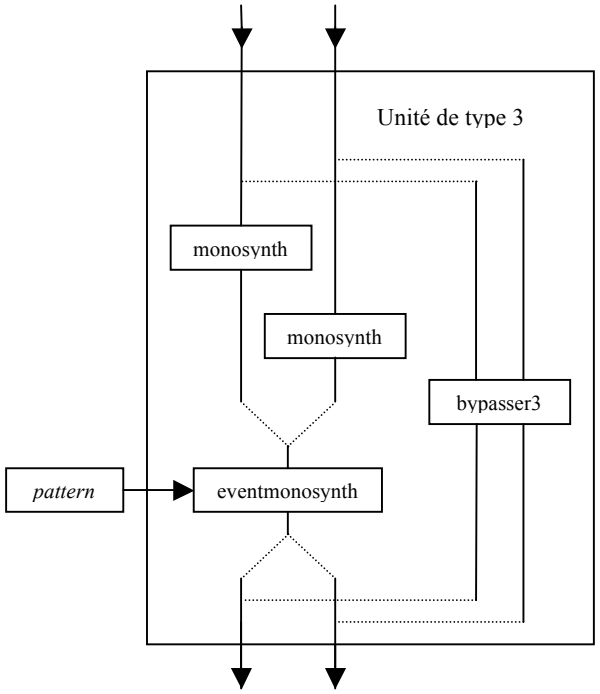

Figure 17. Structure d'une unité de type 3.

4.2. Algorithme de construction et de gestion du graphe audio

Les informations liées à chaque unité créée (objets Group, Synth, Bus, Buffer, *patterns* et informations diverses) sont regroupées dans une base de données dynamique, qui permet d'identifier chaque unité selon son index d'emplacement, inchangé depuis sa création jusqu'à sa destruction. Lorsque des actions sont opérées par l'utilisateur sur l'interface graphique, Max envoie à SuperCollider un message formaté selon l'*array* représenté à la figure 18. À partir de ces informations relatives aux suppressions, créations et déplacements d'unités et de connexions, l'algorithme que nous avons implémenté calcule l'ensemble des opérations de reconfiguration en les exécutant dans l'ordre suivant :
- création des groupes et des structures de données pour les nouvelles unités ;
- création éventuelle des *buffers* et *patterns* associés aux nouvelles unités ;
- création des nouveaux bus et allocation de leurs index à une structure de gestion des bus ;
- création des objets Synth (référençant les objets SynthDef de connexion, de *bypass* et principaux) ;
- modification des positions d'exécution et des indices de bus pour les *nodes* appropriés ;
- suppression des groupes, objets Synth, bus, *buffers* et *patterns* devenus inutiles.

À partir des informations de la base de données « typeData » et des informations de modification du graphe, les calculs relatifs à la gestion des objets Synth et des bus opèrent selon la logique suivante. Soient S la sortie d'un module A et E l'entrée d'un autre module B ou du même module, et l'établissement d'une nouvelle connexion de S à E.

```
modifGraphArray = [
[ // Opérations sur les unités
      [ 1 ], // Suppression de l'unité 1
      [[ \tail, [2, 0]]], // Création d'unités de
type 2 et 0 en queue de graphe
      [[ 0, [4, 5]]] // Déplacement des unités 4 et 5
avant l'unité 0 ],
[ // Opérations sur les connexions
      [ ], // Pas de suppression explicite de
connexion
      [[ [ 4, 0 ], [[ 0, 0 ]] ]], // Création d'une
connexion entre la première sortie de l'unité 4 et la
première entrée de l'unité 0
      [ ] // Pas de déplacement de connexion ]];
```

Figure 18. Informations communiquées depuis Max pour la modification du graphe audio de scsynth

Si S délivre un signal au taux audio et si B précède A dans l'ordre d'exécution du graphe audio, la connexion implique la création d'un module de *feedbacker* entre les deux unités[32]. Si S délivre un signal au taux audio et E doit recevoir un signal au taux contrôle ou inversement, la connexion implique la création d'un module de conversion de taux de signal. Si S délivre un signal circulant sur un nombre de canaux différent de celui pouvant être reçu par E, la connexion implique la création d'un module de conversion du nombre de canaux (par mélange ou par copie). Enfin, si plusieurs sorties sont dirigées vers une même entrée, la création d'un module de mixeur est effectuée. Grâce à l'emploi de ces objets Synth, référencés par les objets SynthDef de connexion, l'utilisateur peut, de manière entièrement transparente, décider de n'importe quelle interconnexion, sans limitation imposée par les caractéristiques de connectique des modules.

5. CONCLUSION

Sous sa forme actuelle, le programme dont nous avons présenté les principaux aspects permet une manipulation simple, flexible et intuitive des *unit generators* de SuperCollider et de leurs possibilités de contrôle dynamique. Une fois la détermination d'unités articulant des objets SynthDef effectuée, l'utilisateur peut en explorer les possibilités musicales et sonores en les interconnectant selon n'importe quelle configuration, d'après le modèle des instruments modulaires rencontrés parmi les lutheries analogiques et numériques. La gestion des différents éléments inhérents à l'implémentation d'un graphe donné (objets Group, Synth, Bus notamment) étant rendue transparente, un important travail de codage, redondant pour qui souhaite utiliser des modules identiques selon différentes combinaisons, est ainsi évité.

L'intérêt de ce programme est potentiellement important pour une large variété d'utilisateurs : utilisable dans le temps de la composition aussi bien qu'en tant qu'instrument de performance temps réel, il constitue un outil efficace et facilement extensible. Pour

[32] Avec la classe InFeedback.

le programmeur averti comme pour le musicien désireux de découvrir les ressources de SuperCollider, l'intégration de types de modules est facilement accessible et peu coûteuse en temps de travail : la déclaration de nouveaux SynthDef et la mise à jour des bases de données de SynthDef et de types d'unités permet la manipulation immédiate de ces derniers au sein de l'environnement existant.

D'ores et déjà exploitable et fonctionnel, le programme comporte de nombreuses limites qui constituent autant de perspectives pour ses développements futurs. En premier lieu, une gestion efficace des *buffers* et des *patterns* devra être implémentée dans l'interface Max. La création d'une mémoire ou d'un motif est encore attachée de manière trop rigide à la création de l'unité correspondante[33] : dans la mesure où ces éléments sont dissociés dans SuperCollider, il convient de permettre à l'utilisateur de gérer sa propre banque d'échantillons et de séquences indépendamment des modules manipulés. Ensuite, la déclaration des types d'unités peut être encore facilitée, par l'implémentation d'un dispositif d'accès graphique pour la création d'un nouveau module. En ce sens, les travaux sur *InkSplorer* présentés par Jérémie Garcia, Theophanis Tsandilas, Wendy E. Mackay et Carlos Agon aux Journées d'Informatique Musicale de 2011 [6] sont d'un grand intérêt pour définir des modules par représentation directement sur papier des unités, entrées-sorties et connexions internes. Enfin, si l'interface synoptique du graphe implémentée dans Max permet une bonne visualisation et favorise des actions simples à partir du binôme trackpad-souris, ses développements devront prendre en compte les possibles interfaçages pour l'utilisateur. Parmi les perspectives que soulève notre programme, son optimisation ergonomique appelle une recherche véritable sur les accès gestuels appropriés à une création musicale reposant sur la modularité virtuelle.

6. REFERENCES

[1] Bencina, R. « Inside scsynth », in Wilson, S., Cottle, D. et Collins, N. (éd.), *The SuperCollider Book*, The MIT Press, Cambridge (MA), 2011, p. 721-740.

[2] Blackwell, A. et Collins, N. « The Programming Language as a Musical Instrument », *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group*, Brighton (UK), Univ. of Sussex, 28 juin-1er juillet 2005, p. 120-130.

[3] Blechmann, T. *Supernova – A Multiprocessor Aware Real-Time Audio Synthesis Engine For SuperCollider*, mémoire de Master en informatique musicale, dir. Ertl, A., Vienna Univ. of Technology, Vienne, 2011.

[4] Bossis, B. « Écriture instrumentale, écriture de l'instrument », in Stévance, S. (dir.), *Composer au XXIe siècle. Pratiques, philosophies, langages et analyses*, Vrin, coll. Musicologies, Paris, p. 119-135.

[5] *Client versus Server Architecture and Operations*, document d'aide intégré à SuperCollider 3.3.4

[6] Garcia, J. Tsandilas, T., Mackay, W. E. et Agon, C. « *InkSplorer* : vers le papier interactif pour la composition musicale », Actes des Journées d'Informatique Musicale 2011, Saint-Étienne, Univ. Jean Monnet, 25-27 mai 2011, p. 237-238.

[7] *GUI Overview and Introduction*, document d'aide intégré à SuperCollider 3.3.4

[8] Harkins, J. « A Practical Guide to Patterns », document d'aide intégré à SuperCollider 3.3.4

[9] Lyon, E. « A Modular Synthesizer Simulation Program », in Boulanger, R. et Lazzarini, V. (éd.), *The Audio Programming Book*, The MIT Press, Cambridge (MA), 2011, p. 629-653.

[10] Mathews, M. *The Technology of Computer Music*, The MIT Press, Cambridge (MA), 1974.

[11] *Max / Cycling'74*, http://cycling74.com/products/max/ (lien vérifié le 29 février 2012).

[12] McCartney, J. « Foreword », in Wilson, S., Cottle, D. et Collins, N. (éd.), *The SuperCollider Book*, The MIT Press, Cambridge (MA), 2011, p. IX-XI.

[13] McCartney, J. « Rethinking the Computer Music Language: SuperCollider », *Computer Music Journal*, vol. 26, n° 4, hiver 2002, p. 61-68.

[14] Parmenter, J. « The Unit Generator », in Wilson, S., Cottle, D. et Collins, N. (éd.), *The SuperCollider Book*, The MIT Press, Cambridge (MA), 2011, p. 55-80.

[15] Risset, J.-C. « Évolution des outils de création sonore », in Vinet, H. et Delalande, F. (dir.), *Interfaces homme-machine et création musicale*, Hermes Science Publications, Paris, 1999, p. 17-36.

[16] Rohrhuber, J. et De Campo, A. « Just-in-Time Programming », in Wilson, S., Cottle, D. et Collins, N. (éd.), *The SuperCollider Book*, The MIT Press, Cambridge (MA), 2011, p. 207-236.

[17] *SuperCollider / About*, http://supercollider.sourceforge.net/ (lien vérifié le 29 février 2012).

[18] Valle, A. « BabaKoto », http://www.fonurgia.unito.it/andrea/wikka.php?wakka=BabaKoto (lien vérifié le 29 février 2012).

[19] Wanderley, M. et Depalle, P. « Contrôle gestuel de la synthèse sonore », in Vinet, H. et Delalande, F. (dir.), *Interfaces homme-machine et création musicale*, Hermes Science Publications, Paris, 1999, p. 145-163.

[33] Par exemple, la création d'une unité de type « échantillonneur » entraîne automatiquement l'allocation d'un nombre fixe d'objets Buffer. La création d'une unité de granulateur d'un type donné entraîne invariablement la création d'un même *pattern*.