Un logiciel de composition musicale combinant un modèle spectral, des structures hiérarchiques et des contraintes

Anthony Beurivé*

Résumé

Cet article décrit un logiciel de composition musicale appelé BOXES, qui fonctionne à la manière d'un séquenceur audionumérique. Avec BOXES, l'utilisateur manipule des sons représentés dans un modèle spectral. Il les place les uns par rapport aux autres dans le temps. Il peut les organiser hiérarchiquement, c'est-à-dire les regrouper dans des structures de plus haut niveau, telles que des mélodies ou des parties de pièces de musique. Il peut également déclarer des contraintes sur les sons et les structures hiérarchiques, pour exprimer des relations temporelles, des relations de hauteur ou des relations de volume. Les contraintes sont ensuite utilisées de manière interactive lors d'une modification de la pièce, afin de mettre automatiquement à jour les éléments qui ne les respectent plus.

Mots clés : logiciel, composition musicale, structures hiérarchiques, contraintes, modèle spectral.

1 Introduction

Cet article décrit de manière informelle le logiciel de composition musicale BOXES, qui s'inscrit dans les travaux du SCRIME (Studio de Création et de Recherche en Informatique et Musique Électroacoustique) sur l'aide à la composition musicale. En l'état actuel, BOXES s'apparente à un séquenceur audionumérique, bien qu'il n'en possède pas toutes les caractéristiques. Peut-être que « logiciel de montage audionumérique » conviendrait mieux.

BOXES fonctionne sous Linux. Il est développé en STk, un interprète du langage de programmation Scheme qui intègre des primitives graphiques et un système de programmation orientée objet à la CLOS [8]. STk permet également le chargement dynamique de modules écrits dans d'autres langages. Ainsi, les parties critiques de BOXES ont été développées en C et en C++. BOXES est un logiciel libre distribué sous license GPL [7].

La raison d'être de BOXES est de valider par la pratique trois notions quelque peu originales dans le contexte des séquenceurs audionumériques :

1. l'utilisation de sons exclusivement issus d'un *modèle spectral* ;
2. la possibilité de structurer *hiérarchiquement* les éléments composés ;
3. l'utilisation de *contraintes* sur les éléments composés.

L'originalité du logiciel tient dans une combinaison particulière de ces trois notions.

La section 2 présente les liens entre BOXES et quelques travaux de la communauté. La section 3 fait un bref survol de l'interface graphique du logiciel. La section 4 énumère les opérations possibles sur les sons. Les structures hiérarchiques sont présentées à la section 5.

*SCRIME, LaBRI, Université Bordeaux 1, beurive@labri.u-bordeaux.fr

Enfin, la section 6 développe les contraintes considérées dans le logiciel ainsi que les mécanismes correspondant.

2 Travaux relatifs

BOXES est proche du concept de *maquette* dans OpenMusic [2]. Il est moins général, dans le sens où il ne permet de manipuler que des sons d'origine exclusivement spectrale, et dont la durée est parfaitement connue. De plus, il ne permet de spécifier que des contraintes linéaires entre les éléments composés, contrairement à OpenMusic qui permet d'avoir des relations fonctionnelles et d'autres types de contraintes. Cependant, la combinaison de structures hiérarchiques et de contraintes est traitée dans BOXES de manière très spécifique, très interactive, ce qui apporte un point de vue original à ce type de représentation.

BOXES s'inspire du logiciel MusicSpace [11] quant à l'utilisation de contraintes. L'objectif de MusicSpace est la spatialisation de sources sonores ; il n'y a donc pas exactement les mêmes types de contraintes dans les deux logiciels. Cependant, du point de vue de l'utilisateur, l'aspect interactif des contraintes est le même. Les outils employés pour satisfaire les contraintes sont pourtant assez différents. MusicSpace utilise un algorithme de propagation qui lui est propre et bien adapté, et qui peut être étendu par de nouveaux types de contraintes. BOXES utilise quant à lui une bibliothèque d'usage général appelée Cassowary, qui ne permet d'employer que des contraintes linéaires, mais avec la possibilité de préciser des « préférences » sur les contraintes [3, 5]. Cette particularité a permis d'intégrer les contraintes aux structures hiérarchiques dans BOXES.

BOXES est une première approche vers des structures fonctionnelles telles qu'on peut les trouver dans OpenMusic, Elody [10] ou BOOMS [4]. Mais il n'apporte pas encore les mécanismes d'abstraction et d'application, fondements de ces travaux. Il reste en effet à déterminer comment les contraintes interfèrent avec ces mécanismes.

Le choix du modèle spectral dans BOXES a été motivé par le rapprochement des travaux au sein du SCRIME. Il est prévu que BOXES s'appuie à court terme sur le modèle SAS, en cours de développement [6]. SAS n'est cependant pas le seul modèle spectral pouvant offrir les opérations utilisées dans BOXES. Le modèle SMS [12], par exemple, conviendrait également. SAS dispose de paramètres sonores plus synthétiques et une abstraction que SMS n'a pas, ce qui en fait un outil un peu plus « musical ». Inversement, SMS considère un ensemble de sons plus grand que SAS en autorisant notamment la présence de transitoires, qui échappent à l'analyse dans SAS.

3 Interface graphique

BOXES tire son nom de la représentation graphique qu'il donne aux sons et aux structures hiérarchiques temporelles, représentation somme toute assez traditionnelle. La figure 1 montre un exemple de fenêtre d'édition. Graphiquement, le temps évolue en abscisse de gauche à droite, linéairement. L'axe des ordonnées n'a pas de signification particulière, et il ne présente pas de pistes.

Dans BOXES, un son ou une structure temporelle possède une date de début et une date de fin parfaitement connues. Le logiciel lui attribue de plus une hauteur graphique arbitraire, qui n'a rien à voir avec son contenu fréquentiel. Il a alors l'apparence d'un rectangle (ou *boîte*). Il y a des boîtes associées à des sons, des boîtes vides correspondant à des silences, et des boîtes

contenant d'autres boîtes. Chaque boîte son affiche des informations spécifiques au son qu'elle contient :

- une figure pleine pour l'évolution de l'amplitude, sur une échelle de 0 à 1 par rapport à la hauteur graphique de la boîte, correspondant à un intervalle de $-\infty$dB à 0dB ;
- une courbe plus sombre pour l'évolution de la fréquence fondamentale, sur une échelle de hauteurs MIDI de 0 à 127 (toujours par rapport à la hauteur graphique de la boîte).

Pour composer de la musique, l'utilisateur a la possibilité de charger un son et de le placer sous forme de boîte dans la fenêtre d'édition. Il peut déplacer les boîtes, changer leur taille (pour changer la durée d'un son par exemple), les détruire, etc. Il peut aussi créer des boîtes hiérarchiques (celles qui contiennent d'autres boîtes) simplement avec la souris, comme on ferait des « sélections ».

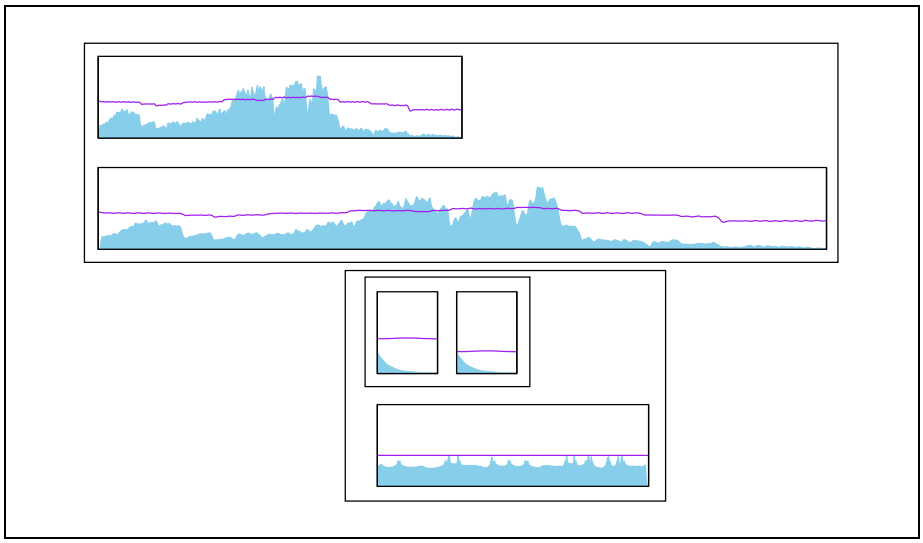


FIG. 1: Le contenu d'une fenêtre d'édition dans BOXES.

4 Modèle spectral

BOXES emploie des sons définis comme des sommes de partiels. Les partiels sont des oscillateurs sinusoïdaux dont l'amplitude et la fréquence varient dans le temps. Cette représentation correspond à un modèle spectral des sons, par opposition au modèle le plus courant, dit *modèle temporel*, basé sur des échantillons de pression de l'air et de voltage. Le fait qu'il existe un modèle spectral sous-jacent autorise des opérations intéressantes d'un point de vue musical, et dont le résultat sonore est plus réaliste que dans le modèle temporel. Par exemple, il est possible d'étirer un son dans le temps d'un facteur positif quelconque, sans pour autant en modifier ni la hauteur ni le « timbre ».

Pour charger et manipuler les sons, BOXES utilise une plate-forme logicielle appelée ProSpect, également développée au SCRIME. ProSpect est basé sur la synthèse additive de partiels dont l'amplitude et la fréquence varient lentement avec le temps, de l'ordre de 600 fois par seconde. En terme de performance, des recherches récentes ont permis la synthèse logicielle et en temps réel de 2 oscillateurs par MHz sur un ordinateur de type Pentium II, soit 800

oscillateurs sur une machine à 400 MHz [13]. Cela rend l'utilisation d'une telle synthèse tout à fait envisageable, même pour des compositions nécessitant simultanément un grand nombre de sons.

L'ensemble des sons que ProSpect permet de manipuler est vaste. Il peut s'agir de sons créés de toute pièce à partir de partiels d'amplitudes et de fréquences arbitraires. Il peut également s'agir de sons issus d'une analyse spectrale. Le logiciel InSpect [9] est utilisé dans ce dernier cas. Il prend en entrée un son du modèle temporel, qui peut être un son naturel enregistré de manière traditionnelle, et donne en sortie un son du modèle spectral. Il faut cependant tenir compte des erreurs produites lors de l'analyse, qui néglige les composantes bruitées et les transitoires des sons, c'est-à-dire les phénomènes sonores auxquels il n'est pas envisageable d'associer de partiels.

Actuellement, BOXES utilise la structure de données *spectral* de ProSpect pour les sons. Cette structure de données définit chaque son comme une somme de partiels. L'utilisateur n'a cependant pas directement accès aux partiels des sons, car il est délicat de les manipuler à un niveau de composition macroscopique : leur nombre est vite trop important pour pouvoir effectuer des opérations de haut niveau, musicales. Les sons, abstraction faite des partiels, sont les seuls objets que l'utilisateur peut véritablement manipuler, par l'intermédiaire de trois paramètres :

- la *durée*, une valeur réelle d positive ;
- l'*amplitude*, une fonction $A : t \to a$ du temps vers l'intervalle $[0, +\infty[$ des valeurs d'amplitude ;
- la *fréquence*, une fonction $F : t \to f$ du temps vers l'intervalle $[0, +\infty[$ des valeurs de fréquence.

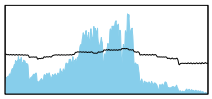
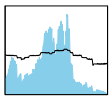
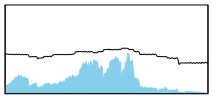
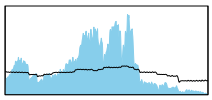
L'amplitude et la fréquence sont des paramètres synthétiques calculés par ProSpect à partir de leurs équivalents au niveau des partiels. Ce sont ces paramètres qui donnent les figures visibles à l'intérieur des boîtes son.

Il a été fait un compromis sur la nature des opérations que l'utilisateur peut effectuer sur chaque paramètre, afin de simplifier le développement de l'interface et d'aboutir rapidement à un résultat. Ces opérations se limitent actuellement à des multiplications par un facteur constant dans le temps. Par exemple, il est possible de modifier la durée d'un son, mais seulement en la multipliant par un facteur constant ; de même pour l'amplitude et la fréquence. La fréquence du son se retrouve donc uniformément transposée d'une hauteur donnée, ce qui est déjà une opération intéressante en soi. Malheureusement, cette dernière opération, telle qu'elle est actuellement implémentée, change également le timbre du son, malgré le fait que l'on soit dans un modèle spectral. Il s'agit d'un défaut de jeunesse qui disparaîtra avec les progrès autour de ProSpect.

Le tableau 1 résume les opérations sonores accessibles à l'heure actuelle depuis l'interface graphique, et montre pour chacune un exemple d'utilisation. Le son d'origine, noté S, est défini par ses trois paramètres d, A et F. Chaque opération modifie ces paramètres en fonction du résultat souhaité.

5 Structures hiérarchiques

L'utilisateur a la possibilité de définir des structures de plus haut niveau que les sons. Ainsi, il peut regrouper des éléments qu'il a déjà créés, afin d'obtenir par exemple des mélodies, des

$S = \langle\, d, A, F \,\rangle$	
stretch$(S, k) = \langle\, k \times d, A', F' \,\rangle$ avec $\begin{cases} A'(t) = A(t/k) \\ F'(t) = F(t/k) \end{cases}$	$k = 0.5$ 
amplify$(S, k) = \langle\, d, A', F \,\rangle$ avec $A'(t) = k \times A(t)$	$k = 0.5$ 
transpose$(S, k) = \langle\, d, A, F' \,\rangle$ avec $F'(t) = k \times F(t)$	$k = 0.25$ 

TAB. 1: Opérations sonores accessibles depuis l'interface graphique de BOXES.

accords, des pistes, etc. Une structure hiérarchique est un élément de la pièce composée, au même titre qu'un son. Elle est caractérisée par une date de début et une date de fin, et par un *contenu*. Ce contenu est un ensemble constitué d'autres éléments de la pièce. Ainsi, une structure hiérarchique peut contenir des sons mais aussi d'autres structures hiérarchiques.

Il est cependant exclu qu'elle se contienne elle-même, ou que l'une de ses sous-structures ne la contienne. Cela poserait des problèmes trop délicats dans l'état actuel du logiciel. Ceci exclu, il est donc possible de construire non seulement des arbres, mais aussi des graphes dirigés acycliques. Un même élément peut donc être présent dans plusieurs structures à la fois. Par exemple, un même son peut être à la fois dans une structure d'accord et dans une structure de mélodie, toutes deux faisant parties de la même pièce.

Une structure hiérarchique est représentée à l'écran comme une boîte. Elle n'est pas associée à des figures d'amplitude ou de fréquence. Si son contenu est vide, elle apparaît comme une boîte vide. Dans le cas contraire, c'est une boîte englobant les boîtes correspondant aux éléments qu'elle contient. Cette représentation graphique est malheureusement ambiguë. Par exemple, une boîte peut apparaître graphiquement englobée par une autre, sans pour autant en faire partie.

Le contenu d'une structure hiérarchique est modifiable dynamiquement dans BOXES. Il suffit par exemple de dessiner une nouvelle boîte ou de charger un son à l'intérieur d'une boîte hiérarchique existante pour que le contenu de cette dernière soit mis à jour. De même, il est possible de supprimer un élément d'une boîte hiérarchique, ou la boîte elle-même.

La durée de vie d'une telle structure est variable. S'il elle n'existe que pour déplacer momentanément un groupe de sons, elle peut être assimilée à une sélection, telle qu'on en trouve dans la plupart des logiciels actuels, et disparaîtra sans doute assez rapidement. Par contre, si elle correspond à une information conceptuelle importante comme une partie de la pièce, sa durée de vie est plus importante, l'utilisateur lui donnant vraisemblablement un rôle formel, structurant.

6 Contraintes

BOXES offre la possibilité d'ajouter ou de retirer des contraintes entre les éléments de la pièce composée. Ces contraintes sont des relations persistantes que le logiciel utilise afin de mettre la pièce à jour lorsque l'utilisateur fait une modification sur l'un des éléments contraints. Le fonctionnement en est simple. L'utilisateur ajoute par exemple une contrainte entre deux sons, qui précise que le premier doit finir avant que le deuxième ne commence. Dès lors, et à moins que l'utilisateur ne retire la contrainte, le logiciel s'efforcera de toujours la respecter. Cela signifie que si l'utilisateur déplace le premier son au-delà de la position actuelle du second, le logiciel acceptera la nouvelle position du premier *et déplacera automatiquement* le second afin que la contrainte soit à nouveau vérifiée. Cette opération se fait de manière interactive : l'utilisateur déplace le premier son et observe le déplacement automatique du second en « temps réel ».

La bibliothèque Cassowary [3] est utilisée pour résoudre les contraintes. Il s'agit d'une bibliothèque écrite en C++ qui s'inspire de l'algorithme du *simplex*. Par rapport au simplex, Cassowary rajoute des fonctionnalités spécifiques aux logiciels interactifs. Ainsi, il est possible de définir dynamiquement des *variables d'édition* parmi l'ensemble des variables disponibles. Lors d'une opération de « drag & drop » par exemple, la position de la souris est associée d'une manière ou d'une autre à des variables d'édition. Un déplacement de la souris entraîne de nouvelles valeurs pour les variables d'édition, ce qui provoque un calcul optimisé des valeurs des autres variables. Ce calcul doit être le plus rapide possible, afin que les résultats d'un déplacement de la souris soient visibles en temps réel. C'est la raison d'être de Cassowary.

Cassowary impose que les contraintes utilisées soient des équations ou des inégalités linéaires entre des variables prenant leurs valeurs dans l'ensemble des nombres réels. L'ensemble que constituent ces contraintes n'est pas assez vaste pour exprimer toutes les contraintes musicales que l'on pourrait souhaiter. Quoi qu'il en soit, il l'est déjà suffisamment pour exprimer de nombreuses contraintes musicales intéressantes.

Dans BOXES, l'ensemble des contraintes peut être divisé en deux sous-ensembles : les contraintes *implicites* et les contraintes *utilisateur*. Les contraintes utilisateur sont celles que l'utilisateur rajoute lui-même afin de préciser davantage les relations entre les éléments de sa pièce. Les contraintes implicites sont celles qui sont automatiquement rajoutées par le logiciel lorsque des boîtes sont construites et manipulées. Elles peuvent à leur tour être divisées en deux parties : les contraintes *intrinsèques* et les contraintes de *comportement*. Les sections suivantes développent chacun de ces points.

6.1 Variables contraintes

Chaque boîte est caractérisée par un ensemble de variables pouvant être contraintes. En ce qui concerne l'axe horizontal (le temps), les variables sont notées x_1, x_2 et w (pour *width*), représentant respectivement la date de début, la date de fin, et la durée de la boîte. Pour l'axe vertical, il existe également des variables, mais comme elles ne sont liées à aucune dimension musicale, il n'en sera plus fait mention.

Chaque boîte son est caractérisée par deux variables supplémentaires. L'une, notée a (pour *amplify*), correspond au facteur multiplicateur d'amplitude. Sa valeur initiale est 1, ce qui indique que l'amplitude du son n'est pas modifiée. L'autre variable, notée b (pour *bend*), représente une transposition en nombre de demi-tons positifs ou négatifs. Sa valeur initiale est 0 ; le son n'est donc pas transposé. Un calcul simple permet de passer de la valeur de b à un

facteur multiplicateur de fréquence (initialement 1, en l'occurrence). Le tableau 2 rappelle les variables qui caractérisent les boîtes dans BOXES.

Variable	Description	Boîte son	Boîte hiérarchique
x_1	date de début	$\surd$	$\surd$
x_2	date de fin	$\surd$	$\surd$
w	durée	$\surd$	$\surd$
a	facteur d'amplitude	$\surd$	*non*
b	valeur de transposition	$\surd$	*non*

TAB. 2: Variables caractérisant les boîtes.

Il peut paraître redondant d'utiliser une variable pour la durée alors que l'on a déjà des variables pour les dates de début et de fin. En fait, il n'en est rien, comme il est expliqué dans la section sur les contraintes de comportement.

6.2 Contraintes intrinsèques

Les contraintes intrinsèques servent à garantir la cohérence fondamentale des boîtes, comme par exemple le fait qu'un son se termine après qu'il ait commencé. Elles permettent également de maintenir les relations entre les boîtes hiérarchiques et leurs contenus. Ainsi, la date de début d'une boîte hiérarchique doit toujours demeurer plus petite que les dates de début de ses éléments. Inversement, sa date de fin doit demeurer plus grande que leurs dates de fin. Ces contraintes « hiérarchiques » peuvent sembler arbitraires et insuffisantes, mais elles permettent de construire toutes sortes de structures hiérarchiques, que des contraintes plus strictes empêcheraient. Le tableau 3 présente l'ensemble des contraintes intrinsèques associées aux boîtes. Les fonctions *min-pitch* et *max-pitch* retournent respectivement la plus petite et la plus grande hauteur MIDI d'une boîte son, sans tenir compte de la transposition.

Boîte son B	Boîte hiérarchique B
$w(B) = x_2(B) - x_1(B)$ $x_1(B) \leq x_2(B)$ $0 \leq w(B)$	
$0 \leq a(B)$ $0 \leq b(B) + \textit{min-pitch}(B)$ $b(B) + \textit{max-pitch}(B) \leq 127$	$\forall B' \in \textit{contenu}(B)$ $x_1(B) \leq x_1(B')$ $x_2(B') \leq x_2(B)$

TAB. 3: Contraintes intrinsèques.

6.3 Contraintes de comportement

Les contraintes de comportement déterminent la manière dont les éléments de la pièce sont mis à jour lors d'une opération interactive. En effet, lors d'une telle opération, la valeur d'au moins une variable d'édition est amenée à changer. Ce changement entraîne vraisemblablement certaines contraintes à ne plus être respectées. Il faut donc également changer les valeurs

d'autres variables, afin que les contraintes soient à nouveau respectées. Cependant, même avec cet objectif il peut exister une multitude de façons de changer les valeurs des autres variables. Les contraintes de comportement permettent de définir une solution « préférable » dans ce cas, même si certaines contraintes ne peuvent plus être respectées. Par exemple, dans la version actuelle de BOXES, on préfère déplacer les boîtes plutôt que de changer leur durée. Ceci est réalisé grâce à des *contraintes de fixation* (en anglais *stay constraints*) précisant que la durée d'une boîte doit être plus « résistante » que ses dates de début et de fin.

Les contraintes de comportement sont liées à la manière dont Cassowary travaille : il tente sans arrêt de minimiser le résultat d'une expression mathématique, appelée *fonction objective* dans l'algorithme du simplex. Les contraintes de comportement sont un moyen de définir quelles contraintes il est préférable de respecter, et ainsi de diriger l'optimisation effectuée par Cassowary. Pour faire un amalgame avec l'algorithme de résolution employée dans MusicSpace, il s'agit en quelque sorte de définir la manière dont les valeurs se propagent dans le graphe de contraintes, bien que Cassowary ne soit justement pas un algorithme basé sur la propagation de valeurs.

Les contraintes de comportement nécessiteraient à elles seules un développement au moins équivalent à cet article. C'est pourquoi leur présentation s'arrête là. Notons simplement que la présence de structures hiérarchiques dans BOXES introduit un traitement particulier de ces contraintes, par un rajout de variables et de contraintes, et par un réajustement dynamique des résistances des variables dans la hiérarchie.

6.4 Contraintes utilisateur

Les contraintes utilisateur sont celles que l'utilisateur peut ajouter volontairement afin de préciser des relations spécifiques sur les éléments de sa pièce. Le fait qu'un son doive commencer exactement au moment ou un autre se termine n'est pas une décision que le logiciel prend automatiquement. C'est à l'utilisateur de faire de tels choix, qui sont plus artistiques que mécaniques.

BOXES permet de rajouter facilement un certain nombre de contraintes utilisateur prédéfinies. Elles sont de différentes natures. Il existe des contraintes temporelles qualitatives, qui rappellent les relations élémentaires d'Allen sur les intervalles temporels [1]. Ces contraintes sont qualitatives parce qu'elles n'expriment que des relations d'ordre ou d'égalité entre des dates. BOXES offre également des contraintes temporelles quantitatives ou métriques. Il s'agit essentiellement de rapports de durées entre les sons et les structures temporelles, comme par exemple le fait qu'un son doive durer deux fois plus longtemps qu'un autre. Enfin, BOXES autorise des contraintes entre les amplitudes des sons, ou entre les hauteurs des sons. Ces contraintes ne sont pas temporelles. Le tableau 4 énumère les contraintes utilisateur prédéfinies. La fonction *val* utilisée dans ce tableau retourne la valeur d'une variable au moment de l'appel.

Toute contrainte utilisateur, une fois ajoutée à l'ensemble des contraintes de la pièce, doit être vérifiée, c'est à dire que les valeurs qui ne respectent pas ces contraintes, même partiellement, sont rejetées. C'est un pré-requis très fort, parfois trop. Les contraintes utilisateur peuvent également être retirées, ce qui atténue le pré-requis, mais pas de manière très satisfaisante. Enfin, lorsque l'utilisateur sauvegarde une pièce, les contraintes utilisateur sont sauvegardées avec la pièce, afin qu'elles réapparaissent lorsque la pièce est rechargée.

Contraintes entre deux boîtes B_1 et B_2 quelconques

B_1 **before** B_2	B_1 doit terminer avant que B_2 ne commence	$x_2(B_1) \leq x_1(B_2)$
B_1 **meets** B_2	B_2 doit commencer lorsque B_1 termine	$x_2(B_1) = x_1(B_2)$
B_1 **overlap** B_2	B_1 et B_2 doivent être « connectées »	$x_1(B_1) \leq x_2(B_2)$ $x_1(B_2) \leq x_2(B_1)$
B_1 **start** B_2	B_1 et B_2 doivent commencer en même temps	$x_1(B_1) = x_1(B_2)$
B_1 **end** B_2	B_1 et B_2 doivent terminer en même temps	$x_2(B_1) = x_2(B_2)$
B_1 **equal-duration** B_2	B_1 et B_2 sont de même durée	$w(B_1) = w(B_2)$
B_1 **prop.-duration** B_2	B_1 et B_2 ont des durées proportionnelles	$w(B_1) = \frac{val(w(B_2))}{val(w(B_1))} \times w(B_2)$

Contraintes entre deux boîtes son B_1 et B_2

B_1 **prop.-amplify** B_2	$a(B_1)$ est proportionelle à $a(B_2)$	$a(B_1) = \frac{val(a(B_2))}{val(a(B_1))} \times a(B_2)$
B_1 **prop.-pitch** B_2	$b(B_1)$ progresse comme $b(B_2)$	$b(B_1) = b(B_2) + val(b(B_1)) - val(b(B_2))$

Contraintes sur une seule boîte B

left-pin B	la date de début de B est fixe	$x_1(B) = val(x_1(B))$
right-pin B	la date de fin de B est fixe	$x_2(B) = val(x_2(B))$
min-width B	la durée de B doit être supérieure ou égale à sa valeur actuelle	$val(w(B)) \leq w(B)$
max-width B	la durée de B doit être inférieure ou égale à sa valeur actuelle	$w(B) \leq val(w(B))$

TAB. 4: Contraintes utilisateur prédéfinies.

7 Conclusion et perspectives

À l'usage, BOXES montre clairement le potentiel que l'on peut tirer de la combinaison d'un modèle spectral, de structures hiérarchiques et de contraintes. Même lorsque chacune de ces notions est réduite à sa plus simple expression, comme c'est le cas pour le moment, ce potentiel est tout à fait évident.

Il est envisagé d'étendre BOXES selon plusieurs directions. Les opérations sonores sont encore insuffisantes et implémentées naïvement. L'utilisation d'une bibliothèque développée autour d'un modèle spectral normalisé doit résoudre ces problèmes. Ce modèle sera sans doute SAS. Il pourrait être également intéressant de faire une abstraction par rapport au modèle spectral, afin de ne pas dépendre d'un modèle en particulier.

Pour ce qui est des structures hiérarchiques, l'utilisation de termes fonctionnels et des mécanismes d'abstraction et d'application est inévitable pour atteindre le pouvoir d'expression d'un véritable langage de programmation. Il serait alors possible d'automatiser la création de mélodies ou d'accords, par exemple. Ces opérations sont actuellement possibles, mais par l'intermédiaire de contraintes utilisateur dont la construction est à la charge de l'utilisateur.

De plus, les structures présentes dans BOXES sont uniquement temporelles. Or, il devrait être possible de regrouper des objets musicaux selon d'autres considérations. Il pourrait y avoir des structures liées au paramètre d'amplification ou de fréquence, afin par exemple de

répartir hiérarchiquement le volume ou la hauteur d'une partie de la pièce. La présence de variables d'amplification et de transposition au niveau des boîtes hiérarchiques, et non pas seulement sur les boîtes son, est nécessaire.

En ce qui concerne les contraintes, de nombreux développements sont encore possibles : représentation graphique appropriée, nouvelles contraintes utilisateur, contraintes non linéaires, disjonctions, comportement paramétrable, etc., rendront le logiciel plus compliqué mais aussi plus à même d'exprimer des idées musicales sophistiquées. Cependant, si l'on gagne en pouvoir d'expression, il est également probable que l'on perde en efficacité, ce qui pénaliserait le côté interactif des contraintes. Des compromis acceptables devraient pouvoir être trouvés.

Références

[1] Allen (J. F.). – Maintaining knowledge about temporal intervals. *Communications of the ACM*, vol. 26, pp. 832–843, 1983.

[2] Assayag (G.), Rueda (C.), Laurson (M.), Agon (C.), et Delerue (O.). – Computer-assisted composition at IRCAM : From PatchWork to OpenMusic. *Computer Music Journal*, vol. 23, n° 3, pp. 59–72, 1999.

[3] Badros (G. J.) et Borning (A.). – *The Cassowary Linear Arithmetic Constraint Solving Algorithm : Interface and Implementation.* – Rapport technique n° UW-CSE-98-06-04, `http://www.cs.washington.edu/research/constraints/cassowary`, University of Washington, 1998.

[4] Barzilay (Eli). – *BOOMS : Booms Object Oriented Music System.* – Thèse, Ben-Gurion University, 1996.

[5] Borning (A.), Freeman-Benson (B.), et Wilson (M.). – Constraint hierarchies. *Lisp and Symbolic Computation*, vol. 5, pp. 223–270, 1992.

[6] Desainte-Catherine (M.) et Marchand (S.). – Vers un modèle pour unifier musique et son dans une composition multiéchelle. Dans : *Journées d'Informatique Musicale, JIM'99*, pp. 59–68. – 1999.

[7] Free Software Foundation. – GNU General Public License. – `http://www.fsf.org`, 1991.

[8] Gallesio (E.). – STk. – I3S CNRS / Université de Nice - Sophia Antipolis, 1995. `http://kaolin.unice.fr/STk`.

[9] Marchand (S.) et Strandh (R.). – InSpect and ReSpect : spectral modeling, analysis and real-time synthesis software tools for researchers and composers. Dans : *Proceedings of the 1999 International Computer Music Conference*, pp. 341–344. – 1999.

[10] Orlarey (Y.), Fober (D.), et Letz (S.). – L'environnement de composition musicale *elody*. Dans : *Journées d'Informatique Musicale, JIM'97*, pp. 122–136. – 1997.

[11] Pachet (F.) et Delerue (O.). – MusicSpace : a constraint-based control system for music spatialization. Dans : *Proceedings of the 1999 International Computer Music Conference*, pp. 272–273. – 1999.

[12] Serra (X.). – *Musical Signal Processing*, chap. Musical Sound Modeling with Sinusoids plus Noise, pp. 91–122. – Swets & Zeitlinger, 1997, *Studies on New Music Research.*

[13] Strandh (R.) et Marchand (S.). – Real-time generation of sound from parameters of additive synthesis. Dans : *Journées d'Informatique Musicale, JIM'99*, pp. 83–88. – 1999.