Vers une programmation réactive structurée

Simon Archipoff, David Janin

VERS UNE PROGRAMMATION REACTIVE STUCTUREE

Simon Archipoff, David Janin

LaBRI, CNRS UMR 5800
INRIA Bordeaux Sud-Ouest
Bordeaux INP
Université de Bordeaux,
F-33405 Talence

simon.archipoff|janin@labri.fr

Résumé

Dans cet article, nous traitons de la programmation de systèmes musicaux interactifs lorsqu'ils sont soumis à des stimulations structurées telles que celles provenant d'un clavier maître. Un nouveau langage, dédié à la programmation musicale réactive, est proposé. Plus abstrait que les langages de programmation événementielle classiques, il reste cependant beaucoup plus simple d'utilisation. Mathématiquement bien structuré, il permet aussi une analyse automatique de la cohérence temporelle des programmes réactifs qu'on peut écrire.

1. INTRODUCTION

Dans cet article, nous nous intéressons au problème de la programmation de systèmes musicaux interactifs lorsqu'ils sont soumis à des stimulations structurées telles que celles provenant d'un clavier maître : chaque enfoncement de touche est nécessairement suivie d'un relâchement de cette même touche.

Des évènements aux notes. En rupture avec les approches classiques qui s'appuient sur une programmation événementielle générique qui traite de façon homogène l'ensemble des évènements reçus, nous souhaitons exploiter ici la « bonne structuration » des stimulations produites par le clavier.

Autrement dit, les flux d'entrées sont traités non pas comme des suites d'évènements indépendants, mais, au contraire, comme des suites d'évènements apparié qui définissent les bordures d'entités musicales atomiques : les notes produites au clavier. Ces notes sont alors les objets primitifs, reçus en entrée, qui peuvent être recopiés, composés et transformés, en changeant, à l'aide de primitives adéquates, tout ou partie de leurs attributs : hauteur, vélocité, durée, etc...

Des notes ou des gestes. Plus généralement, même si les exemples proposés traitent essentiellement de notes jouées au clavier, notre approche se généralise facilement à des objets musicaux plus complexes tels que, par exemple, des gestes musicaux. En effet, toutes les combinaisons, les traitements et les analyses proposés ci-dessous sont applicables aux cas d'objets temporisés qui se définiraient par un évènement de « début », un évènement de « fin », et une succession de valeurs d'attributs positionnés dans le temps entre ces deux évènements.

Cohérence temporelle. Idéalement, toutes les opérations et combinaisons de notes doivent pouvoir être autorisées. Néanmoins, certaines compositions peuvent être temporellement incohérentes. Par exemple, il est possible, dès le début de sa réception, de rejouer une note deux fois plus lentement. Par contre, il est impossible de la rejouer deux fois plus vite.

Avec suffisamment de mémoire, on peut en effet dilater l'échelle de temps des objets musicaux reçus. Par contre, pour des raisons de causalité évidentes, il est impossible de contracter cette échelle de temps. Notre approche, mathématiquement bien structurée, nous permet d'effectuer automatiquement une analyse de la cohérence temporelle des programmes construits.

Langage dédié. Le langage de programmation résultant, dédié à la programmation réactive structurée, est plus abstrait qu'un langage de programmation événementiel classique. Il semble offrir la possibilité de faire, tout à la fois, une description plus musicale et plus simple des réactions du système à concevoir.

En s'appuyant sur une sémantique de plus haut niveau, les programmes définis semblent plus robustes. Ainsi, de nombreux écueils de la programmation événementielle classique, délicate à mener et source de beaucoup d'erreurs, semblent pouvoir être évités.

2. LE LANGAGE T-SCORE

T-score, le langage de programmation musicale décrit à partir de maintenant est un *Domain Specific Language* [10], c'est à dire un langage de programmation dédié à un domaine d'application particulier : la programmation musicale réactive dans le cas qui

nous intéresse. Il est enchâssé dans le langage de programmation fonctionnel Haskell [12] dans lequel il est encodé et dont il hérite de la syntaxe (programmation fonctionnelle) et de la sémantique (sémantique paresseuse sans effets de bord).

D'un point de vue structurel, le langage s'appuie sur la modélisation et la programmation par tuilage développée ces dernières années [16, 13, 4, 1, 3]. A l'exécution, il s'appuie sur la bibliothèque UISF de programmation fonctionnelle réactive développée à l'Université de Yale [15] autour du langage de programmation musicale *Euterpea* [11].

Nous décrivons dans cette section la syntaxe et la sémantique statique des objets manipulés en T-score. La sémantique dynamique, c'est à dire réactive des fonctions écrites en T-score est décrite dans la section suivante.

2.1. Syntaxe

Pour programmer en T-score, il n'est pas nécessaire de comprendre en détail le modèle des tuiles sous-jacent. Retenons, comme décrit dans [14], qu'il s'agit d'un zigzag d'objets musicaux atomique dont on ne retient que le « début » du premier et la « fin » du dernier.

Les tuiles complexes sont construites à partir des tuiles atomiques et des opérateurs de tuiles décrits ci-dessous.

2.1.1. Les silences

Les constantes de T-score sont de deux types : les silences et les notes constantes. Les silences sont toutes les constantes rationnelles, entendues comme des durées exprimées en nombre de battements.

2.1.2. Les notes constantes

Les notes constantes sont de la forme X_n où X est une classe de pitch, et n est un entier positif indiquant le numéro de l'octave.

Les classes de pitch sont à choisir parmi les classes A, B, C, D, E, F, G et leurs altérations $A^\sharp$, $B^\sharp$, $C^\sharp$, $D^\sharp$, $E^\sharp$, $F^\sharp$, $G^\sharp$ ou $A^\flat$, $B^\flat$, $C^\flat$, $D^\flat$, $E^\flat$, $F^\flat$, $G^\flat$.

Par exemple, la note $A_4^\flat$ désigne le *la* bémol du quatrième octave. Par défaut, la durée de ces notes constantes est d'un battement.

2.1.3. Les variables

Les variables de T-score sont soit des lettres minuscules a, b, c, etc. Ou des mots commençant par une lettre minuscule.

2.1.4. Somme, négation, reset et coreset

La somme de deux tuiles a et b est notée $a + b$. La négation d'une tuile a est notée $-a$. Par convention, la somme $a + (-b)$ pourra être notée $a - b$.

Deux opérations dérivées qui jouent un rôle important sont le reset re a et le co-reset co a d'une tuile a. Elles sont définies par :

$$\text{re } a = a - a$$
$$\text{co } a = -a + a$$

2.1.5. Produit et quotient

Lorsque l'une des deux tuiles a ou b est une tuile constante, on peut définir le produit $a \times b$ des deux tuiles a et b.

Lorsque la tuile b est une tuile constante de durée non nulle [1], on peut définir le quotient a/b des deux tuiles a et b.

2.1.6. Transformations élémentaires

D'autres fonctions élémentaires, prédéfinies dans le langage T-score, permettent d'agir sur les tuiles. Par exemple, une fonction de silence mute joue un rôle fondamental. D'autres fonctions de transpositions, chromatique ou diatonique, ou des fonctions de test de durée sont présentées dans la partie suivante avec leur sémantique.

2.2. Sémantique

Nous allons maintenant décrire la sémantique de ces constantes, de ces opérateurs et des tuiles qu'ils permettent de construire.

Le parti pris que nous suivons ici est de tout voir comme des tuiles. Ainsi, les durées sont modélisés par des silences, les hauteur de note sont modélisés par des notes de durées fixe, et toute opération définie sur un silence ou une note peut être définie sur n'importe quelle autre tuile.

De plus, les notations avec addition, négation, produit et quotient nous conduisent à penser les tuiles en termes d'objets vivant dans une sorte d'espace vectoriel. Cette intuition a ses limites, par exemple la somme de deux objets n'est pas commutative, même si le cadre mathématique sous-jacent, robuste, s'en approche à n'en pas douter.

2.2.1. Somme de silences

La sémantique des silences positifs est sans surprise. Par exemple, la fraction $3/2$ décrit un silence de 3 demi-battements, soit une noire pointée si les battements sont des noires [2]. Par exemple, la composition de silence $3 + 5$ va définir un déplacement positif de 3 suivit de 5, soit 8 au total. Autrement dit, on aura $3 + 5 = 8$.

Pour comprendre la sémantique — un peu moins banale — des silences négatifs, on peut interpréter les

1 . en particulier, tout silence différent de zéro

2 . Dans toute la suite, on suppose que la durée d'un battement est la noire.

silences positifs comme des déplacement vers le futur. Placé devant une mélodie, un silence permet de repousser son exécution vers le futur. Avec ce point de vue, les silences négatifs peuvent être vus, au contraire, comme des déplacement vers le passé. Ainsi, la composition $3-5$ définit un déplacement positif de 3 suivi d'un déplacement négatif de 5, soit au total, un déplacement négatif de 2. Autrement dit, $3-5=-2$.

Plus généralement, un silence en T-score peut être vu comme un déplacement temporel de la durée indiquée par le silence, vers le futur dans le cas d'un silence positif, vers le passé dans le cas d'un silence négatif. L'opérateur de somme agit alors comme un opérateur de composition de ces déplacements, la composition de déplacements positifs et négatifs créant des zigzags temporels.

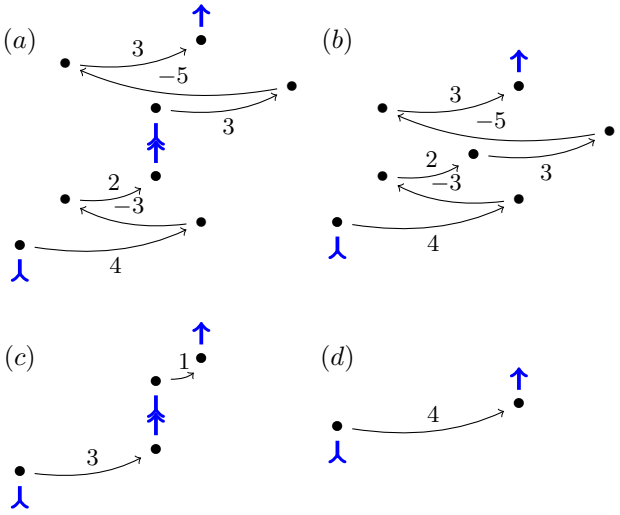

Figure 1: Zigzags de silences positifs et négatifs.

Ce point de vue est illustré figure 1 dans laquelle sont proposée plusieurs représentations de zigzags équivalents. Les flèches entrantes et sortantes verticales, permettent d'expliciter les « débuts » et les « fins » des zigzags.

Dans cette figure, la composition de zigzags (a) représente la somme des silences $4-3+2$ et $3-5+3$. Le zigzag (b) représente le résultat de cette composition, soit le zigzag de somme $4-3+2+3-5+3$. La composition de zigzags (c) représente la somme des silences 3 et 1. Le zigzag (d) représente le résultat de cette composition après calcul, soit le silence 4. Ce silence peut aussi être vu comme une normalisation du zigzag (b).

On constate que les silences composés à l'aide de sommes et de négations se composent exactement comme les rationnels. La sémantique du produit ou du quotient est sans surprise. Les silences se comportent donc comme les nombres rationnels, qu'on peut additionner ou soustraire.

Remarquons aussi que le silence 0 est un neutre pour l'addition et qu'il est sa propre négation. De plus, le reset re s et le co-reset co s de tout silence s est égal au silence nul 0.

2.2.2. Somme de notes et de silences (positifs)

La sémantique des notes constantes est, elle aussi, sans surprise. Chaque symbole de note désigne une note qui est jouée sur une durée de 1 battement, c'est à dire une noire.

La somme de deux notes (ou plus) définit la séquence de ces deux notes (ou plus), jouées l'une à la suite de l'autre. Autrement dit, la séquence
$C_4 + E_4 + G_4$
désigne simplement un *arpège* en do majeur, joué à la noire. Cet arpège est illustré figure 2.

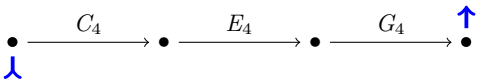

Figure 2: Représentation de l'arpège $C_4 + E_4 + G_4$.

2.2.3. Produit et quotient par des silences positifs

La sémantique du produit (ou du quotient) d'une note par un silence positif est, là aussi, sans surprise. Elle permet d'étirer ou de contracter la durée d'une note ou d'un silence par la valeur de ce silence.

Par exemple, le silence 3×2 désigne un silence de six battements. De même, $3 \times C_4$ est la note C_4 jouée sur trois battements. Les silences étant des rationnels, le quotient se définit tout aussi naturellement. Le silence $3/2$ désigne bien un silence d'un battement et demi, de même que la note $C_4/2$, ou bien $1/2 \times C_4$ définit la note C_4 d'une durée d'un demi-battement, c'est à dire une croche.

Ce produit par un silence s'étend naturellement aux séquences (ou mélodie) en distribuant sur la somme. Par exemple, la séquence

$$1/2 \times (C_4 + E_4 + G_4)$$

équivaut à la séquence $1/2 \times C_4 + 1/2 \times E_4 + 1/2 \times G_4$. Elle désigne toujours un *arpège* en do majeur mais qui est maintenant joué à la demi noire, c'est à dire à la croche, deux fois plus vite qu'à la noire. De la même manière, la séquence

$$2 \times (C_4 + E_4 + G_4)$$

désignera le même arpège, mais joué cette fois-ci à la blanche, soit deux fois plus lentement.

Autrement dit, avec des sommes et des produits par des silences positifs, on peut composer n'importe quelle mélodie à une seule voix. De plus, la possibilité de multiplier une sous-mélodie par un facteur donné

peut être vue comme un changement de métrique/-tempo.

2.2.4. Mise en parallèle avec reset et co-reset, et fonction mute

Pour alléger les calculs de durées qui peuvent vite se révéler fastidieux, on dispose aussi de la fonction mute qui prend en entrée n'importe quelle combinaison de notes et de silences et qui en donne sa durée. Plus précisément, mute m est, pour toute mélodie m, le silence de même durée que m. En particulier, pour tout silence s, on a mute $s == s$.

Étant donnée une mélodie m la sémantique du reset re m de la mélodie m consiste à déplacer le marqueur de fin de mélodie sur son marqueur de début. De façon duale, le co-reset co m de la mélodie m consiste à déplacer le marqueur de début de mélodie sur son marqueur de fin. Ces opérations sont illustrée par la figure 3.

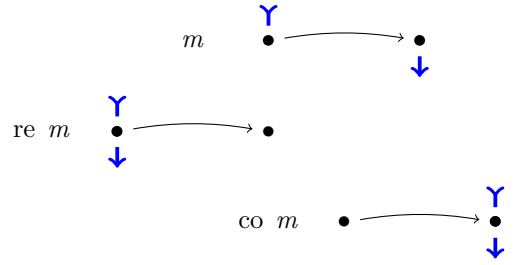

Figure 3: Reset et coreset d'une mélodie m.

Ainsi, comme décrit figure 4, la somme

$$\text{prodI } m_1\ m_2 = (\text{re } m_1) + m_2$$

revient à lancer la mélodie m_1 au début de la mélodie m_2, et la somme

$$\text{prodT } m_1\ m_2 = m_1 + (\text{co } m_2)$$

revient à finir la mélodie m_2 à la fin de la mélodie m_1.

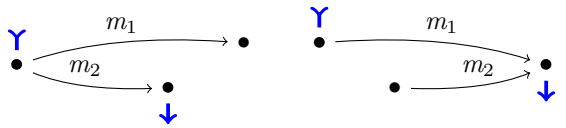

Figure 4: Lancement parallèle avec synchronisation initiale prodI ou terminal prodT.

2.2.5. Produit de mélodies

On convient de généraliser le produit à toute mélodie. Intuitivement, le produit $m_1 \times m_2$ de deux mélodies m_1 et m_2 se définit comme la mise en parallèle de la mélodie m_1 étirée de la durée de m_2 avec la mélodie m_2 étirée de la durée de m_1.

On vérifie que cette définition est compatible avec le produit de silence définit ci-dessus, puisque, étant donné deux silences s_1 et s_2, la mise en parallèle de $s_1 \times s_2$ avec $s_2 \times s_1$ ne fera, par commutativité du produit, que définir du produit des durées. Le produit généralisé peut être décrit grâce au produit restreint à une opérande non silencieuse avec :

$$\begin{aligned} m_1 \times m_2 == \text{re } &(m_1 \times (\text{mute } m_2)) \\ &+ (m_2 \times (\text{mute } m_1)) \end{aligned}$$

Un exemple de produit de deux mélodies est donné par

$$(C_4 + E_4 + G_4) \times C2$$

Il est illustré figure 5. Autrement dit, avec le produit généralisé et le quotient par un silence de durée constante, on peut maintenant décrire toute mélodie polyphonique.

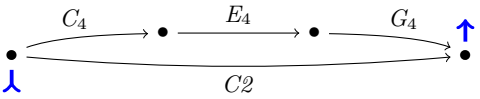

Figure 5: Ajout d'une basse à l'arpège $C_4 + E_4 + G_4$.

2.2.6. Produit élastique

La fonction mute nous permet de définir une sorte de produit parallèle élastique qui permet de préserver la durée du premier argument.

$$\text{prodE } m_1\ m_2 = m_1 \times m_2 / (\text{mute } m_2)$$

qui est défini pour toutes mélodies m_1 et m_2, dès lors que la durée d'une mélodie m_2 n'est pas nulle. En notant $m_3 = (\text{mute } m_1)/(\text{mute } m_2) \times m_2$, cette construction est illustrée figure 6.

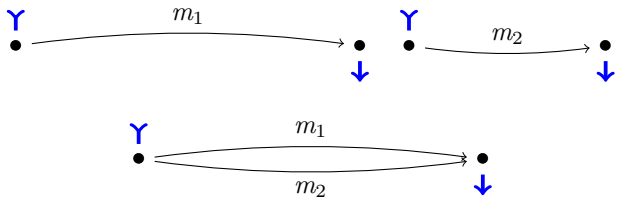

Figure 6: Produit élastique

2.3. Somme généralisée

Dans les exemples précédents, on voit apparaître une notion de points de synchronisation généralisées qui n'apparaissent pas forcément en début ou en fin de mélodie.

Plus précisément, comme déjà évoqué dans le cas des silences, les tuiles manipulées sont en général plus complexe que les mélodies évoquées ci-dessus. En effet, dans notre approche, une tuile se compose d'un ensemble de notes et de silences disposés dans le temps, par exemple une mélodie polyphonique, augmenté de deux points de synchronisations, respectivement synchronisation d'entrée et synchronisation de sortie, qui peuvent être vue comme deux barres de mesures positionnées dans le temps par rapport à la mélodie support.

La somme de deux de deux tuiles m_1 et m_2 ainsi enrichis consiste alors à faire coïncider dans le temps la synchronisation de sortie de m_1 avec la synchronisation d'entrée de m_2, les tuiles pouvant être partiellement superposés. Cette situation est décrite Figure 7.

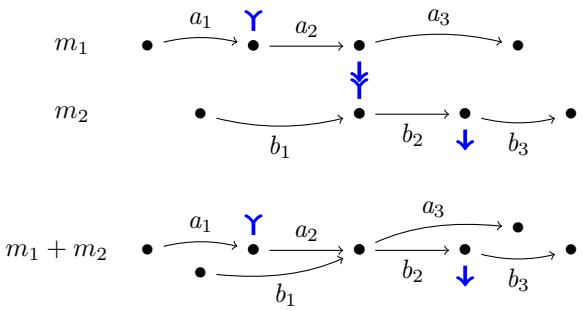

Figure 7: Somme généralisée de deux tuiles, en haut, les deux mélodies sont synchronisées, en bas, fusionnées.

2.3.1. Négation généralisée

La sémantique de la négation appliquée à une mélodie, ou, plus généralement, à une tuile n'est alors plus un mystère. Cette opération revient juste à intervertir les synchronisation d'entrée et de sortie comme illustrée figure 8. Ainsi, le résultat d'une somme de

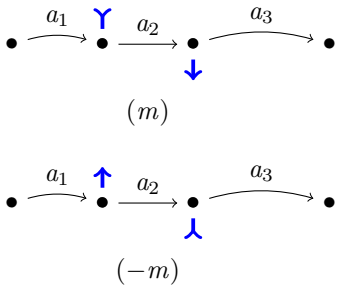

Figure 8: Négation d'une tuile

notes dans lequel apparaît des négations (ou des déférences avec la convention d'écriture évoquée ci-dessus) définit un zigzag temporel [14] dont la mélodie sous-jacente est obtenue en quelque sorte par projection sur l'axe temporel.

2.3.2. Durée généralisée

La durée (ou mute) d'une tuile avec synchronisation généralisé est alors défini non pas comme la durée de la mélodie sous-jacente, mais elle est définie comme la durée (relative) du marqueur d'entrée vers le marqueur de sortie comme illustrée figure 9.

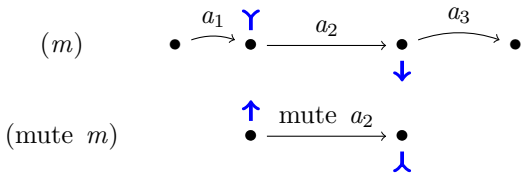

Figure 9: Durée d'une tuile.

2.3.3. Les variables

Derniers ingrédients de notre proposition de langage musical : les variables. Elles se comportent comme des notes mais dont on ignore, a priori, la valeur et la durée. Elles sont typiquement utilisée pour définir des fonctions.

Par exemple, on peut définir la fonction

$$f_1\ x = x + 1/2 \times x$$

qui reçoit une note x en argument et reproduit cette note x suivie de sa répétition deux fois plus rapide $1/2 \times x$. De même, on peut définir la fonction

$$f_2\ x = x + \text{transpose}\ x\ 2 + \text{transpose}\ x\ 4$$

qui reçoit la note x pour rejouer un arpège de tierces mineures à partir de cette note. Dernier exemple, on peut tout aussi bien écrire la fonction

$$f_3\ x = 1/2 \times x$$

qui reçoit une note x et la rejoue deux fois plus vite.

2.3.4. Causalité

L'examen de ce dernier cas nous indique que, bien que le sens de la fonction f_3 soit parfaitement bien définit, son exécution pourrait poser un problème. En effet, une variable représente une tuile reçue à un instant donné dont on va découvrir, avec le temps qui passe, les attributs. Certaines transformations sont donc impossible puisqu'elles violent des contraintes évidentes de causalité temporelle.

Dans le cas le plus simple qui nous intéresse ici, lors de la réception d'une note x, la hauteur de cette note est connue, de même que sa vélocité, dès le début de sa réception. Ces attributs peuvent donc être utilisé sans attendre. Par contre, la durée de x ne sera connue qu'à la fin de la réception de la note.

Alors que les fonctions f_1 et f_2 peuvent être appliquées à une note dès sa réception, la fonction f_3 ne

pourra pas l'être puisque, pour pouvoir jouer la note x deux fois plus vite, il faudra connaître sa durée avant la fin de sa réception ce qui est impossible.

Autrement dit, en supposant que la note x est reçue à l'appel des fonctions ci-dessus, les expressions $f_1\ x$ et $f_2\ x$ sont valides alors que l'expression $f_3\ x$ temporellement incohérente.

Il apparaît que notre langage est défini de tel sorte qu'une analyse de la causalité est possible. Elle est même, dans une large classe de cas d'utilisation, décidable et complète. La description détaillée des mécanismes qui permettent cette analyse de causalité sort du cadre de cet article.

Retenons cependant que la fonction mute, qui nous donne la durée des tuiles, reste parfaitement définissable sur les variables dès lors qu'on s'autorise une manipulation symbolique, avec des inconnues. On peut facilement calculer la date de chaque évènement apparaissant dans une tuile en fonction de la durée qui le sépare du marqueur de synchronisation d'entrée. L'analyse de ces durées et leur comparaison permet alors de vérifier la cohérence temporelle de ces expressions.

3. ARCHITECTURES RÉACTIVES

Nous avons déjà évoqué à la fin de section précédente la possibilité de décrire les réactions d'un système musical provoqués par l'arrivée d'une ou plusieurs notes. Dans cette section nous examinons plusieurs mécanismes permettant de décrire ces réactions.

3.1. Éléments de réactions

Les systèmes réactifs que nous souhaitons décrire se composent principalement :

- **(a)** d'un moteur de réaction qui indique à quels instants le système va réagir,
- **(b)** de paramètres externes de réactions qui indiquent quelles informations le système va automatiquement fournir au programmeur pour définir ces réactions,
- **(c)** de fonctions de réactions qui indiquent, au moment où elles sont appelées, quels objets vont être produits par le système,
- **(d)** de paramètres internes de réactions (ou états) qui indiquent quel type d'information peut être accumulé au cours des réactions successives afin de conditionner les réactions suivantes.

Ces éléments conditionnent largement le type de réactions que nous pourrons attendre d'un système musical réactif. Les éléments (a) et (b) sont définis par l'architecture système sur laquelle sont exécutés les programmes. Les éléments (c) et (d) sont, quant à eux, définis par les programmes réactifs eux-mêmes. Nous pouvons alors étudier et classifier les programmes réactifs par, d'une part, le mode de réaction ainsi que les informations mis à leur disposition pour calculer leur réaction, et, d'autre part, par le type de leurs états, c'est à dire les informations qu'ils engrangent eux-mêmes au cours du déroulement du système.

Par exemple, la possibilité de pouvoir entretenir un état, le mettre à jour et le consulter augmente significativement le pouvoir expressif des programmes. Cela peut aller d'un état ne contenant qu'un bit d'information, tel que, « est-ce que la dernière note reçue était un *la* ? », jusqu'à l'enregistrement de toute une partie de l'entrée afin de pouvoir la reproduire par la suite.

Il semble que la structuration des entrées que nous proposons de prendre en compte permet d'enrichir cette classification de « pouvoir de réactions ».

3.2. Moteur et paramètres de réactions

La première question est de de savoir à quoi réagit un programme réactif. Nous introduisons pour cela trois types de réactions.

La réaction « événementielle ». Une fonction de réaction est évaluée au début de chaque arrivée de note ou groupe de notes nouvelles sur l'entrée du système. Cette fonction, écrite dans le langage T-score, définit une mélodie qui dépend des notes reçues.

Le moteur d'évaluation de la mélodie produite met à jour, automatiquement et au fil de leurs terminaisons, la durée des notes reçues pour qu'elles puissent être prises en compte.

Le cas échéant, afin de pouvoir prendre en compte l'écoulement du temps, le système transmet aussi la mesure du temps (symbolique) écoulé depuis la réaction précédente.

Les sorties d'un système à réaction événementielle sont donc largement structurée en fonction de ses entrées. Nous sommes dans une logique de programmation réactive de type « push », qui pourrait être appelée synchrone puisqu'il n'y a pas de file d'attente pour gérer la réception des évènements d'entrée.

Cette situation est décrite figure 10 où on suppose qu'une seule note n_i est reçue à chaque instant de réaction, ces instants étant séparés du délai d_i. Les

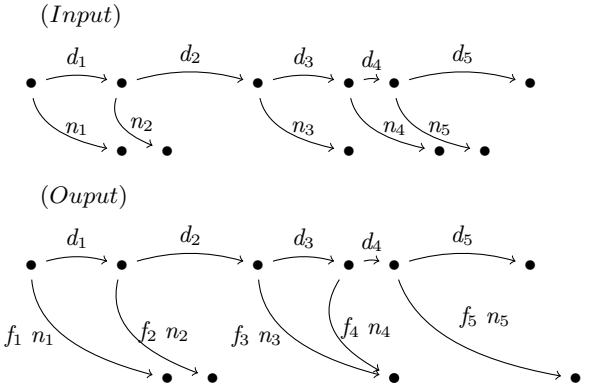

Figure 10: Chaîne de réactions événementielles.

mélodies produites ($f_i\ n_i$) sont lancées en parallèle des délais d_i, l'avancement de la mélodie globale qui reste synchronisés sur l'arrivée des notes.

Un mécanisme de feedback avec états doit permettre de faire dépendre chaque fonction f_i d'un état courant qui mémorise partie de l'historique des réactions. Les réactions élémentaires pourraient être de la forme ($f'_i\ d_{i-1}\ n_i$) pour prendre en compte les durées écoulées entre chaque arrivée de note.

La réaction « temporisée ». Une fonction de réaction est évaluée après écoulement d'un délai fixé lors de la réaction précédente. Cette fonction, écrite dans le langage T-score, définit une tuile qui dépend de ce délai et, des notes reçues, qu'elles soient terminées ou non, reçues pendant ce délai et qui ont été automatiquement collectées par le système. Typiquement, le délai d'attente jusqu'à la réaction suivante est codée par la durée (au sens des marqueurs de synchronisation) de la mélodie produite.

Le cas échéant, pour prendre en compte leur placement dans le temps, ces notes sont transmises à la fonction de réaction accompagnées de leur date relative d'arrivée et de leur durée si elle sont échue.

Les entrées d'un système à réaction temporisé sont donc largement structurées et traitées en fonction des sorties du système : le délai de réaction qui a été programmé. Nous somme dans une logique de programmation réactive de type « pull », qui pourrait être appelé asynchrone puisque les évènements reçus sont mis en attente pour être traités à la demande du programme réactif.

Cette situation est décrite figure 11 sous le même schéma d'entrée que précédemment. Dans cette figure, les tuiles produites sont de la forme

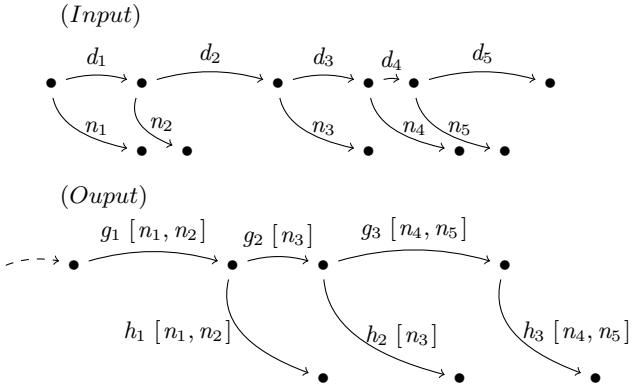

Figure 11: Chaîne de réactions temporisées.

$k\ l = (g\ l) + \mathrm{re}\ (h\ l)$
où g et h sont deux fonctions et l la liste des notes reçues pendant l'intervalle précédent. La fonction h permet de continuer la réaction à une temporisation au delà de la réaction suivante.

Là encore, un mécanisme de feedback avec valeur d'état doit permettre, plus généralement, de faire dépendre chaque fonctions g_i et h_i d'un certain état courant. Nous l'avons déjà évoqué, la suite des notes reçues peut aussi être enrichie de leurs dates (relatives) de réception.

La réaction « mixte ». Une fonction de réaction est évaluée soit à la réception de nouvelles notes, soit après l'écoulement d'un délai fixé qui aura été spécifié lors de la réaction précédente.

Cette fonction, écrite dans le langage T-score, définit une tuile qui dépend de la différence entre le délai d'attente spécifié et l'attente (inférieure ou égale) a été effectué, ainsi que des notes reçues.

Là encore, la spécification du délai d'attente jusqu'à la réaction suivante est codée par la durée (au sens des marqueurs de synchronisation) de la mélodie produite.

Remarquons que ce principe de réaction mixte revient à mettre en oeuvre des attentes réactives (de type « push ») avec une garde temporelle (de type « pull »).

3.3. Exemples de programmes réactifs

Nous donnons maintenant quelques exemples typiques de programmes réactifs qui peuvent être réalisés à l'aide de l'un des schémas de réaction évoqués ci-dessus.

3.3.1. Réaction sans états

Les programmes réactifs les plus simple fonctionnent par simple substitution ; à chaque entrée n on associe une sortie $f\ n$ implicitement synchronisée avec l'entrée, pour une fonction f fixée une fois pour toute. Par exemple, la fonction f peut être l'une des fonctions suivantes :

$f_1\ n = n$
$f_2\ n = 2 \times n$
$f_3\ n = n + 1/2 \times n$

qui implémente respectivement : le recopie pour f_1, le ralentissement de chaque note pour f_2, la répétition accélérée pour f_3. Cette dernière réaction est décrite figure 12 dans le cas où l'entrée est jouée legato, chaque note d'entrée succédant à la précédente.

3.3.2. Réaction événementielle avec états

Une application typique de système musical avec réaction événementielle avec états est le métapiano [9]. Dans cette approche, en première approximation, on stocke dans l'état courant une partition à jouer. Chaque note d'entrée est rejouée presque à l'identique en ne changeant que son pitch donnée par la partition, l'état courant étant mis à jour en avançant dans cette partition.

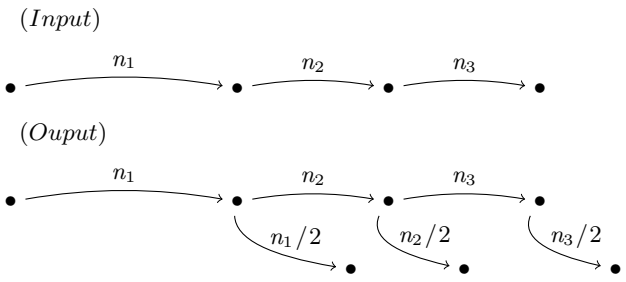

Figure 12: Répétition avec accélération.

Une autre application typique, mise en oeuvre dans le langage T-score, tout à fait analogue, consiste à avancer de façon cyclique dans une liste des 12 classes de pitch ordonnées de façon arbitraire. Chaque note d'entrée est rejouée presque à l'identique en préservant son octave mais en changeant sa classe de pitch. Ce système est actuellement en cours d'expérimentation [2].

3.3.3. Réaction temporisés avec états

Dans les réactions temporisés, la sortie n'est pas produite lorsque de nouvelles données sont disponibles en entrée, mais lorsque le marqueur de fin de la tuile produite en sortie est atteint.

Un exemple de ce type de comportement est décrit figure 13 où, à intervalle régulier, sont jouées en accords toutes les notes reçues pendant l'intervalle précédent.

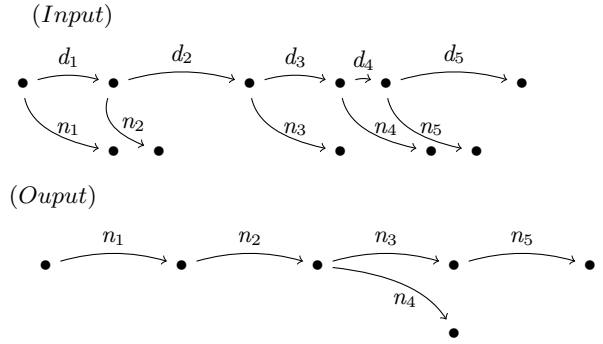

Figure 13: Sélection temporisé.

L'exemple de sélection temporisé décrit ci-dessus peut être vue comme un dual du métapiano. En effet, alors que l'approche metapiano [9] fixe le pitch des notes à jouer en suivant une partition purement « mélodique », cette approche fixerait au contraire leur placement temporelle en suivant une partition purement « rythmique » .

3.3.4. Réaction mixtes avec états

Naturellement, on peut aussi combiner les deux approches décrites ci-dessus. L'écriture de nombreuses applications pourra sans doute être facilitée par la définition de réaction mixte, qui combinent les deux comportements réactif et événementiel.

Dans une telle approche, le système répond soit à l'arrivée de nouvelles notes (ou de nouveaux gestes musicaux), soit à l'arrivée du marqueur de fin de la tuile en cours de production. En un sens, un tel système de réactions mixtes revient à ajouter une garde temporelle à toute attente d'évènement, la réaction provoquée pouvant distinguer ces deux cas.

Un exemple typique de réactions mixtes est celui d'un harmoniseur (doublant l'entrée sans la retarder) avec accompagnement rythmique régulier. Le comportement événementiel permet de jouer et transposer de façon réactives les notes d'entrée, le comportement temporisé permet de produire un accompagnement qui suivrait un tempo donné. Dans ce cas, même si le musicien décide de « jouer » avec la pulsation, en se plaçant selon son gout en avance ou en retard, le tempo de l'accompagnement automatiquement généré reste stable.

Par exemple, on peut ainsi écrire une ligne de basse, jouée au métronome, dont le choix des notes produites reste dépendant des notes reçues et du contexte harmonique qu'elles induisent.

En particulier, il semble que tous les systèmes d'accompagnement avec suivi de partition tel que [7] peuvent être mis en œuvre par ce type de codage quoique cette affirmation mérite d'être vérifiée en détail, le tempo et l'avancement dans la partition étant géré en amont par le module, de suivie de partition.

3.4. Expressivité comparée

Il est évident que les réactions événementielles ou temporisées sont deux cas particulier de réactions mixtes.

Pour ce qui est de comparer les réactions évènementielles avec les réactions temporisées, il est tout aussi évident que, en fixant un pas de résolution temporelle, on peut simuler tous les types de comportements décrit ici à l'aide des seules réactions temporisées. Cela revient à implémenter dans le langage T-score des fonctions d'attente active.

Une telle approche aura cependant deux inconvénients. Tout d'abord, cette attente active pourrait être inutilement consommatrice de resources, interdisant
d'implémenter aux niveau du moteur de reaction une attente événementielle qui serait plus paresseuse, par exemple basée sur un mécanisme d'activation par interruption, déléguant aux systèmes d'exploitation et aux systèmes hardware sous-jacent la tâche de réaliser à moindre frais cette attente active, le pas de résolution temporelle devenant alors indépendant du programme réactif décrit.

De plus, et c'est là un inconvénient beaucoup plus grave, une telle approche forcerait le concepteur du système réactif à écrire un code d'implémentation

bien plus qu'un code de spécification du système musical à mettre en œuvre. Le gain en abstraction offert par le langage T-score serait largement noyé dans de tels détails d'implémentation. Ce serait donc contraire à nos objectifs qui est d'offrir une description structurelle, musicale, du système à réaliser.

Autrement dit, il semble raisonnable, malgré l'existence de ce codage, de supposer ces deux types de réactions, temporisée ou événementielle, comme définissant deux approches incomparables.

Plus encore, sous l'hypothèse d'une résolution temporelle arbitrairement précise, quasi-continue, cette supposition devient même une vérité qui doit pouvoir être démontrée. Dans ce cas, le principe de réaction mixte décrit ci-dessus est donc strictement plus expressif que chacun des autres principes pris séparément.

Il nous faut cependant reconnaître qu'à ce jour, la mise au point d'une interface de programmation adéquate, c'est à dire suffisamment abstraite pour une écriture musicale simplifiée et pertinente, est une piste qui reste encore largement à explorer notamment à travers d'autres expérimentations.

4. CONCLUSIONS

Nous avons donc présenté le langage de programmation T-score, un langage dédié à l'écriture de système musicaux réactifs.

De nature algébrique, il permet de produire des objets complexes à partir d'objets plus simples. Les objets primitifs sont les notes elles-mêmes ou les silences. Ce ne sont plus les évènements comme dans les autres formalismes plus courants. Grâce à cela, il est tout à la fois simple et abstrait. Cette caractéristique devrait en faciliter l'usage.

Techniquement, la mise en œuvre de cette facilité a cependant constitué un défi conceptuel et technique inédit. La simplicité du « front-end » est le fruit d'un « back-end » bien plus subtil à mettre en œuvre. Il ne fait aucun doute aux auteurs que la faisabilité de ce back-end, qui a finalement pris plusieurs années de recherche, a largement reposé sur la robustesse du cadre mathématique sous-jacent.

Bien entendu, cette approche orientée « notes » a déjà été explorée. Par exemple, le langage Elody [17] aujourd'hui abandonné, le langage Euterpea[11], ou encore, quoique plus implicitement, l'outil OpenMusic [5], offrent déjà de telles interfaces.

Cependant, aucune de ces approches de semble permettre de définir simplement des réactions à la réception de notes dont la durée va dynamiquement dépendre de la durée des notes reçues. Remarquons au passage que, autant que nous puissions en juger, notre proposition viole le critère de synchronisme des langages réactifs temps-réel puisque, en théorie, cette prise en compte de la durée des notes en train nécessite de disposer d'une mémoire non bornée, proportionelle à la taille du codage de la durée des notes reçues.

Notre approche pourrait être comparée à l'approche offerte par Max/MSP [6], pilier de nombreux systèmes musicaux réactifs. En première approximation, avec
Max/MSP, la gestion des synchronisations, retards ou délais, est laissé largement au programmeur. La structure temporelle d'une pièce réactive risque donc d'être un peu noyée dans les détails techniques. Ce qui rend cette approche largement incomparable avec notre proposition plus abstraite.

Néanmoins, de nombreux langages ou librairies spécifiques ont été développées en sur-couche de Max/MSP tels que, parmi d'autres, Antescofo [8]. Ces nouvelles approches offrent aujourd'hui des points de vue structurels sans doute largement comparables à notre approche. L'étude comparée de ces autres formalismes reste cependant à faire.

Remarquons pour finir, comme illustré tout au long de cette article, que les tuiles manipulées ici se prêtent aussi à des représentations graphiques. On peut donc envisager de créer une interface graphique qui permettrait d'écrire des programmes dans le langage T-score. Cette autre piste de recherche est aussi à explorer plus en détails.

5. REFERENCES

[1] S. Archipoff. An efficient implementation of tiled polymorphic temporal media. In *Work. on Functional Art, Music, Modeling and Design (FARM)*, pages 25–34. ACM, 2015.

[2] S. Archipoff, J. Arias, E. Buger, and D. Janin. Interpolations : écriture de contraintes réeactives pour improvisations pianistiques. Technical report, LaBRI, Université de Bordeaux, 2016.

[3] S. Archipoff and D. Janin. Pour un raffinement spatio-temporel tuilé. In *Journées Francophones des Langages Applicatifs (JFLA)*, Saint-Malo, France, 2016.

[4] T. Bazin and D. Janin. Flux média tuilés polymorphes : une sémantique opérationnelle en Haskell. In *Journées Francophones des Langages Applicatifs (JFLA)*, 2015.

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

[6] Alessandro Cipriani and Maurizio Giri. *Electronic Music and Sound Design - Theory and Practice with Max/Msp.* Contemponet, 2010.

[7] A. Cont. Antescofo : Anticipatory synchronization and control of interactive parameters in computer music. In *Int. Computer Music Conference (ICMC)*, 2008.

[8] A. Cont, J. Echeveste, J.-L. Giavitto, and F. Jacquemard. Correct automatic accompaniment

despite machine listening or human errors in antescofo. In *Int. Computer Music Conference (ICMC)*, 2012.

[9] J. Haury. La pianotechnie ou notage des partitions musicales pour une interpretation immediate sur le metapiano. In *Actes des Journées d'informatique Musicale (JIM)*, 2009.

[10] P. Hudak. Keynote address - the promise of domain-specific languages. In *Proceedings of the Conference on Domain-Specific Languages (DSL)*, 1997.

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

[12] P. Hudak, J. Hugues, S. Peyton Jones, and P. Wadler. A history of Haskell : Being lazy with class. In *Third ACM SIGPLAN History of Programming Languages (HOPL)*. ACM Press, 2007.

[13] P. Hudak and D. Janin. Programmer avec des tuiles musicales : le T-calcul en Euterpea. In *Actes des Journées d'informatique Musicale (JIM)*, 2014.

[14] P. Hudak and D. Janin. From out-of-time design to in-time production of temporal media. Research report, LaBRI, Université de Bordeaux, 2015.

[15] P. Hudak, D. Quick, M. Santolucito, and D. Winograd-Cort. Real-time interactive music in haskell. In *Work. on Functional Art, Music, Modeling and Design (FARM)*, pages 15–16. ACM, 2015.

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

[17] S. Letz, Y. Orlarey, and D. Fober. Real-time composition in Elody. In *Int. Computer Music Conference (ICMC)*, pages 336–339. ICMA, 2000.