Un langage basé sur des arbres pour la description de partitions musicales.

Dominique Fober, Yann Orlarey, Stéphane Letz, Romain Michon

UN LANGAGE BASÉ SUR DES ARBRES POUR LA DESCRIPTION DE PARTITIONS MUSICALES

D. Fober *Y. Orlarey* *S. Letz* *R. Michon*
Grame-CNCM
Lyon - France
{fober, orlarey, letz, michon}@grame.fr

RÉSUMÉ

Le travail présenté s'inscrit dans le projet INScore, un environnement pour la conception de partition interactives augmentées, tourné vers des usages non conventionnels de la notation et de la représentation de la musique, sans exclure pour autant les approches classiques. Cet environnement est entièrement pilotable par des messages Open Sound Control [OSC]. Un langage de script, basé sur une version textuelle étendue de ces messages permet de concevoir des partitions sous forme modulaire et incrémentale. Cet article présente une révision majeure de ce langage de script, fondée sur la description et la manipulation d'arbres.

1. INTRODUCTION

Il existe un grand nombre de langages de description de partitions musicales (Lilypond [10], Guido [9], MuseData [8], MEI [11], MusicXML [7] etc.), tous axés sur une notation symbolique occidentale.

Certains de ces langages ont été étendus, afin de leur ajouter de la *programmabilité*, comme par exemple des opérations de composition de partitions musicales dans Guido [5], ou le langage Scheme dans Lilypond. Il existe aussi des langages de programmation dédiés à la notation musicale commune, comme CMN [12] ou ENP qui sont en fait des dialectes de Lisp.

L'approche proposée par INScore [4] est assez différente : la notation musicale symbolique est supportée (via le langage et le moteur de rendu Guido [2, 9]), mais elle constitue un des moyens de représentation musicale parmi d'autres, sans être privilégiée. Des partitions purement graphiques peuvent être conçues. Tous les éléments d'une partition (y compris les éléments purement graphiques) ont une dimension temporelle (date, durée et tempo) et peuvent être manipulés dans l'espace graphique et temporel. La notion de temps est à la fois événementielle et continue [6], ce qui permet de concevoir des partitions interactives et dynamiques. La figure 1 présente un exemple d'une partition réalisée avec INScore. Elle inclut de la notation symbolique, des images, une vidéo et des curseurs (la vidéo en est un) dont les positions sont synchronisées par les gestes de l'interprète.

INScore a été initialement conçu pour être piloté par des messages OSC [13]. OSC est conçu un protocole de

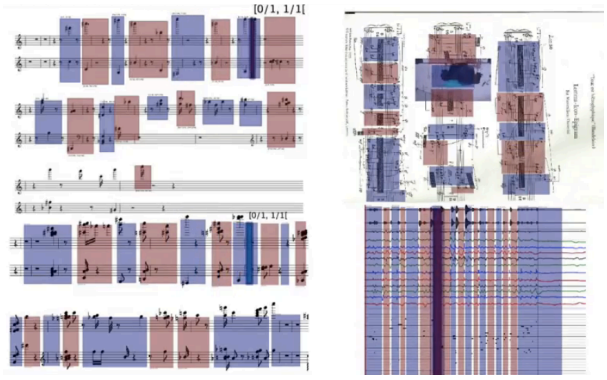

Figure 1. Une partition réalisée avec INScore, utilisée dans le cadre d'un environnement basé sur des capteurs, dédié au traitement et à l'apprentissage de musiques complexes appelé *GesTCom* (*Gesture Cutting through Textual Complexity*) [1].

communication. Une version textuelle des messages OSC constitue le format de stockage d'INScore, qui a été étendu en un langage de script, [3] permettant une plus grande flexibilité dans la conception des partitions musicales. Ces extensions (variables, adresses étendues, section Javascript, etc.) souffrent néanmoins d'une rigidité inhérente à une conception ad hoc et incrémentale. Par exemple, l'analyseur syntaxique fait une distinction claire entre les adresses OSC et les données associées, ce qui empêche l'utilisation de variables dans les adresses OSC. Une révision majeure de ce langage s'est donc avérée nécessaire. Elle est basé sur la manipulation d'une structure arborescente régulière qui est également homogène au modèle d'INScore. La figure 2 donne un exemple d'une telle hiérarchie du modèle, qui peut être décrite dans le langage de script courant (i.e. OSC) en listant toutes les branches à de l'arbre à partir de la racine. (Figure 3)

Aucune de ces approches ne couvre l'angle très particulier que nous avons adopté : celui d'un langage de description de partitions musicales basé sur la structure hiérachique à la fois de la partition et des messages OSC.

Après quelques définitions, nous présenterons les opérations élémentaires sur les arbres ainsi que la grammaire correspondante. Nous introduirons ensuite les opérations mathématiques sur les arbres, les concepts de *variables* et

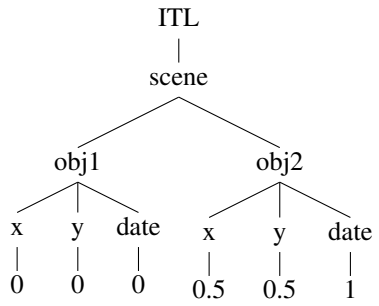

Figure 2. Exemple d'une partition INScore comprenant 2 objets $obj1$ et $obj2$, qui possèdent des attributs x, y, et $date$. Les feuilles de l'arbre sont les valeurs des attributs.

```
/ITL/scene/obj1 x 0;
/ITL/scene/obj1 y 0;
/ITL/scene/obj1 date 0;
/ITL/scene/obj2 x 0.5;
/ITL/scene/obj2 y 0.5;
/ITL/scene/obj2 date 1;
```

Figure 3. Script qui décrit la partition en figure 2.

de *valeurs expansées*, et nous présenterons enfin la transformation de ce langage en messages OSC classiques avant de conclure.

2. DÉFINITIONS

Un arbre t est constitué d'une valeur v (d'un certain type de données) et de la liste (éventuellement vide) de ses sous-arbres.

$$t : v \times [t_1, ..., t_k]$$

Un arbre avec une liste de sous-arbres vide $t : v \times [\,]$ s'appelle une feuille.
Une valeur est soit une valeur littérale (i.e. texte, nombre) soit une valeur spéciale parmi les types suivants :

- forêt ($\varnothing$) : désigne un arbre comprenant uniquement des sous-arbres,
- operator mathématique : indique une opération mathématique entre sous-arbres,
- variable : désigne un arbre dont la valeur fait référence à un autre arbre,
- expand : indique un arbre à développer,
- slash (/) : utilisé pour la conversion en OSC

L'utilisation et l'évaluation de ces valeurs sont détaillées dans les sections suivantes.

3. OPERATIONS SUR LES ARBRES

Nous définissons deux opérations abstraites sur les arbres : la mise en séquence et en parallèle. Ces opérations n'ont pas de sémantique musicale, ni d'un point de vue temporel, ni d'un point de vue graphique. Elles sont définies comme des méthodes de construction algorithmique de messages OSC et opèrent sur l'organisation topologique des arbres.

3.1. Mise en séquence

Mettre deux arbres t et t' en séquence consiste à ajouter t' comme enfant de toutes les feuilles de t. Nous noterons $|$ l'opération de mise en séquence.
Soit 2 arbres $t : v \times [t_1, ..., t_k]$ et t'. Alors :

$$v \times [t_1, ..., t_k] \mid t' \rightarrow v \times [t_1 |\, t', \ldots, t_k |\, t']$$

avec :

$$\begin{cases} v \times [\,] \mid t' \rightarrow v \times [t'] \\ v \times [\,] \mid \varnothing \times [t_1, ..., t_k] \rightarrow v \times [t_1, ..., t_k] \end{cases}$$

La flèche droite ($\rightarrow$) indique le résultat de l'évaluation d'une expression.

3.2. Mise en parallèle

Mettre deux arbres t et t' en parallèle consiste à les mettre dans une même forêt. Nous noterons $\|$ l'opération de mise en parallèle. Soit 2 arbres t et t' :

$$t \parallel t' \rightarrow \varnothing \times [t, t']$$

Le résultat est un arbre dont la valeur $\varnothing$ denote une forêt. La parallelisation appliquée à une forêt préserve l'ordre des sous-arbres :

$$\begin{cases} \varnothing \times [t_1, ..., t_k] \parallel t' \rightarrow \varnothing \times [t_1, \ldots, t_k, t'] \\ t' \parallel \varnothing \times [t_1, ..., t_k] \rightarrow \varnothing \times [t', t_1, \ldots, t_k] \end{cases}$$

4. GRAMMAIRE

Un arbre est syntactiquement définit en BNF comme suit :

```
tree := value       → t := value [ ]
     | tree tree    → t := tree | tree
     | / tree       → t := '/' | tree
     | tree , tree  → t := tree ‖ tree
     | ( tree )     → t := tree
     ;
```

La flèche droite ($\rightarrow$) indique l'arbre créé pour chaque construction syntactique. L'arbre dont la valeur est *slash* (*/*) joue un rôle spécial dans la conversion en messages OSC. Ce rôle est décrit en section 6.

5. VALEURS ET ÉVALUATION

Cette section décrit comment sont évalués les arbres ayant les valeurs spéciales *opérateur mathématique*, *variable* et *expand*.

5.1. Opérateurs mathématiques

Les opérations mathématiques sur des arbres sont vues comme des opérations qui portent sur leurs valeurs tout en conservant les sous-arbres. Ces opérations comprennent les opérations arithmétiques et logiques, les fonctions trigonométriques et hyperboliques, les fonctions exponentielles et logarithmiques, la puissance, la racine carrée, etc.

Nous désignerons ces opérations par `op`. Puis pour 2 arbres $t : v \times [t_1, ..., t_k]$ et $t' : v' \times [t'_1, ..., t'_k]$:

$$\texttt{op} \times [t, t'] \rightarrow (\texttt{op}\ v\ v') \times [t_1, \ldots, t_k, t'_1, \ldots, t'_k]$$

Exemple :

Le script en figure 4 genère l'arbre illustré en figure 5 après évaluation des opérations mathématiques.

```
/A (b ((* (math.sin 1), 2))),
   (c (math.pow 2.2, 0.4)),
   (d (math.max 1, 2, 3));
```

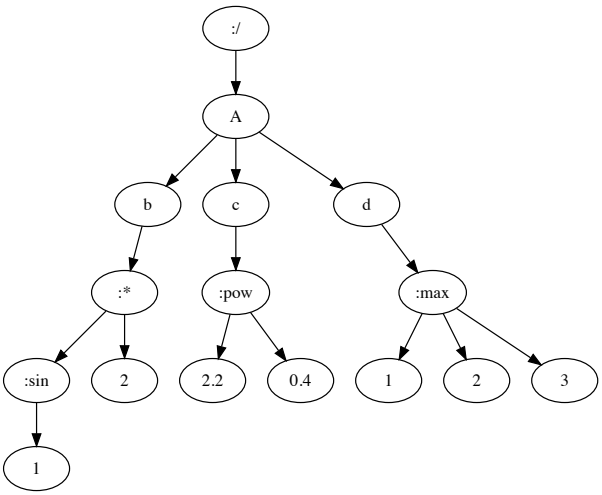

Figure 4. Arbre construit par le script ci-dessus.

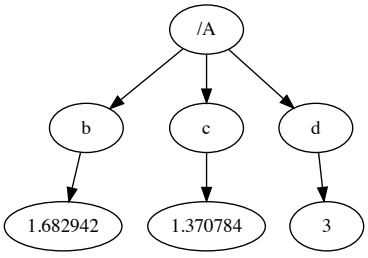

Figure 5. Arbre généré après évaluation des opérations mathématiques de l'arbre en figure 4.

5.2. Variables

Le type de valeur *variable* désigne un arbre dont la valeur fait référence à un autre arbre. L'évaluation d'un arbre variable consiste à développer l'arbre référencé à la position variable. Définissons une variable var et un arbre variable t' comme suit :

$$\begin{cases} var = v \times [t_1, ..., t_k] \\ t' : \$var \times [t'_1, ..., t'_k] \end{cases}$$

alors

$$t'_{\{var\}} \rightarrow v \times [t_1, ..., t_k] \mid \varnothing \times [t'_1, ..., t'_k]$$

t'$_{\{var\}}$ représente l'arbre t' dans un environnement contenant la définition de la variable var.

Exemple :

```
x = x 0;
y = y 0;
/A/B $x, $y;    ⇒ /A/B (x 0), (y 1);
```

5.2.1. *Environnement local à une variable*

Chaque arbre est évalué dans un environnement constitué de la liste de toutes les variables de son parent. Cependant, une variable peut-être évaluée dans un environnement qui lui est local. Les variables locales sont définies entre accolades :

$$\begin{cases} var = t \\ \$var\{a = t1, b = t2, \ldots\} \end{cases} \rightarrow t_{\{a,b,\ldots\}}$$

5.3. Valeurs expansées

Une *valeur expansée* est une forme compacte de description d'une forêt. On peut les voir également comme une manière de décrire des boucles. Leur forme syntactique est la suivante :

$id[n \ldots m]$ où n et m sont des entiers

$id[ab \ldots xy]$ où a, b, x, y sont des lettres.

Nous noterons ε l'opération d'expansion :

$$\begin{cases} \varepsilon(id[n \ldots m]) \rightarrow \varnothing\ [id_n, id_{n+1}, \ldots, id_m] \\ \varepsilon(id[ab \ldots xy]) \rightarrow \varnothing\ [id_{ab}, id_{ac}, \ldots, id_{ay}, \\ \qquad\qquad\qquad\qquad \ldots, \\ \qquad\qquad\qquad\qquad id_{xb}, id_{xc}, \ldots, id_{xy}] \end{cases}$$

où chaque id_n est un arbre $v \times [\,]$ dont la valeur v est la concaténation de la valeur de base id et de l'index courant n.

5.3.1. *Formes particulières*

Une *valeur expansée* peut également prendre les formes particulières suivantes :

$id[i : n \ldots m]$ où i est un identificateur

$id[i : j : ab \ldots xy]$ où i, j sont des identificateurs.

Ces identificateurs désignent des variables qui sont instanciées dans l'environnement par l'opération d'expansion, avec la valeur d'indice courante, i.e. :

$$\varepsilon(id[i : n \ldots m]) \rightarrow \varnothing\ [id_{n\{i=0\}}, \ldots, id_{m\{i=m-n\}}]$$

6. CONVERSION EN OSC

Un message OSC est constitué d'une adresse OSC (similaire à un chemin Unix) suivie d'une liste de données (qui peut éventuellement être vide) La valeur spéciale *slash* d'un arbre est utilisée pour distinguer l'adresse OSC et les données. Pour ce faire, nous allons *typer* les valeurs et nous définissons @ comme le type d'une valeur faisant partie d'une adresse OSC. Nous noterons $\texttt{type}(v)$ pour faire référence au type de la valeur v.

Nous noterons t^a pour désigner un arbre t dont la valeur est de type @ . Ensuite, nous définissons une opération Γ qui transforme un arbre en arbre *typé* :

$$\Gamma(v \times [t_1, ..., t_k]) \rightarrow \begin{cases} \varnothing \times [t_1^a, ..., t_k^a], v = / \\ v \times [t_1, ..., t_k], v \neq / \end{cases}$$

La conversion d'un arbre t en messages OSC transforme l'arbre typé $\Gamma(t)$ en une forêt d'adresses OSC suivies de données :

$$\texttt{osc}(v \times [t_1, ..., t_k]) \rightarrow$$

$$\begin{cases} \varnothing \times [v \times \texttt{osc}(t_1), \ldots, v \times \texttt{osc}(t_k)], \texttt{type}(v) = @ \\ v \times [\texttt{osc}(t_1), \ldots, \texttt{osc}(t_k)], \qquad \texttt{type}(v) \neq @ \end{cases}$$

La figure 6 donne une exemple de transformation d'un arbre en chemins OSC.

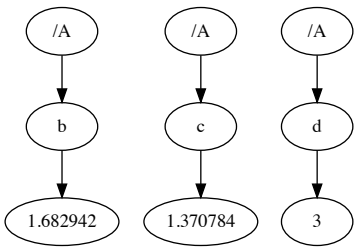

Figure 6. Transformation de l'arbre de la figure 5 en chemins OSC.

7. EXEMPLE DE SCRIPT

Le script ci-dessous présente un exemple de cette nouvelle version du langage INScore. Les variables sont indiquées en bleu, les variables locales sont déclarées en rouge.

```
# variables declaration
pi    = 3.141592653589793;

# the 'step' variable makes use of 'count'
# a variable that is instantiated locally
step  = / ( * 2, $pi), $count;

# the variable 'i' is defined by the
# expansion of the address 'n_[i:1...9]'
x = math.sin ( * $step, $i );
y = math.cos ( * $step, $i );
```

```
# the following variables select part of
# guido music notation code to
# build a short score
dyn = (? (% $i, 3), '\i<"p">', '\i<"ff">');
note = (+ $dyn, "␣", (?  (% $i, 2),
        "e2",
        "g1/8"));

# this is a classical OSC message
# that simply clears the scene
/ITL/scene/* del;

# this is the main variable. It will be
# expanded to create a series of small
# scores. The variables are computed
# using locally defined variables.
notes = (/ITL/scene/$addr
        (set gmn (+ "[", $note, "]")),
        (scale 0.7),
        (x * $x, $radius),
        (y * $y, $radius));

# finally the 'notes' variable is used
# addr, count and radius are local.
# which could be viewed as a function call
$notes{ addr=n_[i:1...9],
        count=9,
        radius=0.7};
```

L'évaluation de ce script produit des messages OSC pleinement compatibles avec la version précédente du langage et qui sont schématiquement présentés ci-dessous.

```
//ITL/scene/n_1 set gmn '[ ı<"ff"> g1/8]';
/ITL/scene/n_1 scale 0.7;
/ITL/scene/n_1 x 0.0;
/ITL/scene/n_1 y 0.7;
/ITL/scene/n_2 set gmn '[ ı<"p"> c2]';
/ITL/scene/n_2 scale 0.7;
/ITL/scene/n_2 x 0.411449;
/ITL/scene/n_2 y 0.566312;
...
...
/ITL/scene/n_9 set gmn '[ ı<"ff"> c2]';
/ITL/scene/n_9 scale 0.7;
/ITL/scene/n_9 x -0.411452;
/ITL/scene/n_9 y 0.56631;
```

Dans la pratique, cet exemple exprime la scène illustrée en figure 7 en quelques lignes seulement.

8. CONCLUSION

A partir de deux opérations élémentaires sur des arbres - la mise en séquence et en parallèle -, nous avons pu introduire de manière homogène, les notions de variables et d'opérations mathématiques et logiques portant sur des arbres. Le langage résultant est d'une expressivité sans commune mesure avec la version précédente du langage

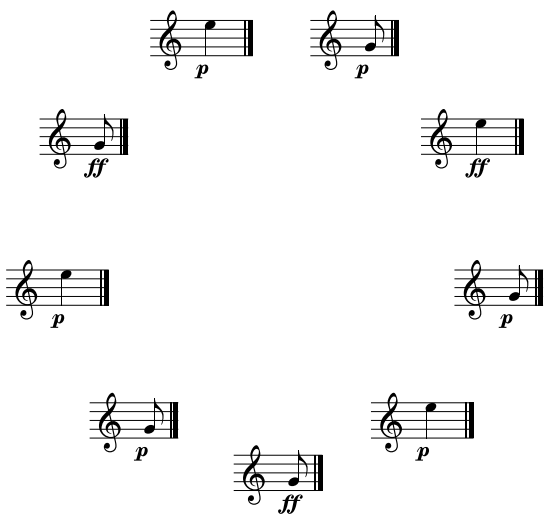

Figure 7. Partition INScore correspondant au script ci-dessus.

de script d'INScore. Il permet notamment la mise en parallèle des arguments d'un message, d'utiliser des variables pour décrire des adresses, d'exprimer des séries d'adresses de manière concise, et d'utiliser des variables locales permettant la réutilisation de scripts ou de parties de scripts dans des contextes différents.

9. REFERENCES

[1] Pavlos Antoniadis. *Embodied navigation of complex piano notation : rethinking musical interaction from a performer's perspective*. Theses, Université de Strasbourg, June 2018.

[2] C. Daudin, Dominique Fober, Stephane Letz, and Yann Orlarey. The Guido Engine – A toolbox for music scores rendering. In LAC, editor, *Proceedings of Linux Audio Conference 2009*, pages 105–111, 2009.

[3] Dominique Fober, Stéphane Letz, Yann Orlarey, and Frederic Bevilacqua. Programming Interactive Music Scores with INScore. In *Proceedings of the Sound and Music Computing conference – SMC'13*, pages 185–190, 2013.

[4] Dominique Fober, Yann Orlarey, and Stéphane Letz. INScore - An Environment for the Design of Live Music Scores. In *Proceedings of the Linux Audio Conference – LAC 2012*, pages 47–54, 2012.

[5] Dominique Fober, Yann Orlarey, and Stéphane Letz. Scores Level Composition Based on the Guido Music Notation. In ICMA, editor, *Proceedings of the International Computer Music Conference*, pages 383–386, 2012.

[6] Dominique Fober, Yann Orlarey, and Stéphane Letz. INScore Time Model. In *Proceedings of the International Computer Music Conference*, pages 64–68, 2017.

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

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

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

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

[11] Perry Roland. The Music Encoding Initiative (MEI). In *MAX2002. Proceedings of the First International Conference on Musical Application using XML*, pages 55–59, 2002.

[12] Bill Schottstaedt. *Common Music Notation.*, chapter 16. MIT Press, 1997.

[13] Matthew Wright. *Open Sound Control 1.0 Specification*, 2002.