Composition de partitions symboliques dans INScore

Gabriel Lepetit-Aimon, Dominique Fober, Yann Orlarey, Stéphane Letz

COMPOSITION DE PARTITIONS SYMBOLIQUES DANS INSCORE

G. Lepetit-Aimon *D. Fober* *Y. Orlarey* *S. Letz*
Grame
Centre nationale de création musicale
Lyon - France
{gabriel.lepetit.aimon,fober,orlarey,letz}@grame.fr

RÉSUMÉ

INScore est un environnement pour la conception de partition interactives augmentées, tourné vers des usages non conventionnels de la notation musicale. L'environnement permet d'utiliser et de composer des ressources graphiques arbitraires pour la représentation de la musique aussi bien que de la notation symbolique aux formats GMN (Guido Music Notation) ou MusicXML. INScore a été étendu pour fournir des opérations de composition de partitions en notation symbolique avec un jeu d'opérateurs qui de manière consistante, prennent des partitions en entrée pour produire une partition en sortie. L'API d'INScore inclut des *score expressions*, aussi bien au niveau OSC que dans son langage de script. Le travail présenté est basé sur une recherche précédente qui a porté sur les problèmes de consistance de la notation musicale à travers des opérations de composition de partitions. Ce sont les aspects langage et stratégies d'évaluation des *score expressions* qui sont abordés ici.

1. INTRODUCTION

La notation musicale fait face à nombreux défis posés par la création contemporaine. Les musiques spatialisées, les nouveaux instruments, les partitions interactives et temps-réel, sont parmi les nouveaux domaines couramment explorés par les artistes. La notation musicale conventionnelle ne couvre pas les besoins de ces nouvelles formes musicales et de nombreuses recherches et approches ont récemment émergé, témoignant d'une certaine maturité des outils informatiques pour la notation et la représentation de la musique. Les problèmes posés par la spatialisation de la musique [1], par l'écriture pour de nouveaux instruments [2] ou de nouvelles interfaces [3] (pour n'en citer que quelques uns), sont maintenant le sujet de recherches actives et de propositions originales.

Les musiques interactives et les partitions temps-réel représentent également un domaine en pleine expansion dans le champ de la création musicale. Les partitions numériques et la maturation des outils informatiques pour la notation et la représentation de la musique constituent les fondements du développement de ces formes musicales, souvent basées sur des représentations non traditionnelles de la musique [4, 5] sans toutefois occulter son usage [6, 7].

Afin de répondre aux besoins évoqués ci-dessus, INScore [8, 9] a été conçu comme un environnement ouvert aux représentations non conventionnelles de la musique (tout en supportant la notation symbolique), et tourné vers l'interactivité et les usages temps-réel [10, 11]. INScore est spécialisé sur la représentation de la musique uniquement et diffère en cela des outils intégrés dans des environnements de programmation tels que Bach [12] ou MaxScore [13].

INScore a été étendu avec des *score expressions* qui fournissent des opérations de composition de partitions (e.g. composer des partitions en séquence ou en parallèle). La création de partitions par composition de partitions existantes en notation symbolique n'est pas à proprement parler une innovation. La module Haskore de Haskell permet ce genre d'opérations [14]. Freeman et Lee ont également proposé des opérations de composition de partition en temps-réel dans un contexte de notation interactive [15]. En ce qui concerne les opérateurs de composition de partitions d'INScore, elles proviennent d'un travail de recherche précédent [16] qui portait sur la consistence de la notation musicale à travers des opérations de composition arbitraires.

La nouveauté de l'approche proposée repose sur les aspects dynamiques des opérations de composition ainsi que sur la persistence des expressions de composition. Une partition peut être composée comme un graphe arbitraire d'expressions et équipée d'un contrôle fin sur la propagation des changements qui affectent les composants de ces expressions. Dans une certaine mesure, l'évaluation de ces expressions se rapproche des aspect réactifs qui constituent une extension récente d'Open Music [17], sans en avoir toute la généralité. Sa spécificité repose notamment sur l'homogénéité des entrées et des sorties de ces expressions, d'où résulte un langage de composition de partitions simple, exempt d'erreur de *connexion*.

Cet article présente tout d'abord les expressions de composition de partitions. Les différentes stratégies d'évaluation sont ensuite exposées et illustrées d'exemples. Puis, l'articulation avec l'environnement d'INScore est décrite en détail et suivie de cas d'usages concrets. L'article introduit ensuite l'extension du design initial de composition de partitions à la composition d'expressions. La généralisation de cette approche à l'ensemble des objets graphiques d'INScore est finalement abordée en conclusion.

2. SPÉCIFICATION DU LANGAGE

L'idée directrice de ce travail est de concevoir un langage simple pour composer et manipuler des partitions en notation symbolique. Les opérateurs ayant été définis précédemment [16], l'enjeu est ici de fournir une interface maniable pour les utiliser dans INScore, et surtout d'exploiter pleinement les aspects dynamique de l'environnement.

2.1. Les opérateurs

Tous les opérateurs du langage ont une interface commune : ils prennent deux partitions en entrée pour produire une partition en sortie. L'uniformité de cette interface permet de combiner ces opérateurs pour créer des opérateurs de plus haut niveau, ce qui permet d'étendre le jeu d'opérations initiales.

La table 1 donne la liste des opérateurs. Les partitions sont exprimées au format Guido Music Notation (GMN) [18]. Il n'y a aucune contrainte sur les partitions en entrée. Pour les opérations `par` et `rpar` et le cas échéant, la partition la plus courte est suffixée ou préfixée avec la durée nécessaire pour les rendre *égales*. Ces extensions apparaissent comme des des portées vides, ce qui s'exprime aisément en GMN.

2.2. Le langage

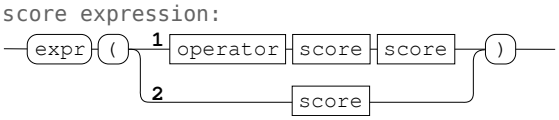
Dans le contexte des outils informatiques pour la création musicale, la conception d'un langage délimite un cadre pour le processus de création, auquel les artistes ne pourront échapper. Dans cette mesure, la définition d'un langage définit également un *workflow* que l'utilisateur devra adopter. L'uniformité des entrées et des sorties des opérateurs permet de composer par transformation et aggregation successive de partitions ou de fragments de partitions. Dans une certaine mesure, ce processus (appliquer des transformations sur divers matériaux et les combiner en un tout) est similaire au processus de création en électroacoustique où, après avoir choisi ses matériaux sonores, le compositeur les transforme, applique des effets, les mixe... jusqu'au résultat final où le matériau d'origine n'est souvent plus reconnaissable.

L'adaptation de cette approche à la notation symbolique la situe dans une même métaphore, plaçant la structure musicale au centre du processus de création où la qualité artistique émerge des processus qui transforment et lient des fragments musicaux ensemble. C'est dans cette perspective que la syntaxe des expressions du langage a été définie. En particulier ces expressions peuvent utiliser divers matériaux hétérogènes, y compris d'autres expressions ou des objets existants de la partition.

Bien que cette approche se situe conceptuellement dans la lignée des outils de CAO, les opérateurs de base proposés ne sont pas destinés à couvrir le processus de composition à la manière d'Open Music [19] ou de Bach [12], mais fournir des outils de calcul de partition dynamique, en particulier pour le contexte de la performance musicale.

2.3. Syntaxe

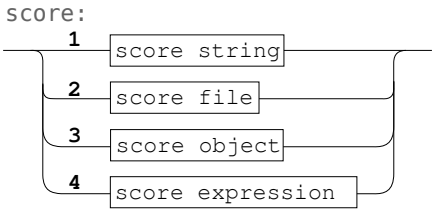
Une *score expression* peut être définie de la manière suivante :



1. la première forme [1], reflète l'interface des opérateurs : deux partitions sont combinées pour en produire une nouvelle.
2. la deuxième forme [2] défini une expression comme une simple partition, ce qui trouve son utilité pour la duplication d'objets, par exemple pour en fournir des vues différentes (voir section 6.2)

A noter que le mot clé `expr` est présent pour lever l'ambiguité des parenthèses qui sont également utilisées dans les scripts INScore pour définir des listes de messages.

Les deux formes prennent des `score` comme arguments. Le système est permissif pour leur type :



1. `score string` : une partition au format GMN ou MusicXML, définit de manière littérale.
2. `score file` : fait référence à un fichier au format GMN ou MusicXML. Les chemins d'accès doivent suivre les règles définies par INScore et peuvent indiquer des chemins absolus, relatifs ou encore une URL.
3. `score object` : fait référence à un objet existant d'INScore en utilisant une adresse OSC relative ou absolue.
   Cet objet doit être un objet Guido (`gmn`, ou `gmnf`), musicxml (`musicxml`, ou `musicxmlf`) ou pianoroll (`pianoroll`, ou `pianorollf`).
   Les streams sont également supportés (`gmnstream`, ou `pianorollstream`).
4. `score expression` : une *score expression* peut être utilisée comme argument d'une *score expression* (dans ce cas, le mot clé `expr` est optionnel).

Voici un exemple de *score expression* qui met 2 partitions en parallèle avec 2 partitions en séquence :

```
expr( par score.gmn (seq "[c]" score) )
```

3. EVALUATION D'UNE EXPRESSION

Une expression du langage est tout d'abord transformée en une représentation interne en mémoire. Dans un second temps, cette représentation est évaluée pour produire du code au format GMN en sortie, qui est finalement passée à l'objet INScore comme données spécifiques.

operation	arguments	description
seq	$s1$ $s2$	met les partitions $s1$ et $s2$ en séquence
par	$s1$ $s2$	met les partitions $s1$ et $s2$ en parallèle
rpar	$s1$ $s2$	met les partitions $s1$ et $s2$ en parallèle, alignées à droite
top	$s1$ $s2$	prend les n premières voix de $s1$ où n est le nombre de voix de $s2$
bottom	$s1$ $s2$	coupe les n premières voix de $s1$ où n est le nombre de voix de $s2$
head	$s1$ $s2$	prend le début de $s1$ sur la durée de $s2$
evhead	$s1$ $s2$	prend les n premiers événements de $s1$ où n est le nombre d'événements de $s2$
tail	$s1$ $s2$	coupe le début de $s1$ sur la durée de $s2$
evtail	$s1$ $s2$	coupe les n premiers événements de $s1$ où n est le nombre d'événements de $s2$
transpose	$s1$ $s2$	transpose $s1$ pour que sa première note soit la première note de $s2$
duration	$s1$ $s2$	étire ou compresse $s1$ à la durée de $s2$ (cette opération peut produire des durées qui ne sont pas affichables)
pitch	$s1$ $s2$	applique les hauteurs de $s2$ à $s1$ en boucle
rhythm	$s1$ $s2$	applique le rythme de $s2$ à $s1$ en boucle

Table 1. Liste des opérateurs

3.1. Représentation interne d'un expression

Quand il rencontre une *score expression*, le parser INScore en crée une représentation mémoire sous forme d'arbre : les feuilles contiennent les arguments et les opérateurs sont des noeuds (Figure 1). Cette forme d'arbre facilite la manipulation et l'évaluation des *score expressions*.

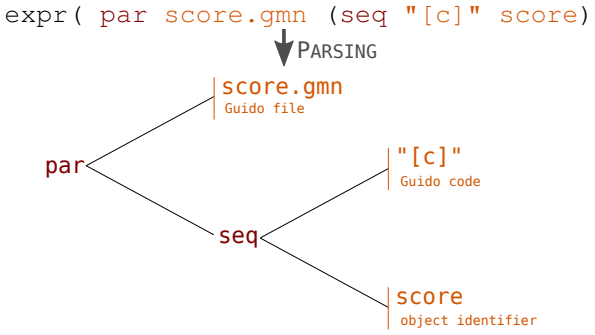


Figure 1. Représentation d'une expression sous forme d'arbre.

La représentation sous forme d'arbre est strictement similaire à la version textuelle d'une expression. Le typage des arguments constitue la seule différence : alors que le type des arguments (GMN code, fichier, identificateur ou expression) est implicite dans l'expression textuelle, celui-ci devient explicite dans la représentation mémoire.

La représentation sous forme d'arbre est stockée avec l'objet correspondant, prête à être évaluée.

3.2. Le processus d'évaluation

Le processus d'évaluation traverse l'arbre d'une expression en utilisant un algorithme de *depth first post-order traversal*, en réduisant chaque noeud à du code GMN (Figure 2). L'évaluation d'un noeud dépend de son type :

— un fichier GMN renvoie son contenu,
— une chaîne GMN renvoie la chaîne,
— un fichier MusicXML renvoie son contenu converti au format GMN,
— une chaîne MusicXML renvoie la chaîne convertie au format GMN,
— l'identifiant d'object renvoie son code GMN,
— un opérateur renvoie l'application de l'opérateur à ses arguments.

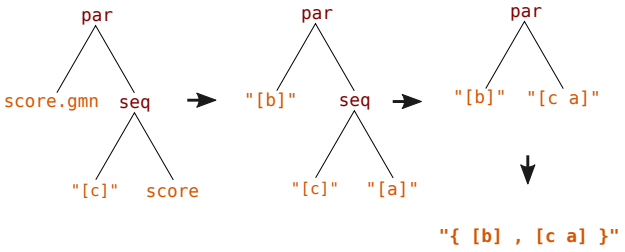


Figure 2. Evaluation d'une expression, où `score` est défini comme `[a]` et `score.gmn` contient `[b]`.

Ce schéma d'évaluation permet d'éviter les problèmes de récursion (e.g. un objet qui se modifie lui même en utilisant une expression basée sur son propre contenu) puisque l'objet appelant n'est modifié qu'à la fin du processus d'évaluation. Une expression est référentiellement transparente : par défaut, chaque argument n'est évalué qu'une fois et sa valeur est alors considérée comme constante.

3.3. Evaluation dynamique

Dans notre cas, la transparence référentielle (qui permet la mémoïsation) peut être une limitation. Par exemple, pour une expression utilisant un stream GMN, on pourrait vouloir maintenir le résultat de l'évaluation de l'expression à jour de l'état actuel du stream. Pour cela, des *arguments variables* ont été introduits en les préfixant du caractère `&` : un argument variable est toujours évalué, quelques soit ses valeurs précédentes (Figure 3). Seuls les arguments susceptibles de modification (les fichiers ou les objets INScore) peuvent être déclarés *variables*.

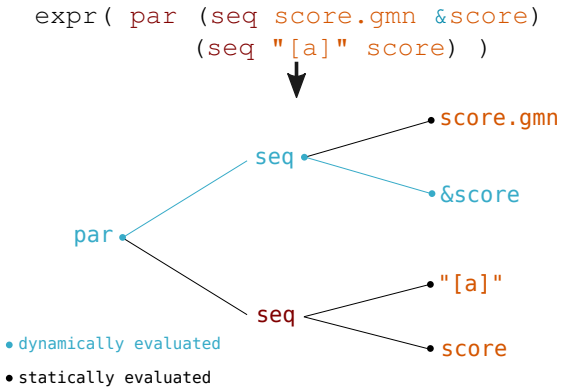


Figure 3. Propagation d'une évaluation dynamique. `&score` est mis à jour avec la valeur courante de `score` à chaque évaluation, tandis que `score` conserve la valeur calculée à la première évaluation. Ainsi, lorsqu'on réévalue l'expression, la deuxième opération `seq` ne sera pas recalculée.

Un arbre qui contient un argument variable est un *arbre dynamique*. Quand un argument variable est rencontré sur une branche, l'attribut *dynamique* est propagé jusqu'à la racine de l'arbre. Après la première évaluation, seules les parties dynamique d'un arbre sont recalculées. Pour des raisons d'optimisation, le système vérifie que la valeur d'un argument a changé et ne recalcule la branche correspondant que si nécessaire.

L'usage d'arguments variables permet de décrire des expressions comportant des parties variables arbitraires : on peut le voir comme la composition d'une partition symbolique par agrégation de parties statiques et variables.

4. L'API DES EXPRESSIONS DANS INSCORE

Pour une intégration complète des *score expressions*, l'implémentation s'appuie sur les fonctionnalités existantes de INScore. De ce fait, les *score expressions* supportent des URLs comme arguments pour les fichiers, ainsi que les événements d'interaction. La typologie des événements d'interaction a été étendue notamment pour servir les besoins de l'évaluation dynamique (voir section 4.3).

4.1. Déclaration d'une expression

Les objets `gmn` et `pianoroll` peuvent être définis par des *score expressions* via une extension du message `set`. L'évaluation de l'expression est déclenchée lorsque l'objet cible traite le message `set`.

```
/ITL/scene/score set gmn expr(score.gmn);
/ITL/scene/pr set pianoroll expr(&score);
```

L'exemple précédent crée 2 objets : `score` qui est une représentation symbolique du fichier GMN `score.gmn`, et `pr` qui est une représentation en piano roll de `score` (évaluée dynamiquement en raison du préfixe `&`).

4.2. Messages spécifiques des expressions

Les objets définis par des *score expressions* acceptent des messages spécifiques :

— `reeval` : déclenche la re-évaluation de l'expression. Seules les parties dynamiques sont calculées.

— `renew` : déclenche la re-évaluation complète de l'expression, y compris les parties statiques.

Ces messages sont une extension du message `expr` :

```
/ITL/scene/score expr reeval;
/ITL/scene/score expr renew;
```

Enfin, l'expression qui définit un objet peut être interrogée avec le message `get expr` :

```
/ITL/scene/score get expr;
```

4.3. Extension de la typologie des événements

Les fonctionnalités d'interaction de INScore sont basées sur l'association d'événements à des listes de messages OSC arbitraires [10]. Ces messages sont émis à l'occurence de l'événement correspondant (e.g. un click). La typologie des événements a été étendue avec un nouvel événement : `newData`, qui est déclenché par un objet lorsque sa valeur change, soit en raison de la réception d'un message `set` ou `reeval`, soit parce que des données ont été écrites dans un objet définit par un stream.

L'association du message `expr reeval` à l'événement `newData` permet la réévaluation automatique d'une expression lorsque la valeur d'un objet change :

```
/ITL/scene/score set gmn "[a]";
/ITL/scene/copy set gmn expr(&score);
/ITL/scene/score watch newData
    (/ITL/scene/copy expr reeval);
```

Dans l'exemple ci-dessus, le changement de valeur de `score` changera automatiquement la valeur de `copy`.

Pour gérer les problèmes de boucle infinies, les messages associés à `newData` sont postés pour la prochaine tâche du temps d'INScore. De ce fait, la mise à jour complète d'une partition après changement de la valeur d'un de ses composants peut s'étendre sur plusieurs tâches du temps (si un objet fait référence à un autre objet, lui-même faisant référence à un autre...) et durant ce processus, la partition peut passer par des états transitoires. De cette manière, les objets faisant référence à eux-même et qui installent un mécanisme de réévaluation automatique ne seront pas bloquants pour le système.

5. COMPOSITION D'EXPRESSIONS

Alors que le langage des expressions permet de composer des partitions symboliques, il permet également de composer les expressions avec le préfixe `~`. En effet, alors que les arguments `score` et `&score` font référence à la valeur de l'objet `score`, `~score` fait référence à l'expression qui définit l'objet `score`. Dans la pratique, et avant la première évaluation, tous les arguments préfixés

par ~ sont remplacés par une copie de l'arbre d'expression de l'objet correspondant. Il est ainsi possible de réutiliser des *score expressions* déjà définies pour construire des expressions plus complexes.

La figure 4 illustre l'expansion de l'arbre d'expression pour l'exemple ci-dessous.

```
/ITL/scene/score set gmn
  expr(seq "[a]" &sample);
/ITL/scene/score set gmn
  expr(seq (seq ~score "[b]") ~score);
```

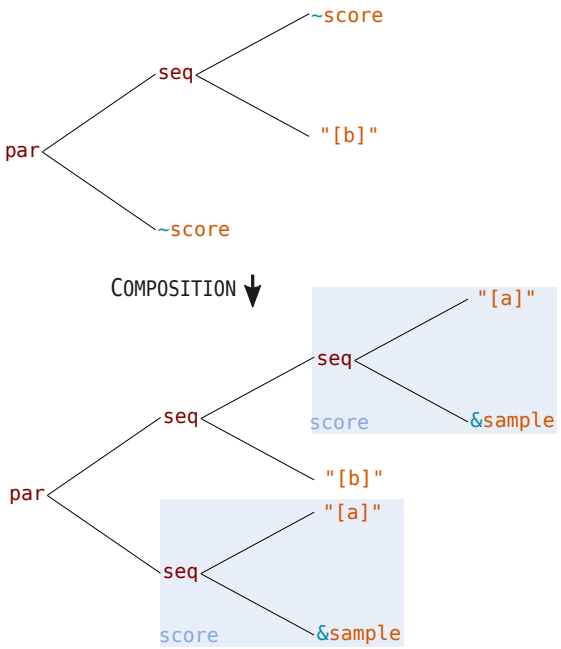


Figure 4. Composition d'expressions

6. EXEMPLES

6.1. Canon

La forme *canon* constitue une structure musicale simple qui peut être facilement décrite par des *score expressions*.

Dans l'exemple ci-dessous, la première ligne crée un objet `score` basé sur un fichier GMN. Il est ensuite transposé à la quinte et une seconde voix est ajoutée avec un délai d'une blanche. Du fait que la transposition à un intervalle spécifique n'est pas une opération de base des opérateurs (l'intervalle de transposition est calculé à partir des 2 partitions), nous combinons `transpose` avec `seq` et `evtail` pour préfixer la partition avec une note qui détermine l'intervalle de transposition, et qui est ensuite supprimée.

```
/ITL/scene/score set gmnf score.gmn;

# Transposition de la partition
/ITL/scene/canon set gmn
  expr(evtail
    (transpose (seq "[c]" score) "[g]")
    "[a]"
  );

# Mise en séquence /ITL/scene/canon set
gmn
  expr(seq score canon);

# Ajout de la second voix décalée d'une
blanche
/ITL/scene/canon set gmn
  expr(par canon (seq "[_/2]" canon));
```

Le résultat est un illustré par la figure 5

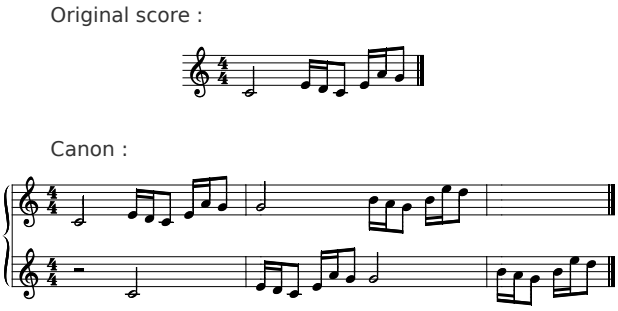


Figure 5. Canon

6.2. Partitions multiformes synchronisées

Les *score expressions* permettent de dupliquer et de transformer dynamiquement des partitions en les maintenant synchronisées avec l'original.

```
/ITL/scene/stream set gmnstream
  '[\meter<"4/4">]';

/ITL/saxo/score set gmn
  expr( evtail
    (transpose
      (seq
        "[e&1]"
        &/ITL/scene/stream )
      "[c2]" )
    "[a]"
  );

/ITL/audience/score set pianoroll
  expr( &/ITL/scene/stream);

/ITL/scene/stream watch newData
  (/ITL/*/score expr reeval);
```

L'exemple ci-dessus crée 2 copies d'un objet GMN stream. La première est transposée pour le saxophone et la se-

conde est affichée sous forme de piano roll, en tant que support visuel pour le public. Chaque partition est créée dans des scènes distinctes. La dernière ligne assure la mise à jour des copies lorsque l'objet `stream` est modifié. Le résultat est présenté par la figure 6.

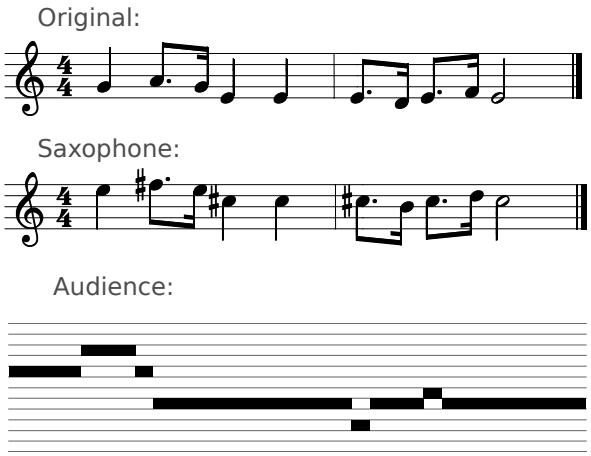


Figure 6. Des partitions multi-formes

6.3. Composer des éléments dynamiques et statiques

Cet exemple illustre la composition et la transformation de partitions dynamiques et statiques en temps-réel. Dans un premier temps, nous créons une partition sous forme de flux (nommée `stream`), destinée à être écrite en temps réel, et une partition statique (nommée `static`).

```
/ITL/scene/stream set gmnstream
    '[\meter<"4/4">]';
/ITL/scene/static set gmn
    '[\meter<"4/4"> g e f a f d c/2]';
```

Dans un deuxième temps, les deux dernières mesures de `stream` sont extraites et affectées à un nouvel objet nommé `tail`. Du fait que l'opération 'tail' coupe le début de la partition sur la durée de la partition donnée en second argument, la durée de cet argument est exprimée comme la fin de `stream` après la durée désirée (2 rondes) L'expression de `tail` fait usage d'une référence à `stream` de manière à être mis à jour de ses nouvelles valeurs.

```
/ITL/scene/tail set gmn
    expr(tail &stream
         (tail &stream '[a*2]')));
```

Le résultat final est obtenu avec les opérations 'par' et 'transpose'. Il inclut une référence à `tail` mais l'objet `static` est embarqué statiquement. `tail` est utilisé comme un objet intermédiaire pour des raisons d'optimisation et facilite par ailleurs la lecture de l'expression. Il peut être caché du rendu final sans affecter le résultat.

```
/ITL/scene/score set gmn
    expr(par &tail
         (transpose static &tail));
```

L'activation du calcul dynamique de la partition fait usage de l'événement `newData`, surveillé par l'objet `stream`, qui informe `tail` et `score` que leurs expressions doivent être re-évaluées.

```
/ITL/scene/stream watch newData
    (/ITL/scene/part  expr reeval,
     /ITL/scene/score expr reeval
    );
```

7. CONCLUSIONS

En combinant les opérateurs Guido avec les fonctionnalités d'INScore (API OSC, dimension Web, interactivité...), les *score expressions* intègrent pleinement la composition de partitions symboliques dans l'environnement interactif des partitions augmentées. Elles étendent le processus d'écriture à la structure musicale et à l'agrégation de partitions par la combinaison de différents matériaux symbolique, incluant d'autres objets de la partition. Elles constituent un jeu d'outils simples pour manipuler des partitions quelle que soit leur origine (fichiers, URLs, streams) ou quelle que soit leur représentation (symbolique, piano roll), et pour concevoir des partitions dynamiques basées sur des compositions arbitraires.

Dans le futur, nous allons étudier l'extension des *score expressions* à tous les objets d'INScore. Une telle approche - la composition de ressources graphiques arbitraires avec une sémantique musicale - pose des problèmes non triviaux à résoudre. En effet, si les opérations dans le domaine temporel peuvent s'appliquer à n'importe quel objet en raison de leur dimension temporelle commune, les transformations dans le domaine des hauteurs ou dans le domaine du temps structuré (tel que le rythme) impliquent l'extension de la sémantique de l'espace graphique.

8. REFERENCES

[1] E. Ellberger, G. Toro-Perez, J. Schuett, L. Cavaliero, and G. Zoia, "A paradigm for scoring spatialization notation," in *Proceedings of the First International Conference on Technologies for Music Notation and Representation - TENOR2015* (M. Battier, J. Bresson, P. Couprie, C. Davy-Rigaux, D. Fober, Y. Geslin, H. Genevois, F. Picard, and A. Tacaille, eds.), (Paris, France), pp. 98–102, Institut de Recherche en Musicologie, 2015.

[2] T. Mays and F. Faber, "A notation system for the karlax controller," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, (London, United Kingdom), pp. 553–556, Goldsmiths, University of London, June 2014.

[3] W. Enstr"om, J. Dennis, B. Lynch, and K. Schlei, "Musical notation for multi-touch interfaces," in *Proceedings of the International Conference on New Interfaces for Musical Expression* (E. Berdahl and J. Allison, eds.), (Baton Rouge, Louisiana, USA), pp. 83–86, Louisiana State University, May 31 – June 3 2015.

[4] R. R. Smith, "An atomic approach to animated music notation," in *Proceedings of the First International Conference on Technologies for Music Notation and Representation - TENOR2015* (M. Battier, J. Bresson, P. Couprie, C. Davy-Rigaux, D. Fober, Y. Geslin, H. Genevois, F. Picard, and A. Tacaille, eds.), (Paris, France), pp. 39–47, Institut de Recherche en Musicologie, 2015.

[5] C. Hope, L. Vickery, A. Wyatt, and S. James, "The decibel scoreplayer - a digital tool for reading graphic notation," in *Proceedings of the First International Conference on Technologies for Music Notation and Representation - TENOR2015* (M. Battier, J. Bresson, P. Couprie, C. Davy-Rigaux, D. Fober, Y. Geslin, H. Genevois, F. Picard, and A. Tacaille, eds.), (Paris, France), pp. 58–69, Institut de Recherche en Musicologie, 2015.

[6] R. Hoadley, "Calder's violin : Real-time notation and performance through musically expressive algorithms," in *Proceedings of International Computer Music Conference* (ICMA, ed.), pp. 188–193, 2012.

[7] R. Hoadley, "December variation (on a theme by earle brown)," in *Proceedings of the ICMC/SMC 2014*, pp. 115–120, 2014.

[8] D. Fober, Y. Orlarey, and S. Letz, "Inscore – an environment for the design of live music scores," in *Proceedings of the Linux Audio Conference – LAC 2012*, pp. 47–54, 2012.

[9] D. Fober, Y. Orlarey, and S. Letz, "Augmented interactive scores for music creation," in *Proceedings of Korean Electro-Acoustic Music Society's 2014 Annual Conference [KEAMSAC2014]*, pp. 85–91, 2014.

[10] D. Fober, S. Letz, Y. Orlarey, and F. Bevilacqua, "Programming interactive music scores with inscore," in *Proceedings of the Sound and Music Computing conference – SMC'13*, pp. 185–190, 2013.

[11] D. Fober, Y. Orlarey, and S. Letz, "Representation of musical computer processes," in *Proceedings of the ICMC/SMC 2014*, pp. 1604–1609, 2014.

[12] A. Agostini and D. Ghisi, "Bach : An environment for computer-aided composition in max," in *Proceedings of International Computer Music Conference* (ICMA, ed.), pp. 373–378, 2012.

[13] N. Didkovsky and G. Hajdu, "Maxscore : Music notation in max/msp," in *Proceedings of International Computer Music Conference* (ICMA, ed.), 2008.

[14] P. Hudak, T. Makucevich, S. Gadde, and B. Whong, "Haskore music notation – an algebra of music," *Journal of Functional Programming*, vol. 6, pp. 465–483, May 1996.

[15] S. W. Lee and J. Freeman, "Real-time music notation in mixed laptop-acoustic ensembles," *Computer Music Journal*, vol. 37, pp. 24–36, Dec. 2013.

[16] D. Fober, Y. Orlarey, and S. Letz, "Scores level composition based on the guido music notation," in *Proceedings of the International Computer Music Conference* (ICMA, ed.), pp. 383–386, 2012.

[17] J. Bresson and J.-L. Giavitto, "A Reactive Extension of the OpenMusic Visual Programming Language," *Journal of Visual Languages and Computing*, vol. 25, no. 4, pp. 363–375, 2014.

[18] H. Hoos, K. Hamel, K. Renz, and J. Kilian, "The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music.," in *Proceedings of the International Computer Music Conference*, pp. 451–454, ICMA, 1998.

[19] J. Bresson, C. Agon, and G. Assayag, "OpenMusic – Visual Programming Environment for Music Composition, Analysis and Research," in *ACM MultiMedia (MM'11)*, (Scottsdale, United States), 2011.