Représentation et rendu de structures rythmiques

Carlos Agon, Karim Haddad, Gérard Assayag
Équipe Représentation Musicale, IRCAM
Email : {agonc, haddad, assayag}@ircam.fr

Résumé

Nous proposons un format de représentation ainsi qu'un ensemble d'outils pour la notation musicale sur ordinateur. Cette proposition ne se veut pas universelle, elle offre plutôt une approche efficace au problème de notation musicale. Ce format pourrait contribuer à l'implémentation d'éditeurs musicaux dans divers environnements existants. Nous essayerons de fournir un cadre de travail général qui s'adapterait à tout type de langage de programmation. Ce protocole fut implémenté dans l'environnement OpenMusic qui utilise le langage CLOS. Ce papier est une généralisation a posteriori de cette implémentation.

1. Introduction

Les standards pour la représentation et la transmission de la notation musicale dans le domaine informatique continuent à susciter beaucoup d'intérêt de nos jours. Parmi les standards existants citons : [Moun96], [Gran95], [sch97], [HHRK98]. Malgré les divers efforts, nul standard ne s'est imposé (au même titre que le standard MIDI). Ceci peut s'expliquer par la diversité dans la finalité de leur utilisation : visualisation, gravure, édition et saisie, etc. Suite à la destination de ces protocoles, qui pour la plupart tendent vers une finalité éditoriale, on constate l'amalgame de deux problématiques qui rendent le contexte plus complexe et plus lourd à gérer. D'une manière plus générale, on observe un dualisme entre la représentation du contenu musical et celle de sa destination finale qui est la mise en page. Bien évidemment la plupart des standards se situent entre ces deux extrêmes, presque tous ces formats mélangent de manière indistincte des informations musicales et des informations de formatage. Enfin, ce qui nous a paru faisant le plus défaut dans la plupart des descriptions est le manque d'information hiérarchique du contenu lui-même. Très souvent les formats existants sont axés essentiellement autour de la description syntaxique et ne rendent pas compte d'une information fondamentale de la transmission de la musique, à savoir, la sémantique.

Loin de proposer un standard universel, nous nous sommes orientés vers ce qu'il conviendrait d'appeler la représentation de la notation musicale symbolique. L'idée principale est offrir une solution efficace aux applications dont la finalité est la manipulation et la construction des structures musicales. Notre approche vise à éliminer la prise en considération du formatage final concrétisé par la mise en page en vue de l'édition. Cette stratégie nécessite une méthodologie pour le rendu qui comblera le manque d'information concernant la pagination. De plus, dans la plupart des systèmes existants, les algorithmes de rendu ne sont pas publics. C'est pour les raisons suivantes que nous avons divisé le problème en deux parties complémentaires : premièrement, définition d'une représentation symbolique de structures rythmiques ; deuxièmement, mise à disposition d'un ensemble d'algorithmes pour le passage des représentations vers la notation sous forme de partition.

La représentation symbolique repose essentiellement sur une description hiérarchique du contenu musical comme on le verra plus loin. Ceci permet une lisibilité de la structure hiérarchique prise dans son ensemble, ce qui résulte dans un format manipulable symboliquement (inversion, récursivité rétrograde, etc.) car le format est l'abstraction de l'objet rythmique même. Les algorithmes de rendu sont cependant indépendants du format de description. Leur intégration et leur implémentation auprès de divers environnements existants ou à venir sont possibles. Ceci nous permet à la fois d'intégrer des additions futures et d'ouvrir une dynamique pouvant élargir le champ descriptif du protocole même.

Dans la section 2, nous aborderons la définition des structures de représentation que nous appellerons arbres rythmiques (**RTs**). Celle-ci sera illustrée par quelques exemples musicaux. La section 3 sera dédiée à l'algorithmique qui permettra le passage des **RT**s vers leur représentation graphique. En guise de conclusion, nous aborderons nos projets futurs.

2. Arbres rythmiques

Le format des arbres rythmiques (**RT**) devrait être compris comme une alternative pour la description de structures rythmiques symboliques permettant une parfaite cohésion avec la notation musicale de la plus traditionnelle à la plus complexe.

Ce format est issu des différentes tentatives de description rythmique faites dans les logiciels Patchwork et OpenMusic [ARLA99].

2.1. La syntaxe

Un **RT** est défini comme un couple (**D S**) où **D** est une fraction (> 0) et **S** est une liste de n-éléments définissant n-proportions de **D**. Chaque élément de **S** peut être soit un entier, soit un **RT**. Voici quelques **RT** qui correspondent à la syntaxe définie ci-haut :

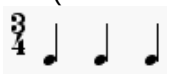
```
(1 (1 1 1 1))
(2 ((1 (1 1 1 1)) (1 (1 1 1 1)))
```

2.2. La sémantique

Pour un **RT** = (**D S**), **D** exprime un champs temporel (une durée). **S** définit un groupe de proportions ayant lieu dans **D**. Par exemple, pour **RT** = (1 (1 1 1 1)) en prenant comme unité la noire, le rythme représenté sera le suivant :

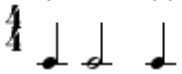
Les **RT**s nous permettent d'exprimer, d'une façon homogène, divers types d'objet musicaux. Polyphonies, voix, mesures, groupes[1] sont exprimés comme **RT**s.
Quand la valeur de **D** est au niveau de l'objet mesure, par convention, il est exprimé en unités de ronde[2]. Par exemple, si l'on veut exprimer une mesure de "3/4 " contenant trois noires , le **RT** sera :

```
(3/4  (1  1  1))
```



Comme il a été défini précédemment, **S** représente un ensemble de proportions ayant lieu dans **D**. Ceci est illustré dans l'exemple suivant :

```
(4/4(1 2  1))
```

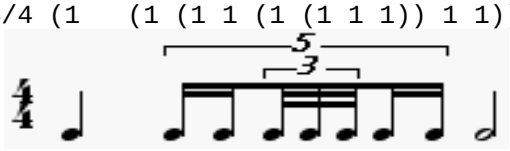


Jusqu'à présent nous avons examiné des **RT** où **S** était sous la forme d'éléments simples (des entiers), qui représentaient des notes (pulsations). Dans l'arbre qui suit l'on verra apparaître des éléments plus complexes :

```
(4/4 (1  (2 (1 1 1)) (1 (1 1 1))))
```

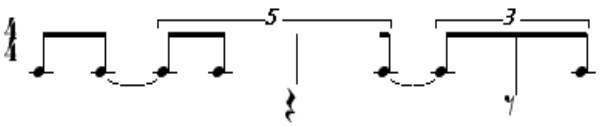
Les **RT**s contenus récursivement dans **S** représentent ce que l'on appellera des groupes. En général, les groupes sont représentés symboliquement par des notes groupées au niveaux des hampes. De même que les mesures, les groupes peuvent aussi à leur tour contenir soit des notes soit d'autres groupes, voici un exemple :

```
(4/4 (1   (1 (1 1 (1 (1 1 1)) 1 1))  2))
```



Nous étendrons la syntaxe des **RT**s afin d'intégrer les silences et les liaisons rythmiques qui seront représentés respectivement par des entiers négatifs et des nombres flottants.

```
(1 ((4/4  ((1 (1  1))(2(1.0 1  -2    1))(1(1.0 -1   1))))))
```

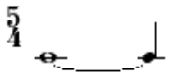


[1] Le concept de groupe est présenté plus loin.
[2] Toutefois dans d'autres implémentations, s'il est nécessaire cette valeur pourrait être redéfinie, dans le cas par exemple d'une utilisation de mensurations de musique ancienne où la valeur de référence est la longue.

Un **RT** rend compte non seulement d'un rythme, mais aussi de la façon dont il est représenté. Ainsi pour les deux rythmes affichés dans la figure suivante les deux **RT**s associés seront différents :

```
(2/4(2 2  2 2))      (2/4((1(1 1))(1(1 1)))
```

Dans certains cas, nous pourrons avoir des **RT**s syntaxiquement corrects qui n'ont pas leur équivalent dans la notation. Prenons par exemple le **RT** (5/4 (5)) qui correspond au rythme suivant :



et dont le **RT** associé est (5/4 (4 1.0)).
Nous avons donc besoin d'un ensemble de règles de réécriture qui transforment ces **RT**s avant leur visualisation. Voici deux exemples de réécriture :

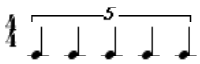
- Transformation avec liaisons

Dans l'exemple précédent nous avons remplacé la valeur (5) par (4 1.0). L'algorithme de cette transformation est le suivant :

```
add-liaisons
n un entier susceptible d'être transformé
-----------------------------------------------------------------------
pi = première puissance de 2 inférieur au égal à n
si pi = n or pi*1.5 = n or pi*1.75 = n
        return n
sinon si n > (* bef 1.5))      return [pi + pi/2 , add-liaisons(  n - (pi*5))  ]
sinon
        return [pi , add-liaisons(  n - pi) ]
```

- Groupement

Prenons par exemple le RT (4/4 (1 1 1 1 1)) puisque la mesure est divisée par une valeur irrationnelle[3] (5 : 4), nous devons donc transformer le RT en (4/4 ((4 (1 1 1 1 1))) qui sera affiché ainsi :



L'algorithme de cette transformation est le suivant:

```
create-irrationel-groupe
Un RT pour une mesure de la forme (signature, répartition)
-----------------------------------------------------------------------
num = numerator (signature)total-subdiv  = Addition des éléments en
répartition
ratio = total-subdiv / num
si ratio est un entier ou une puissance de 2
        return     (signature, répartition)
sinon si (subdivs est une puissance de 2) et
         (num est puissance de 2 or  (num mod 3 = 0 et ratio est un entier))
        return  (signature, repartition)
sinon si num / total-subdiv n'est pas un entier
        return (signature , (num , répartition))
sinon   return (signature, répartition)
```

[3] Nous employons ici le terme de valeur irrationnelle dans un contexte musical, à savoir toute valeur qui ne subdivise pas le temps d'une manière binaire.

3. Des arbres à la partition

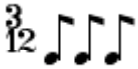
La base de notre système de rendu est la mesure. S'il est possible de générer une mesure, générer une voix consisterait à générer des mesures l'une à la suite de l'autre. Une polyphonie consiste à la génération de voix superposées les unes aux autres en suivant l'algorithme d'espacement. Ainsi, nous allons décrire dans ce qui suit les algorithmes pour l'écriture des mesures et ceux concernant l'espacement.

3.1 Interprétation des mesures

Pour illustrer les algorithmes, nous utiliserons comme exemple la mesure suivante :

`(3/4(1 1 1))`

Nous appelons unité symbolique **us** la figure de notation définie par le chiffrage d'une mesure. Pour notre mesure à 3/4 l'unité symbolique est la noire. Elle est déduite en prenant le rapport entre 1 et le dénominateur du chiffrage (i.e. 1/4). Cependant cela n'est qu'un cas particulier. Prenons par exemple l'arbre rythmique suivant : `((3/12 (1 1 1))` qui correspond à la mesure suivante :



Ici l'unité symbolique est la croche de noire 1/8 et non la croche de triolet 1/12. Nous définissons l'unité symbolique comme le ratio entre 1 et une approximation symbolique du dénominateur du chiffrage. Cette approximation est donnée à l'aide de l'algorithme suivant :

```
Approx-symbolique
n est un entier, 1 pour la ronde 4 pour la noire, 8 pour la croche, etc.

si n = 1 return 1
sinon si n appartient à {2, 3} return 2
sinon si n appartient à {4, 5, 6, 7} return 4
sinon si n appartient à {8, 9, 10, 11, 12, 13, 14} return 8
sinon si n appartient à {15, 16} return 16
otherwise       pi = première puissance de 2 inférieur au égal à n
                ps = première puissance de 2 supérieure à n
                si |n - pi| > |n - ps| return ps
                sinon return pi
```

Une fois calculée l'unité symbolique, nous déterminons les figures qui correspondent à chaque élément **s** de **S** de l'arbre rythmique (**D S**). Rappelons que **s** peut être un entier (figure simple) ou un **RT** (groupe).

- Figure simple

Pour simplifier, nous pouvons définir une figure par[4] :

• une tête de note appartenant à l'ensemble {*square*, *whole*, *half*, *quarter*} ;

• un nombre de hampes et un nombre de points.

Pour déterminer une figure **f** dans une mesure **m = (D S)**, nous procédons d'abord par la définition de la durée symbolique **ds** comme suit :

```
ds (f,m) = (f * numerator de D) /  ( 1/us (m) *  Addition des éléments en S)
```

Dans notre exemple **ds** (3) = (3*3) / (4 * 6) = 3. Une fois **ds** calculée pour une figure donnée, nous pouvons obtenir alors la tête de note, le nombre de hampes et le nombre de points qui lui correspond à l'aide de l'algorithme suivant :

[4] Les silences étant du même ordre que l'expression des notes, le problème ne sera pas soulevé dans ce papier.

```
tête-hampes-points
n  une durée symbolique,  noire = 1; 2 = blanche ; etc.

num = numerator (n)
denom = denominator (n)
pi = première puissance de 2 inférieure ou égale à n
si      pi = numerator
        { head = get-head (n)
         points = 0
         beams = get-beams (n) }
sino si         ( pi * 1.5) = numerator
        { head = get-head (pi / denominator)
          points = 1
          beams = get-beams (pi / denominator)}
sino si ( pi * 1.75) = numerator
        { head = get-head (pi / denominator)
          points = 2
          beams = get-beams (pi / denominator)}
return [head , points , beams]
```

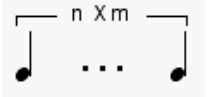
Par convention, les notes peuvent au maximum être doublement pointées. Les notes triplement pointées auront une représentation alternative calculée par l'algorithme de réécriture défini dans la section 1. Les fonctions *get-head* et *get-beams* sont définies par :

```
get-head
n  un ratio 1 = ronde, 1 /4 noire, etc.

si n > 2 return square
sinon si n = 1 return whole
sinon si n = 1/2 return half
sinon return quarter
```

```
get-beams
n  un ratio 1 = ronde, 1 /4 noire, etc.

si n > 2 return 0
sinon si n = 1/4 ou n = 1/2 n = 1 return 0
sinon return log2 denominator(n)  - 2
```

- Groupes

Les groupes étant représentés en tant que **RT**s, nous pourrons par conséquent utiliser la même procédure que celle donnée pour les mesures. On devrait donc trouver l'information symbolique pour chaque élément dans la partie **S** du groupe représenté par (**D S**). La seule chose à tenir en compte serait la propagation de l'unité rythmique de la mesure au niveau du groupe. De plus, nous devrons déterminer pour les groupes ce qu'on appelle la subdivision du groupe. La subdivision d'un groupe est représentée graphiquement dans la figure suivante :



qui se lit ainsi : " il y a n figures là où il devrait y avoir m ". Certains groupes n'affichent pas cette information ou l'affichent partiellement. Par exemple :

Il y a 4 double croches là où il devrait y avoir 4 double croches. La subdivision n'est pas affichée car n/m = 1. Autre exemple :

Il y a 3 double croches là où il devrait y avoir 2 double croches. On affiche uniquement le 3 car 2 est une puissance de 2. ‹Voici l'algorithme pour déterminer la subdivision d'un groupe.

```
get-group-subdivision
G un groupe de la forme (D S)

ds = duree symbolique de G
subdiv = Adition des elements en S

n =     si subdiv = 1 return ds
        sinon si       ds/subdiv est un entiere and
                       (ds/subdiv est une puissance de 2 or
                        subdiv/ds est une puissance de 2)
                       return ds
        sinon return subdiv
m =     si n est binaire
              return Approx-symbolique (n)
        sinon si ratio est ternaire
              return Approx-symbolique (n)* 3/2
        sinon   { num = numerator de n
                  si (num + 1) = ds return ds
                  sinon si num = ds return num
                  sinon si num < ds return [n = num*2, m = ds]
                  sinon si num < ((ds * 2) - 1) return ds
                  sinon {pi = première puissance de 2 inférieur au égal à n
                         ps = première puissance de 2 supérieure à n
                         si |n - pi| > |n - ps| return ps
                         sinon return pi
                        }
                }
return [n, m]
```

3.2 Algorithme d'espacement

L'algorithme suivant nous sert pour l'affichage de polyphonies. La contrainte principale est la cohésion de l'ordre temporel entre les figures des diverses voix. Les figures ayant le même temps d'attaque doivent être à la même position horizontale dans la partition. L'idée principale est d'assigner une position horizontale pour toutes les figures (i.e. silences, accords et barres de mesures). Pour cela nous partirons d'une liste aplatie des figures de toutes les voix, qui sera ordonnée dans un ordre croissant par rapport à l'attaque de chaque figure. Nous supposons alors que pour chaque figure nous avons une fonction *x-pos* qui nous permettra de lire et d'écrire la position dans l'axe horizontal. Voici notre algorithme d'espacement :

```
space-figures
L une liste de n figures ordonnées.

current-x-pos = 0
loop for i 0 1 to n - 1
       current-x-pos = set-space-position (L[i+1], L[i], current-x-pos)
```

```
set-space-position¶
obji+1, obji, current-x-pos¶
¶
si obji+1 et obji sont des barres de mesures¶
   →   {x-pos (obji+1) = x-pos (obji)¶
   →   return current-x-pos}¶
sinon si obji est un barre de mesure¶
   →   {x-pos (obji) = current-x-pos¶
   →   return current-x-pos + space-for-signature}¶
sinon si obji est la derniere figure¶
   →   {x-pos (obji) = current-x-pos¶
   →   return current-x-pos + space-for-last barre}¶
sinon si obji+1 est un barre de measure et obji est un accord¶
   →   {x-pos (obji) = current-x-pos¶
   →   si x-pos (obji) = x-pos (obji+1)¶
             return current-x-pos + space-for-signature¶
       sinon return current-x-pos + (x-pos (obji+1) - x-pos (obji))¶
sinon si obji est un accord¶
   →   {x-pos (obji) = current-x-pos¶
   →   si x-pos (obji) = x-pos (obji+1)¶
             return current-x-pos ¶
       sinon return current-x-pos + (x-pos (obji+1) - x-pos (obji))¶
sinon si obji est le dernier élément¶
   →   {x-pos (obji) = current-x-pos ¶
return current-x-pos + final-space¶
```

4. Conclusion

Nous avons proposé dans ce papier une méthodologie simple pour la description et l'affichage des structures rythmiques. L'idée principale est de permettre aux divers types d'applications, et tout spécialement les applications de CAO, de représenter et de partager facilement leur contenu musical, au même titre que le standard MIDI, et ce au niveau de la notation musicale. Dans l'avenir une page Web sera disponible et sera consacrée à la publication de cette méthodologie. De plus, nous espérons y inclure des algorithmes supplémentaires concernant les hauteurs. Dans l'immédiat, nous attendons une discussion ouverte concernant les autres facettes de la notation (la représentation les nuances, les gestes dynamiques, pour ne citer qu'une partie), ainsi que l'énonciation et la proposition de solutions éventuelles à d'autres problèmes en rapport à la notation. Techniquement, nous sommes amenés à explorer en priorité un format permettant l'exportation des **RT**s vers d'autres implémentations. Vu le caractère récursif des **RT**s, le format XML nous semble le plus approprié. Finalement, afin d'obtenir des vrais partitions à partir de notre format, il est nécessaire d'implémenter des fonctionnalités d'exportation vers d'autres formats d'édition.

5. Références

[ARLA99] Gérard Assayag, Camilo Rueda , Mikael Laurson, Carlos Agon, O. Delerue. *Computer Assisted Composition at Ircam : PatchWork & OpenMusic*. In Computer Music Journal 23:3, to appear, fall 1999.

[Gran97] Grande, Cindy. *The Notation Interchange File Format* : A *Windows* - Compliant Approach. In: E. Selfridge-Field, ed. *Beyond MIDI* - A Handbook of Musical Codes. Massachusetts : MIT Press. , 1997.

[HHRK98] H. H. Hoos, K. A. Hamel, K. Renz, J. Kilian. *The GUIDO Music Notation Format - A Novel Approach for Adequately Representing Score-level Music*. In ICMC'98 Proceedings, p.451-454.

[Moun96] Stephen R. Mounce. A Brief Discussion of Standard Music Description Language. ISO/IEC Draft International Standard 10743, 1996.

[sch97] Bill Schottstaedt. Common Music Notation. In : Beyond MIDI, The handbook of musical codes. Eleanor Selfridge-Field ed. MIT press, Cambridge Masssachusetts, 1997.