HAL open science

Caractérisation et représentation des processus musicaux.

Dominique Fober

▶ To cite this version:

Dominique Fober. Caractérisation et représentation des processus musicaux.. 2013. hal-00861440v2

HAL Id: hal-00861440
https://hal.science/hal-00861440v2

Submitted on 24 Mar 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ANR-12-CORD-0009

INEDIT Project

Programme ContInt

http://inedit.ircam.fr/

Caractérisation et représentation des processus musicaux

Dominique Fober

GRAME, Centre National de Création Musicale

fober@grame.fr

Abstract

Ce rapport présente une étude préliminaire réalisée dans le cadre du projet ANR INEDIT. L'objectif est de définir les éléments de caractérisation des processus musicaux et la manière de les représenter, pouvant servir notamment à une représentation des processus musicaux au sein de la partition.

1 Introduction

Le développement des technologies électroniques puis numériques de captation, de codage et de production du son tout au long du XXème siècle a entrainé une révolution musicale et l'émergence de genres nouveaux solidement installés dans la culture et l'imaginaire musicaux.

De ces évolutions émergent de nouveaux besoins notamment en terme de notation et de représentation [**?**]. Soulevées par des formes artistiques comme les *musiques interactives*, ou le *live coding*, les questions de notation dynamique, de représentation de processus prennent une importance significative, liée aux enjeux correspondants pour la création musicale. Dans les musiques interactives, la cohabitation entre les écritures statiques et les processus d'interaction reste à organiser. Dans le live coding, les frontières entre programmation et partition tendent à se dissoudre [**?**]. L'objet même de la notation musicale nécessite aujourd'hui d'être élargi pour prendre en compte les nouvelles dimensions des pratiques artistiques actuelles.

Ce document traite de la représentation dynamique des processus et de leur état, le tout au sein d'un unique espace graphique, celui de la partition musicale. Le contexte musical donne une dimension particulière à la problématique de la représentation des processus puisque l'on ne s'intéresse pas seulement à un état présent, mais à un ensemble d'états qui sont ordonnés dans le temps, incluant les futurs états possibles. Les contraintes d'usage impliquent également que ces états soient lisibles en temps réel, c'est à dire dans le temps de l'exécution de la partition, et que leur lecture puisse constituer un guide pour les choix d'interaction du musicien.

Dans le cadre du projet INEDIT, l'objectif est d'intégrer la représentation des processus dans INScore [**?**], un environnement pour le design de partitions augmentées et interactives issu du projet Interlude [**?**].

Nous présenterons tout d'abord les travaux existants dans le domaine de la représentation des processus, avec un point de vue particulier sur les applications musicales. Nous définirons ensuite les éléments de caractérisation des processus d'interaction dans une approche centrée sur les exigences du domaine musical. Nous proposerons enfin un modèle de représentation de ces processus, non pas dans l'espace graphique - qui reste dans le domaine de la partition et de la responsabilité du compositeur - mais en terme de communication entre les outils qui peuvent concourir à la réalisation d'une oeuvre.

2 Approches existantes

Généralement intégrés au système d'exploitation, des outils permettent de visualiser l'état du système en terme de ressources utilisées : temps CPU, mémoire, bande passante réseau, etc. (figure 1). Dans le domaine musical, les outils pour le calcul de la musique proposent des représentations similaires : l'état du système peut inclure à la fois les ressources utilisées et l'état des paramètres du système, généralement calculés en regard des traitements audio.

Dans le domaine musical, parmi les outils fréquemment utilisés dans la conception de pièces interactives, il y a notamment PureData [**?**] et Max/MSP [**?**]. PureData fournit des outils très rustiques pour rendre compte de l'état du système (figure 2b). Max/MSP inclut un moniteur des ressources audio (figure 2a) qui donne également la valeur des paramètres audio tels que fréquence d'échantillonnage, taille des tampons d'entrée/sortie, etc.

Du point de vue des musiques interactives, la position courante dans la partition fait partie de l'état du système mais la notion de partition est absente dans des outils comme Max/MSP ou PureData. Des extensions comme Bach [**?**] ou MAXScore [**?**] permettent d'introduire la notation musicale dans Max/MSP sans toutefois prendre en compte la représentation des processus musicaux.

Dans des environnements comme Open Music [**?**] ou i-score [**?**], les processus sont vus comme des boites *opaques*. Leur dimension temporelle est prise en compte et un curseur permet de localiser la position courante sans qu'il n'y ait plus d'information sur l'activité des processus.

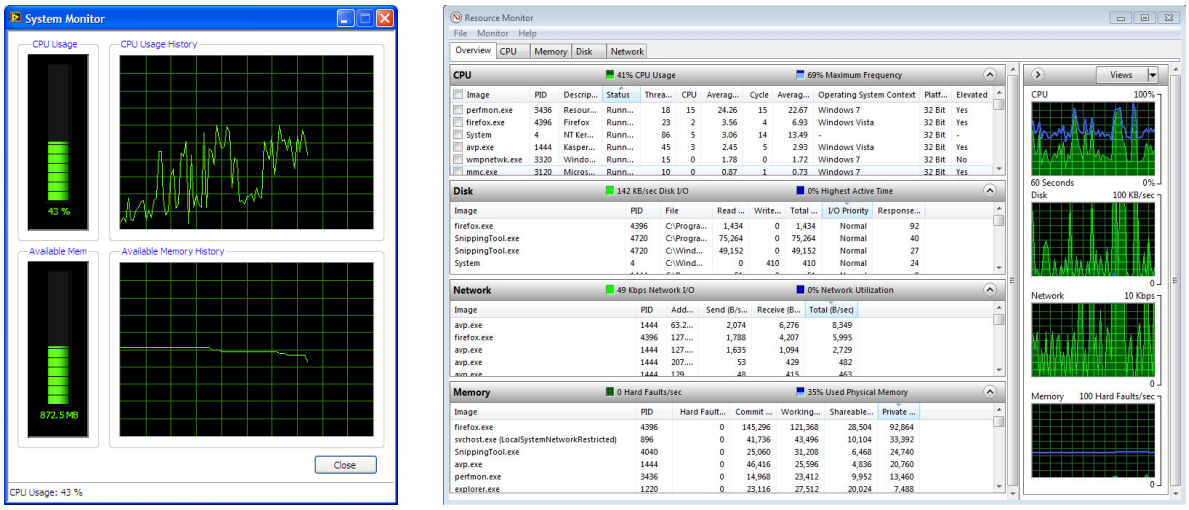
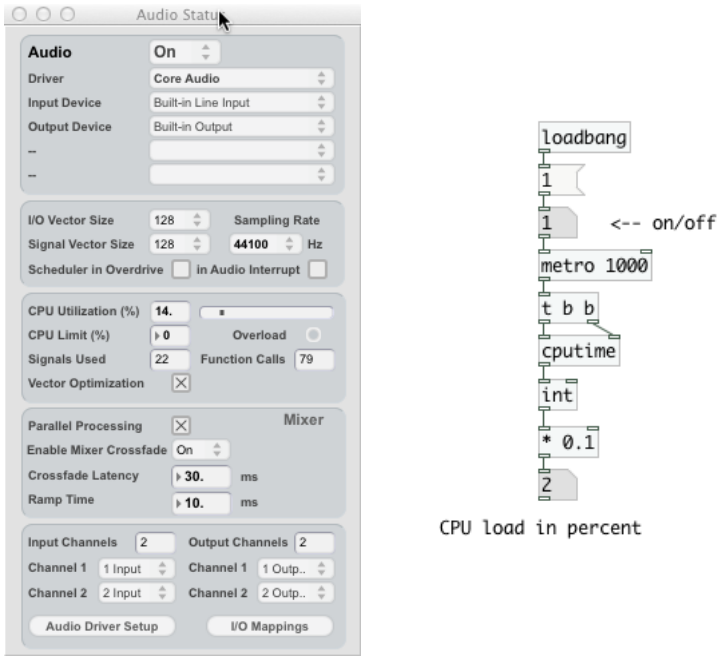


Figure 1: Visualisation de l'état du système sous Windows



a) Max/MSP b) Pure Data

Figure 2: Visualisation de l'état du système avec Max/MSP et Pure Data

C'est dans les domaines du *live coding* et des systèmes interactifs que les réflexions et approches semblent les plus avancées.

Thor Magnusson développe une réflexion et des outils qui tendent à rapprocher codage et partition musicale [**?**, **?**]. Ge Wang propose également une approche basée sur le code [**?**] comme facteur d'expression instrumentale pour le *live coding*, basée sur le langage Chuck [**?**]. Cette approche est notamment mise en oeuvre dans l'environnement de programmation audio *The Audicle* [**?**] où différentes visualisations à la fois du code et de l'activité du système sont proposées dans une approche originale et extensible (figure 3).

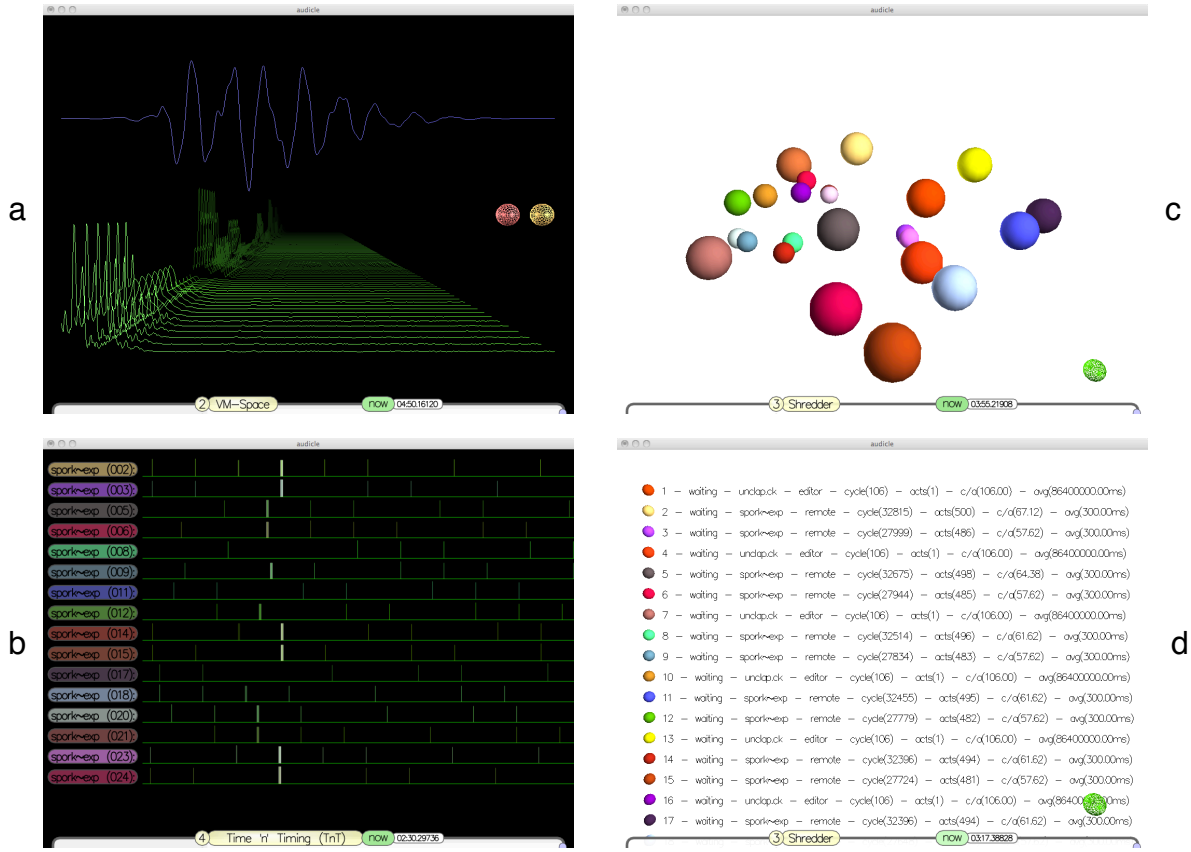


Figure 3: Visualisation de l'état du système dans Audicle : a) forme d'onde et spectre, b) information temporelle par thread, c) activité des threads sous forme graphique, d) sous forme textuelle détaillée.

Une autre approche de la visualisation est centrée sur l'auditeur avec comme objectif d'améliorer la perception par des informations graphiques dans un contexte de *live coding* [**?**] ou encore pour rendre perceptible les mécanismes d'instruments électroniques [**?**] (figure 4).

Toutes ces approches ont pour principal limitation d'être parcellaires et centrées sur les applications qui les proposent. Par ailleurs, il n'y a pas de modèle général qui en émerge et l'inter-opérabilité entre les outils n'est pas prise en compte.

3 Caractérisation des processus musicaux

Nous allons considérer les processus musicaux comme résultant de l'écriture musicale - au sens de la composition - et donc, faisant partie de la partition. De ce point de vue, un processus prend place dans le temps : il peut être

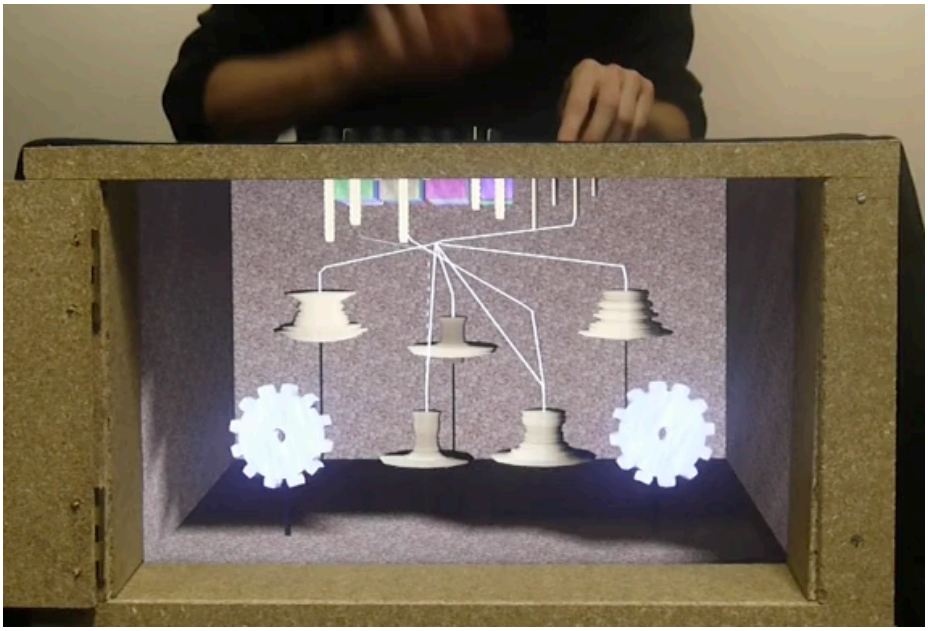


Figure 4: Visualisation des mécanismes d'instruments électroniques.

passé, présent ou futur. Nous parlerons de processus *actif* quand il s'agit d'un processus qui s'inscrit dans le présent, et de processus *inactif* pour un processus passé ou futur.

Ce statut temporel va déterminer la manière de caractériser les processus : un processus actif (ou présent) aura des propriétés différentes d'un processus inactif (i.e. passé ou futur). De même, un processus passé aura des propriétés temporelles qui pourront être différentes de celles d'un processus futur, dont la date et la durée peuvent être indéfinies, relatives à un événement externe.

Pour caractériser un processus musical, nous allons différencier trois types d'information en fonction de leur fréquence :

— un état statique : représente des informations qui ne changent pas (par exemple le processus parent) ou qui peuvent changer mais à faible fréquence (par exemple la fréquence d'échantillonnage)

— un état dynamique : représente des informations qui varient dans le temps en fonction de l'exécution du processus (par exemple le temps cpu utilisé)

— un état temporel : la date d'occurrence du processus et sa date de fin ou sa durée. Ces propriétés peuvent être indéfinies dans un contexte interactif et événementiel.

3.1 Etat statique

L'état statique d'un processus est constitué par l'ensemble des informations qui sont invariantes ou qui changent à faible fréquence. Cet état est entièrement défini pour un processus présent. Il reflète le dernier état actif pour un processus passé. Pour un processus futur, il peut indiquer le premier état à activer si celui-ci est connu.

Pour un processus générique, l'état peut inclure :

- son processus parent
- son statut (actif | inactif)
- son mode de calcul (vectorisation, parallélisation,...)

- ...

Pour un processus *audio*, l'état peut inclure :

- la taille des buffets d'entrée/sortie
- la fréquence d'échantillonnage
- les périphériques d'entrée/sortie
- le pilote utilisé
- ...

Pour un processus quelconque, l'état peut inclure la valeur des paramètres de contrôle. Par exemple pour un écho : les valeurs du délai et du feedback.

L'ensemble des propriétés statiques est dépendante du processus.

3.2 Etat dynamique

L'état dynamique d'un processus est étroitement associé à son activité et caractérise un processus présent.

Pour un processus générique, cet état peut inclure :

- le temps cpu utilisé
- le nombre de threads utilisés
- la mémoire utilisée
- ...

Le résultat du calcul du processus peut également refléter son état : par exemple les valeurs générés par un processus qui calcule un signal.

L'état dynamique peut comporter des indicateurs sur le calcul effectué : par exemple un indice de confiance pour un processus qui extrait des hauteurs MIDI d'un signal audio ou qui effectue un suivi de partition.

L'ensemble des propriétés dynamiques est dépendante du processus.

3.3 Etat temporel

Dans le cas des processus musicaux et particulièrement pour les musiques interactives [**?**], l'état temporel (i.e. la date d'occurrence du processus et sa date de fin ou sa durée), peut être partiellement ou totalement indéfini.

Dans le cas d'un processus actif, sa date de début est connue mais sa date de fin peut dépendre d'un autre processus et/ou d'un événement externe. Pour un processus qui est inscrit dans le futur, ce sont aussi bien sa date de début et/ou de fin qui peuvent être indéfinis.

La manière de représenter les dates ou les durées peut conduire à un résultat indéfini, par exemple si elles sont exprimées en temps relatif à un tempo qui lui même est indéfini.

Dans un contexte musical, ces informations temporelles sont critiques pour l'exécution des oeuvres et elles nécessitent donc d'être représentées quel que soit leur statut.

Les propriétés temporelles sont communes à tous les processus.

4 Représentation de l'état des processus musicaux

Nous proposons un format de représentation de l'état des processus musicaux qui permette la collaboration entre applications et notamment de dissocier les processus qui interviennent dans le calcul d'une oeuvre de leur représentation graphique, celle ci pouvant être vue comme intégrée à la partition musicale.

Nous définissons l'état d'un processus comme l'ensemble des valeurs qui caractérisent ce processus à un instant donné. Nous parlerons de *caractéristique* pour faire référence à une de ces valeurs. Pour représenter la connaissance que l'on a de l'état d'un processus, il faut que ce processus puisse décrire l'ensemble de ses caractéristiques.

4.1 Définition d'une caractéristique

Une caractéristique d'un processus peut être vue comme l'association d'un identifiant et d'une valeur qui peut varier dans le temps. La valeur peut être un type numérique ou encore un vecteur de valeurs : une position dans l'espace par exemple sera définie par 3 valeurs.

La valeur d'une caractéristique peut être bornée par un intervalle. Enfin, la fréquence de variation d'une caractéristique peut également être définie (figure 5).

```
caract :  ident type [ range freq ]
ident  :  string | url
freq   :  integer
type   :  'int' | 'float' | 'bool' | 'probability' | ( vector )
vector :  type | vector type
range  :  value value
value  :  int | float | bool | ( vlist )
vlist  :  value | vlist value
```

Figure 5: Description d'une caractéristique d'un processus

Le type des valeurs est parmi :

- **int** : les valeurs sont des nombres entiers
- **float** : les valeurs sont des nombres flottants
- **bool** : les valeurs doivent être interprétées comme des booléens
- **probability** : les valeurs expriment une probabilité. Par convention, elle seront exprimées par des nombres flottants dans l'intervalle [0, 1].

Les valeurs de type **vecteur** sont définies par une liste de types.

Par convention, la fréquence permet définir la vitesse de variation des valeurs : elle indique un nombre de variations par seconde. La valeur 0 sera utilisée pour les valeurs constantes. Pour les caractéristiques dont la vitesse de variation n'est pas connue, la fréquence devra être omise.

4.2 Déclaration des constituants de l'état d'un processus

L'état d'un processus est défini comme une liste de caractéristiques (figure 6). Pour que la connaissance de l'état d'un processus soit représentable, il faut que ce processus puisse communiquer la description de son état.

```
process-state :  states
states        :  caract | states caract
```

Figure 6: Description de l'état d'un processus

Le format JSON [**?**] peut être utilisé pour déclarer les caractéristiques d'un processus. La figure 7 donne l'exemple d'un processus FAUST [**?**] nommé "karplus" dont l'état comporte la valeur des paramètres de contrôle ainsi que le signal généré.

4.3 Etat d'une caractéristique

L'état d'une caractéristique sera émis sous la forme d'une paire associant son identifiant et sa valeur.

Quand une liste de valeurs sont associées à un identifiant (figure 9), elles sont interprétées en fonction de la fréquence. Lorsque la fréquence est définie, l'ensemble de ces valeurs s'inscrit dans la durée correspondante.

Le protocole OSC [**?**] peut être utilisé pour transmettre des états. Dans ce cas, l'adresse OSC peut servir d'identifiant (figure 10)

4.4 Représentation des états temporels

Relativement au contexte musical, tous les processus ont deux caractéristiques communes : leur date de début et leur durée (ou date de fin). Celles-ci peuvent être indéfinies (pour un processus dans le futur par exemple) ou partiellement définies (pour un processus présent dont on ne connait pas la date de fin).

Dans le contexte des musiques interactives, une date indéfinie correspond à un événement externe. Par exemple :

- le début ou la fin d'un processus peuvent correspondre au début ou à la fin d'une séquence d'improvisation,
- un processus peut se déclencher de manière conditionnelle, e.g. si une séquence de notes est jouée ou s'il y a du silence

De manière générale, l'ordonnancement temporel de tels processus peut se décrire en terme de relations de Allen [**?**], relativement aux événements dont ils dépendent. Nous parlerons de *date ou durée événementielle* pour faire référence à des dates ou durées indéfinies.

Les caractéristique de l'état temporel d'un processus sont décrites par la figure 11 : elles sont définies par un début et une fin ou une durée. Les valeurs de début ou de fin seront exprimées sous forme de temps ou sous forme événementielle. Les durées seront toujours exprimés en temps.

4.4.1 Représentation du temps

Le temps est exprimé soit en temps absolu, soit en temps musical (i.e. relatif à un tempo). Les unités utilisées seront :

- la milliseconde dans le cas du temps absolu,
- la ronde dans le cas du temps musical

Le temps musical sera exprimé sous forme de rationnel.

```
{
    "process": "karplus",
    "states": [
        {
            "ident": "excitation",
            "type" : "float",
            "range": [ { "min": 2 }, { "max": 512 } ]
        },
        {
            "ident": "play",
            "type" : "bool"
        },
        {
            "ident": "level",
            "type" : "float",
            "range": [ { "min": 0 }, { "max": 1 } ]
        },
        {
            "ident": "attenuation",
            "type" : "float",
            "range": [ { "min": 0 }, { "max": 1 } ]
        },
        {
            "ident": "duration",
            "type" : "float",
            "range": [ { "min": 2 }, { "max": 512 } ]
        },
        {
            "ident": "signal",
            "type" : "float",
            "range": [ { "min": -1 }, { "max": 1 } ],
            "freq" : 44100
        }
    ]
}
```

Figure 7: Description d'un processus FAUST au format JSON

```
{ "excitation" :  124 }
```

Figure 8: Valeur de l'`excitation` du processus `karplus` au format JSON

```
{ "signal" :  [ -0.2, 0.1, 0.23, -0.05, -0.01, 0.8, 0.8, 0.02, -0.5 ] }
```

Figure 9: Valeurs du `signal` du processus `karplus` au format JSON

```
/karplus/signal -0.2 0.1 0.23 -0.05 -0.01 0.8 0.8 0.02 -0.5
```

Figure 10: Valeurs du `signal` du processus `karplus` émises via OSC

```
process-temporal-state
      :  begin [ end | dur ]
begin :  time | event
end   :  time | event
dur   :  time | event
```

Figure 11: Caractéristiques de l'état temporel d'un processus

4.4.2 Représentation des événements

Pour les processus dont la date est spécifiée de manière événementielle, nous souhaitons pouvoir les représenter, au moins de manière approximative. Pour cela nous définirons une *date événementielle* comme un triplet $\mathcal{T} = (t_{left}, t, t_{right})$, associé à un indice de confiance $\mathcal{P}$ et suivi d'un label optionnel (figure 13).

$\mathcal{T}$ est tel que $t_{left} \leqslant t \leqslant t_{right}$. $\mathcal{T}$ définit un interval de réalisation $[t_{left}, t_{right}]$ ainsi qu'une date de réalisation probable t. l'indice de confiance $\mathcal{P}$ représente la probabilité de réalisation de l'événement à la date t. Il sera exprimé avec une valeur flottante entre 0 et 1.

Ce type de représentation est intermédiaire entre la description de relations de Allen et la datation des processus. A titre d'exemple, la relation suivante A m (B si (C fi D)) exprimée en terme de relations de Allen et illustrée par la figure 14 pourra s'exprimer de manière *semi-instanciée* comme suit :

```
{   "process": "A",
    "start": "0/1",
    "dur": [ "1/4", "1/2", "1/1", 0.7 ]
}
{   "process": "B",
    "start": [ "1/4", "1/2", "1/1", 0.7 ],
    "dur": "1/4"
}
{   "process": "C",
    "start": [ "1/4", "1/2", "1/1", 0.7 ],
    "dur":   [ "1/2", "3/4", "1/1", 0.7 ]
}
{   "process": "D",
    "start": [ "0/1", "1/2", "1/2", 0.8 ],
    "end":   [ "3/4", "5/4", "3/2", 0.5 ]
}
```

On notera que la date de début du processus D exprime une contrainte de durée qui devra être supérieure ou égale à une noire.

```
{ "start" :  "4/1" }
{ "dur"   :  2000 }
```

Figure 12: Caractéristiques temporelles d'un processus qui commence à la quatrième ronde et qui dure 2 secondes.

```
event      :  timeset confidence [ label ]
timeset    :  ( leftbound, expected, rightbound )
leftbound  :  time
expected   :  time
rightbound :  time
```

Figure 13: Approximation du temps événementiel.

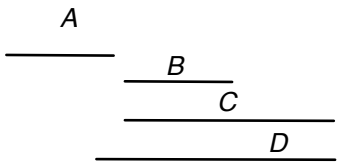


Figure 14: Relations entre 4 processus : B et C commencent avec la fin de A et D se termine avec C.

Une des possibilités de représentation peut consister à utiliser un rendu par dégradé de couleur pour rendre compte des incertitudes, comme illustré par la figure 15, qui reprend l'exemple décrit ci-dessus.

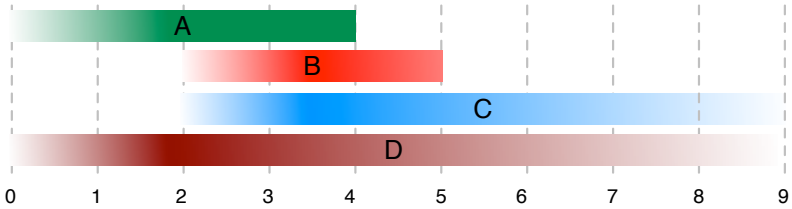


Figure 15: Une représentation des processus A, B, C, D utilisant des dégradés de couleur pour rendre compte des incertitudes.

5 Conclusion

Nous proposons une description simple de l'état des processus musicaux. Cette description est déconnectée de tout format de représentation ou protocole de communication. Toutefois des exemples en JSON et en OSC sont donnés : leur simplicité de mise en oeuvre nous a semblé en adéquation avec la description des processus musicaux proposée.

Le problème critique de la représentation du temps événementiel est traité en adoptant une approche *semi-instanciée* probabiliste. Elle évite une description générale en terme de relations de Allen, qui aurait imposé aux applications de visualisations de résoudre les contraintes correspondantes. Elle se prête à des représentations dont l'adéquation reste toutefois à vérifier.

[1] Andrea Agostini and Daniele Ghisi. Bach: An environment for computer-aided composition in max. In ICMA, editor, *Proceedings of International Computer Music Conference*, pages 373–378, 2012.

[2] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, November 1983.

[3] Antoine Allombert, Myriam Desainte-Catherine, and Gérard Assayag. Iscore: a system for writing interaction. In Sofia Tsekeridou, Adrian David Cheok, Konstantinos Giannakis, and John Karigiannis, editors, *Proceedings of the Third International Conference on Digital Interactive Media in Entertainment and Arts, DIMEA 2008, 10-12 September 2008, Athens, Greece*, volume 349 of *ACM International Conference Proceeding Series*, pages 360–367. ACM, 2008.

[4] Gérard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue. Computer-assisted composition at ircam: From patchwork to openmusic. *Comput. Music J.*, 23(3):59–72, September 1999.

[5] Florent Berthaut. Rouages: Revealing the mechanisms of digital musical instruments to the audience. In *(submitted to) Proceedings of the NIME conference*, 2013.

[6] D. Crockford. The application/json media type for javascript object notation (JSON). *RFC4627*, 2006.

[7] Nick Didkovsky and Georg Hajdu. Maxscore: Music notation in max/msp. In ICMA, editor, *Proceedings of International Computer Music Conference*, 2008.

[8] D. Fober, C. Daudin, Y. Orlarey, and S. Letz. Interlude - a framework for augmented music scores. In *Proceedings of the Sound and Music Computing conference - SMC'10*, pages 233–240, 2010.

[9] Dominique Fober, Yann Orlarey, and Stephane Letz. INScore – an environment for the design of live music scores. In *Proceedings of the Linux Audio Conference – LAC 2012*, pages 47–54, 2012.

[10] Jason Freeman. Bringing instrumental musicians into interactive music systems through notation. *Leonardo Music Journal*, 21(15-16), 2011.

[11] Thor Magnusson. The ixiquarks: Merging code and gui in one creative space. In *Proceedings of the International Computer Music Conference*, 2007.

[12] Thor Magnusson. Algorithms as scores: Coding live music. *Leonardo Music Journal*, 21:19–23, 2011.

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

[14] Alex McLean, Dave Griffiths, Nick Collins, and Geraint Wiggins. Visualisation of live code. In *Proceedings of the 2010 international conference on Electronic Visualisation and the Arts*, EVA'10, pages 26–30, Swinton, UK, UK, 2010. British Computer Society.

[15] Yann Orlarey, Dominique Fober, and Stephane Letz. *NEW COMPUTATIONAL PARADIGMS FOR COMPUTER MUSIC*, chapter FAUST : an Efficient Functional Approach to DSP Programming, pages 65–96. 2009.

[16] Miller Puckette. Combining Event and Signal Processing in the MAX Graphical Programming Environment. *Computer Music Journal*, 15(3):68–77, 1991.

[17] Miller Puckette. Pure data: another integrated computer music environment. In *Proceedings of the International Computer Music Conference*, pages 37–41, 1996.

[18] Robert Rowe. *Interactive Music Systems: Machine Listening and Composing*. MIT Press, Cambridge, MA, USA, 1992.

[19] Ge Wang and Perry R. Cook. Chuck: a concurrent, on-the-fly audio programming language. In ICMA, editor, *Proceedings of International Computer Music Conference*, pages 219–226, 2003.

[20] Ge Wang and Perry R. Cook. The audicle: a contextsensitive, on-the-fly audio programming environ/mentality. In ICMA, editor, *Proceedings of the International Computer Music Conference*, 2004.

[21] Ge Wang and Perry R. Cook. On-the-fly programming: Using code as an expressive musical instrument. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 138–143, 2004.