Representation of Musical Computer Processes

Dominique Fober, Yann Orlarey, Stéphane Letz
GRAME
Centre national de création musicale
Lyon, France
fober@grame.fr orlarey@grame.fr letz@grame.fr

ABSTRACT

The paper presents a study about the representation of musical computer processes within a music score. The idea is to provide performers with information that could be useful especially in the context of interactive music. The paper starts with a characterization of a musical computer process in order to define the *values* to be represented. Next it proposes an approach to time representation suitable to asynchronous processes representation.

1. INTRODUCTION

Throughout the last decades, the development of electronic and digital technologies, including sound production, gesture captation, has led to a musical revolution and to the emergence of new genres firmly established in the culture and musical imagination. From these evolutions, new needs have emerged, notably in terms of music notation and representation [1]. Raised by artistic forms like *interactive music* or *live coding*, questions related to dynamic notation or musical process representation are taking a significant importance. With interactive music, the cohabitation between static scores and interaction processes is to be organized. With live coding, the border between programmation and score tends to be blurry [2]. Today, the object of the musical notation needs to be redefined to take account of the new dimensions of the current artistic practices.

This paper addresses the dynamic representation of processes and their states in the single graphical space of the musical score.

The musical context gives a specific dimension to the problematic of a process representation, because we are interrested in its current state, but also in a set of time ordered states, including futur possible states. Usability constraints require these state to be readable in real-time, i.e. in the time of the score performance, and to be a guide for the interaction choices of the performer.

The objective of the study is to integrate the representation of processes to INScore [3, 4], an environment for the design of augmented interactive music score.

We will first present existing approaches in the domain of process representation, with a viewpoint centered on musical applications. We will next define the elements to characterize an interaction process. We will finally propose a model of process representation, not in the graphic space - that remains the score domain and the composer responsibility - but in terms of communication between the tools involved in a piece realization.

2. EXISTING APPROACHES

Generally provided with operating systems, there are tools to visualize the system state in terms of resources usage: CPU, memory, network bandwith, etc. (figure 1). In the musical domain, tools for music computation are proposing similar representations: the system state may include both the application resources usage and the system parameters, generaly reflecting audio processes.

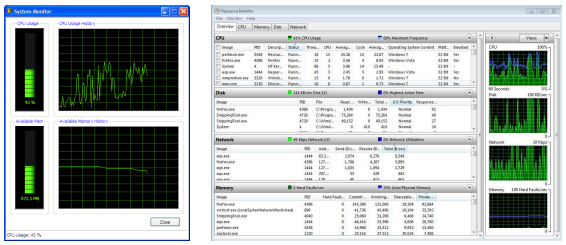
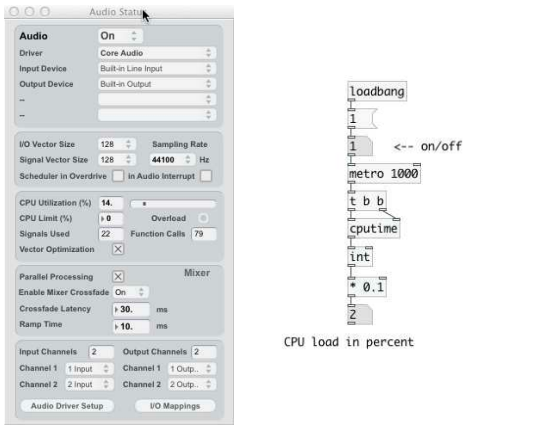


Figure 1. Visualization of the system state under Windows

In the musical domain, PureData [5] and Max/MSP [6] are among the applications frequently used for interactive music design. PureData provides rough tools to represent the system state (figure 2b). Max/MSP includes an audio resources monitor (figure 2a) that gives also the value of the audio setup, like sampling rate, I/O vector size, etc.

In the context of interactive music, the current position in the score is part of the system state but the notion of score is missing win tools like Max/MSP or PureData. Extensions like Bach [7] or MAXScore [8] allow to introduce the music notation in Max/MSP, but without taking account of the musical processes representation.

With environments like Open Music [9] or i-score [10], a process is viewed as an *opaque* box. Its temporal dimension is taken into account and a cursor locates its current time position, but without more information about the process activity.



a) Max/MSP b) Pure Data

Figure 2. Visualization of the system state with Max/MSP and Pure Data

That's in the domain of live coding and interactive systems that the approaches look the more advanced. Thor Magnusson is developping a thought and tools that tend to narrow programming and music score [11, 2]. Ge Wang proposes also an approach based on code [12] as factor of instrumental expression for the *live coding*, with the programming language Chuck [13]. This idea is notably implemented in the audio programming environment *The Audicle* [14] where different visualizations of the code and of the system activity are proposed in an original and extensible approach(figure 3).

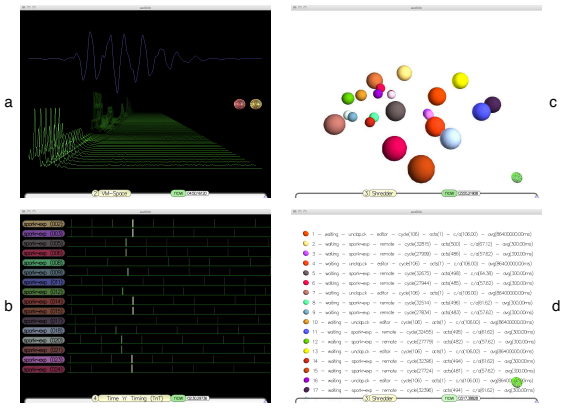


Figure 3. Visualization of the system state in the Audicle : a) waveform and spectrum, b) temporal information per thread, c) threads activity in a graphic form, d) in a detailled textual form.

Another approach of the visualization is centered on the listener, aiming at improving the perception using graphical information in a live coding context [15] or to make the mechanism of electronic instruments perceptible [16] (figure 4).

The main limitation of all these approaches is to be partial and centered on the applications that propose them. In addition, there is no emergent general model and the interoperability between the tools is not taken into account.

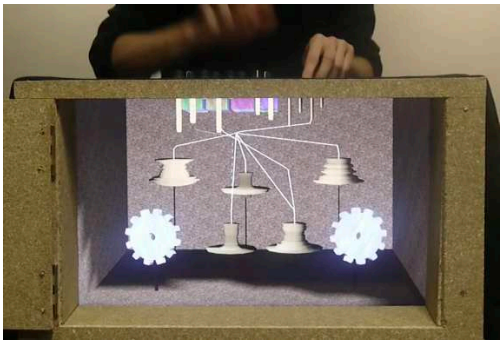


Figure 4. Visualization electronic instruments mechanism.

3. CHARACTERIZATION OF A MUSICAL COMPUT PROCESS

We will consider music computer processes as resulting from the music composition and thus, as being part of the music score. From this viewpoint, a process takes place in time: it can be passed, present or futur. We will talk of *active* process for a process that is present, and of *inactive* process for a past or futur process.

The temporal status will determine different ways to characterize a process: the properties of a process will be different whether active or inactive. Similarly, properties of a past process will differ from those of a futur process, which date and duration may be undefined, relative to an external event.

The characterization of a musical computer process we will be based on three types of information, classified according to their change rate:

- a static state: represent the information that doesn't change (e.g. the parent process) or that may change at a low rate (e.g. the sampling freq.)
- a dynamic state: represents the information that changes over time and depend on the process execution (e.g. the CPU usage)
- a temporal state: the process start date and its end date or duration. These properties may be undefined in an interactive context.

3.1 Static state

A process static state is made of the set of information that are invariant or that change at a low rate. This state is completely defined for an active process. It reflects the last active state for a passed process. For a futur process, it may indicate the first state to activate when it is known in advance.

For a generic process, the state may include:

- its parent process
- its status (active — inactive)
- its computation mode (vectorization, parallélization,...)
- ...

For an audio process, the state may include:

- the I/O buffers size
- the sampling rate
- the I/O devices
- the driver
- ...

For any process, the state may include the value of the control parameters. For example: delay and feedback values of an echo.

The set of static properties is process dependant.

3.2 Dynamic state

The dynamic state of a process is closely linked to its activity and characterizes an active process.

For a generic process, the state may include:

- the CPU usage
- the threads count
- the memory usage
- ...

The result of a process computation may also reflect its state, e.g. the values of a process that computes a signal.

The dynamic state may include indications on the current computation, e.g. an index of confidence for a process that is doing pitch recognition or that is doing score following.

The set of dynamic properties is process dependant.

3.3 Temporal state

In a musical context and especially for interactive music [17], the temporal state of process (i.e. its start date, its end date or its duration) may be partially or totally undefined.

In case of an active process, the start date is known but its end may depend on another process and/or an external event. For a future process, both the start and end dates may be undefined.

In addition, the way to represent dates and durations may lead to undefined results, e.g. when expressed in a time relative to a tempo that is undefined.

In a musical context, this temporal information is critical for the performance of a piece and thus, require to be represented whatever its status.

The temporal properties are common to all the processes.

4. REPRESENTING THE STATE OF A MUSICAL COMPUTER PROCESS

We propose a format to represent the state of a musical process, suitable to inter-applications collaboration, allowing to dissociate a process involved in a piece computation and its graphic representation, that could be viewed as part of the music score.

We define the state of a process as a set of values that characterise this process at a given time. We'll talk of *property* to refer to one of these values. To represent a process state, this process should be able to describe its properties set.

4.1 Definition of a property

A process property is the association of an identifier and a value that may change over time. The value may be a number or a vector of values: e.g. a space position is defined by 3 values.

The value of a property may be bounded by an interval. Finally, the frequency of a property variation may also be defined.

The type of the values is in:

- **int** : for an integer number
- **float** : for a floating point number
- **bool** : for a boolean
- **probability** : the value is a probability. By convention, it will be expressed by floating point numbers in the interval [0, 1].

Values of a **vector** type are defined by a list of types.

```
property : 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. A property definition

By convention, the frequency defines the rate of a value variation: it indicates a number of changes by second. The value 0 denotes constant values. When the variation rate is unknown, the frequency should be omitted.

4.2 Declaration of a properties set

A process state is defined by a list of properties (figure 6). In order to represent this state, a process must be able to communicate a description of its state.

```
process-state : states
states        : property
              | states property
```

Figure 6. Description of a process state

JSON [18] could be used to declare the properties of a process. The figure 7 gives the example of a FAUST [19] process which state include the value of the control parameters as well as the computed signal.

4.3 A property state

The state of a property will be send as a pair associating an identifier and its value.

```
{
  "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. A FAUST process described using JSON

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

Figure 8. Value of the `excitation` of the process `karplus` using the JSON format

When a list of values is associated to an identifier (figure 9), they could be interpreted in the context of the property frequency: when a frequency is defined, the set of values takes the corresponding duration.

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

Figure 9. Values of the `signal` computed by the process `karplus`

OSC [20] may be used to transmit a state. In this case, the OSC address could be used as identifier (figure 10)

4.4 Temporal state representation

All the processes have two common properties: a start date and a duration (or end date). These properties may be undefined (e.g. for a futur process), or partially defined (e.g. for an active process which end date is undefined).

In the context of interactive music, an undefined date corresponds to an external event. For example,

- a process start or end may correspond to the start or end of an improvisation sequence,

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

Figure 10. Values of the `signal` transmitted via OSC

- a process mey be conditionally triggered, e.g. when a specific note sequence is played or in case of silence.

More generally, the scheduling of such processes may be described in terms of Allen relations [21], relatively to the events which they depend on. We will talk of *event based date or duration* to refer to these undefined dates or durations.

The properties of a process temporal state are described by a start and an end or duration (figure 11). The values can be expressed as time or under event form.

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

Figure 11. Properties of a process temporal state

4.4.1 Events representation

Even when a process has an event based date, we would like to represent it, at least in an approximative manner. In order to provide support for such representation, an event based date is defined as a triplet $\mathcal{T} = (t_{left},\ t,\ t_{right})$, associated to a confidence level $\mathcal{P}$ and followed by an optionnal label (figure 12).

$\mathcal{T}$ is such that $t_{left} \leqslant t \leqslant t_{right}$. $\mathcal{T}$ defines a realisation interval $[t_{left},\ t_{right}]$ and a possible realisation date t. The confidence level $[t_{left},\ t_{right}]$ represents the realisation likelihood of the event at the date t. It is expressed as a floating point value in the interval 0, 1.

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

Figure 12. Approximation of event based time.

This kind of representation is intermediate between the description of Allen relations and classical dating. As an example, the following relation `A m (B si (C fi D))` expressed in terms of Allen relations (figure 13) could be expressed in a *semi-instanciated* way as follows:

```
{   "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 ]
}
```

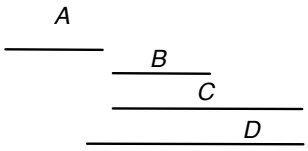


Figure 13. Relations between 4 processes: B and C start with the end of A, D ends with C.

Note that the start date of the process D expresses a constraint on the duration, that should be greater or equal to a whole note.

One of the possibilities for the representation may consist to use a color gradient to account for uncertainties, as illustrated in figure 14, that is based on the example above.

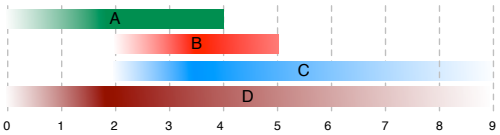


Figure 14. Representation of the processes A, B, C, D, using color gradients to account for uncertainties.

5. CONCLUSION

We propose a simple description of a musical process state. This description is disconnected from any representation format or communication protocol. However, examples using JSON or OSC are given because their simplicity of implementation was consistent with the proposed description of musical processes.

The critical problem of the representation of event based time is treated using a *probabilistic semi-instantiated* approach. This solution is less general than a description in terms or Allen relations, but it avoids solving the corresponding constraints to visualization applications, while realization systems have already to do it.

Acknowledgments

This research has been conducted in the framework of the INEDIT project that is funded by the French National Research Agency [ANR-12-CORD-009].

6. REFERENCES

[1] J. Freeman, "Bringing instrumental musicians into interactive music systems through notation," *Leonardo Music Journal*, vol. 21, no. 15-16, 2011.

[2] T. Magnusson, "Algorithms as scores: Coding live music," *Leonardo Music Journal*, vol. 21, pp. 19–23, 2011.

[3] 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*, 2012, pp. 47–54.

[4] 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*, 2013, pp. 185–190. [Online]. Available: fober-smc2013-final.pdf

[5] M. Puckette, "Pure data: another integrated computer music environment," in *Proceedings of the International Computer Music Conference*, 1996, pp. 37–41.

[6] ——, "Combining Event and Signal Processing in the MAX Graphical Programming Environment," *Computer Music Journal*, vol. 15, no. 3, pp. 68–77, 1991.

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

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

[9] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue, "Computer-assisted composition at ircam: From patchwork to openmusic," *Comput. Music J.*, vol. 23, no. 3, pp. 59–72, Sep. 1999. [Online]. Available: http://dx.doi.org/10.1162/014892699559896

[10] A. Allombert, M. Desainte-Catherine, and G. Assayag, "Iscore: a system for writing interaction," in *Proceedings of the Third International Conference on Digital Interactive Media in Entertainment and Arts, DIMEA 2008, 10-12 September 2008, Athens, Greece*, ser. ACM International Conference Proceeding Series, S. Tsekeridou, A. D. Cheok, K. Giannakis, and J. Karigiannis, Eds., vol. 349. ACM, 2008, pp. 360–367.

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

[12] G. Wang and P. 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*, 2004, pp. 138–143.

[13] ——, "Chuck: a concurrent, on-the-fly audio programming language," in *Proceedings of International Computer Music Conference*, ICMA, Ed., 2003, pp. 219–226.

[14] ——, "The audicle: a contextsensitive, on-the-fly audio programming environ/mentality," in *Proceedings of the International Computer Music Conference*, ICMA, Ed., 2004.

[15] A. McLean, D. Griffiths, N. Collins, and G. Wiggins, "Visualisation of live code," in *Proceedings of the 2010 international conference on Electronic Visualisation and the Arts*, ser. EVA'10. Swinton, UK, UK: British Computer Society, 2010, pp. 26–30. [Online]. Available: http://dl.acm.org/citation.cfm?id=2227180.2227185

[16] F. Berthaut, "Rouages: Revealing the mechanisms of digital musical instruments to the audience," in *(submitted to) Proceedings of the NIME conference*, 2013.

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

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

[19] Y. Orlarey, D. Fober, and S. Letz, *NEW COMPUTATIONAL PARADIGMS FOR COMPUTER MUSIC*, 2009, ch. FAUST : an Efficient Functional Approach to DSP Programming, pp. 65–96.

[20] M. Wright, *Open Sound Control 1.0 Specification*, 2002. [Online]. Available: http://opensoundcontrol.org/spec-1_0

[21] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, pp. 832–843, November 1983. [Online]. Available: http://doi.acm.org/10.1145/182.358434