INTERLUDE - A FRAMEWORK FOR AUGMENTED MUSIC SCORES

D. Fober C. Daudin Y. Orlarey S. Letz
Grame - Centre National de Création Musicale
{fober,daudin,orlarey,letz}@grame.fr

ABSTRACT

An *Augmented Music Score* is a graphic space providing the representation, composition and manipulation of heterogeneous music objects (music scores but also images, text, signals...), both in the graphic and time domains. In addition, it supports the representation of the music performance, considered as a specific sound or gestural instance of the score. This paper presents the theoretical foundation of the augmented music score as well as an application - an augmented score viewer - that implements the proposed solutions.

1. INTRODUCTION

Music notation has a long history and evolved through ages. From the ancient neumes to the contemporary music notation, the western culture is rich of the many ways explored to represent the music. From symbolic or prescriptive notations to pure graphic representation, the music score has always been in constant interaction with the creative and artistic process.

However, although the music representations have exploded with the advent of computer music [1, 2, 3], the music score, intended to the performer, didn't evolved in proportion to the new music forms. In particular, there is a significant gap between interactive music and the static way it is generally notated: a performer has generally a traditional paper score, plus a computer screen displaying a rough number or letter to indicate the state of the interaction system. At the same time, we can observe the emergence of new needs in terms of music representation.

In the domain of electro-acoustic music, analytic scores - music scores made *a postriori* - like the "Portraits polychromes" [1], become common tools for the musicologists but have little support from the existing computer music software, apart the remarkable approach proposed for years by the Acousmograph [4, 5].

In the music pedagogy domain and based on a mirror metaphor, experiments have been made to extend the music score in order to provide feedback to students learning and practising a traditional music instrument [6, 7]. With this approach, an extended music score has been developed, supporting various annotations, including performance representations based on the audio signal, all in the context of a dynamic score layout. The limitations of the system rely mainly on a monophonic score centered approach and on a static design of the performance representation, making the system tricky to extend and to reuse.

New technologies allow now for real-time interaction and processing of musical, sound and gestural information. The Interlude project [2] takes place in this domain and touches upon new digital paradigms for exploration and interaction of expressive movement with music. The present work on *Augmented Music Scores* is part of this project and addresses interaction with symbolic content issues, while extending and generalizing previous music score extension approaches [7].

At the heart of the Augmented Music Score are the following main objectives:

- the score extension to arbitrary graphic objects: actually, we aim to consider arbitrary graphic objects (music scores but also images, text, signal representation...) as possible score candidates;
- the expression of relations between graphic and time space: considering that time is a constant and common property of all musical objects, we give a time position and a time dimension to any score component, which also implies to make the temporal relations graphically visible;
- the performance representation, gesture or audio based, with the aim to develop a system dynamically extensible.

None of the existing systems for music representation includes these kind of features: although generally extended to support contemporary music, tools like Lilypond [8], ENP [9], NoteAbility [10] or the Guido Engine [11] proposes a *traditional* music score approach and are not suited to dynamic music notation.

Although the organization of the graphic space consistently to the time space [12], or the synchronization of various medias [13], are among the current concerns, no formalism has been proposed to express relations between time and graphic space in a general music context.

Although tools for sound visualizations have been developed [14, 4], they are based on a fixed set of representations and don't support dynamic extensions.

[1] http://www.ina-entreprise.com/entreprise/activites/recherches-musicales/portraits-polychromes.html

[2] http://interlude.ircam.fr

Synchronization of heterogeneous medias in the graphic domain raises issues related to non-linearity, non-continuity, non-bijectivity. Based on previous experience, we propose to approach the problem using *segmentation* and the description of relations between *segments* - we will next use the term *mappings* to refer to these relations.

The representation of the music performance, whether sound or gestural, is approached with a reverse perspective: the graphic representation of a signal is viewed as a *graphic signal*, i.e. as a composite signal including all the information for the graphic rendering. This approach, which abstracts the representation calculation, results in an opened system, that can be dynamically extended.

We will first describe the theoretical foundations of the mappings and the context of use in the augmented music score framework. Next, we will explain how the system handles the performance signals to build *graphic signals*. Finally, an augmented music score viewer is presented and particularly its control API, which is actually a set of OSC messages[15].

2. TIME AND GRAPHIC RELATIONS

We talk of *time synchronization in the graphic domain* to refer to the graphic representation of the temporal relations between components of a score. Our previous experience in this domain [6, 7] led us to approach the question of these relations by the means of *segmentation* and the description of relations between *segments*. The term *mappings* is used to refer to these relations.

The role of a *mapping* is to make connections between the *segments* of different resources, where a *segment* is a contiguous zone of a resource. As previously mentioned, a resource can be an arbitrary object (music notation, images, text, signals...). The mappings are typically used to link graphic positions, music time and audio resources locations. For example, a mapping between an audio recording and a music score makes the connection between audio locations (expressed in frames) and the music time (expressed in quarter note divisions). A mapping between a music score and its graphic representation makes the connection between music time positions and graphic positions. Combining these mappings allows to make connections between all the time based resources.

The next sections describe the theoretical foundation for the notions of *segment, segmentation* and *mapping*. These foundations are independent of any implementation and of any resource specific information. They are followed by concrete use cases, implemented in the framework of the augmented score viewer.

2.1 Definitions

We will first introduce the notions of time and graphic segments. Next we will generalize these concrete definitions to an abstract, generic segment definition.

2.1.1 Time segment

A time segment is defined as an interval $i = [t_0, t_1[$ such as $t_0 \leqslant t_1$.

An interval $i = [t_0, t_1[$ is said empty when $t_0 = t_1$. We will use $\emptyset$ to denote empty intervals.

Intersection of time segments (figure 1) is the largest interval such as:

$$\forall i_m,\ \forall i_n,\ i_m \cap i_n := \{j \mid j \in i_m\ \wedge\ j \in i_n\}$$

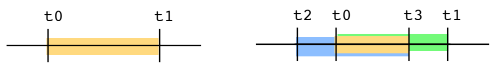

Figure 1. From left to right: a time segment and time segments intersection.

2.1.2 Graphic segment

A graphic segment g is defined as a rectangle given by two intervals $g = (i_x, i_y)$ where i_x is an interval on the x-axis and i_y, on the y-axis.

A graphic segment $g = (i_x, i_y)$ is said empty when $i_x = \emptyset$ or $i_y = \emptyset$

The intersection operation $\cap$ between graphic segments (figure 2) is defined such as:

$$\forall g = (i_x, i_y),\ \forall g' = (i'_x, i'_y),\ g \cap g' = (i_x \cap i'_x, i_y \cap i'_y)$$

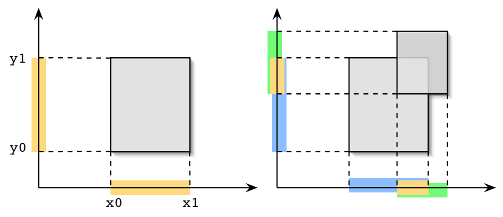

Figure 2. From left to right: a graphic segment and graphic segments intersection.

2.2 Segment generalization

We will extend the definitions above to a general definition of a n-dimensional segment. A n-dimensional segment is defined as a set of n intervals $s^n = \{i_1, ..., i_n\}$ where i_j is an interval on the dimension j.

A segment s^n is said empty when $\exists i \in s^n \mid i = \emptyset$

Intersection between segments is defined as the set of their intervals intersection:

$$s_1^n \cap s_2^n = (i_1 \cap j_1, ..., i_n \cap j_n) \tag{1}$$

where $s_1^n = (i_1, ..., i_n)$ et $s_2^n = (j_1, ..., j_n)$

2.3 Resource segmentation

A *segment-able* resource R is an n dimensions resource defined by a segment S^n of dimension n.
The segmentation of a resource R is the set of segments $Seg(R) = \{s_1^n, ...s_i^n\}$ such as:

$$\begin{array}{rll} \forall i, j \in Seg(R) & i \cap j = \emptyset & \text{the segments are disjoint} \\ \forall i \in Seg(R) & i \cap S^n = i & \text{segments are included in R} \end{array}$$

2.4 Mapping

A *mapping* is a relation between *segmentations*.

For a mapping $M \subseteq Seg(R_1) \times Seg(R_2)$ we define two functions:

$$M^+(i) = \{i' \in Seg(R_2) \mid (i, i') \in M\} \quad (2)$$

that gives the set of segments from R_2 associated to the segment i from R_1; and the reverse function:

$$M^-(i') = \{i \in Seg(R_1) \mid (i, i') \in M\} \quad (3)$$

that gives the set of segments from R_1 associated to the segment i' from R_2.

These functions are defined for a set of segments as the union of each segment mapping:

$$M^+(\{i_1, ...i_n\}) = \{M^+(i_1) \cup M^+(i_2) ... \cup M^+(i_n)\} \quad (4)$$

and

$$M^-(\{i_1, ...i_n\}) = \{M^-(i_1) \cup M^-(i_2) ... \cup M^-(i_n)\} \quad (5)$$

2.5 Mappings composition

Mappings composition is quite straightforward.
For a mapping $M_1 \subseteq Seg(R_1) \times Seg(R_2)$
and a mapping $M_2 \subseteq Seg(R_2) \times Seg(R_3)$, then :

$$M_1 \circ M_2 \subseteq Seg(R_1) \times Seg(R_3)$$

2.6 Augmented Score segmentations and mappings

All the resources that are part of an augmented score have a graphic and a temporal dimension. Thus, they are *segment-able* in the graphic and time spaces. Unless specified otherwise, time is referring to music time (i.e. metronomic time).

In addition, each resource type is *segment-able* in its specific space: audio frames linear space for an audio signal, two dimensional space organized in lines/columns for text, etc.

type	segmentations and mappings required
text	*graphic* $\leftrightarrow$ **text** $\leftrightarrow$ **time**
score	*graphic* $\leftrightarrow$ *wrapped time* $\leftrightarrow$ *time*
image	*graphic* $\leftrightarrow$ **pixel** $\leftrightarrow$ **time**
vectorial graphic	*graphic* $\leftrightarrow$ **vectorial** $\leftrightarrow$ **time**
signal	*graphic* $\leftrightarrow$ **frame** $\leftrightarrow$ **time**

Table 1. Segmentations and mappings for each component type

Table 1 lists the segmentations and mappings used by the different component types. Mappings are indicated using arrows ($\leftrightarrow$). Note that the arrows link segments of different types (the *segment* qualifier is omitted). Segmentations and mappings in *italic* are automatically computed by the system, those in **bold** have to be provided externally. This typology could be extended to any kind of resource, provided that for any new type, a mapping exists to go from the graphic space to the time space.

Note that an intermediate time segmentation, the *wrapped time*, is necessary for music score in order to catch repeated sections and jumps (to sign, to coda, etc.).

Composition of these mappings is at the basis of the mechanisms to address and synchronize the components both in the graphic and time spaces.

2.7 Synchronization examples

Let's consider two score components A and B with their corresponding graphic and time segmentations:

$$Seg(A_g), Seg(A_t), Seg(B_g), Seg(B_t).$$

In addition, B has an intermediate segmentation $Seg(B_l)$ expressed in the resource local space units (e.g. frames for an audio signal). The mappings

$M_A \subseteq Seg(A_g) \times Seg(A_t)$
and $M_B \subseteq Seg(B_t) \times Seg(B_l)$

give the correspondence between graphic and time space for A and between time and local space for B.

When synchronizing objects and deciding on what position should be used as base position, we have introduced a master/slave relation between components: a slave is always constrained to its master space.

2.7.1 Graphic alignment of time positions

It corresponds typically to the alignment of a cursor on a score: the cursor indicates a time position but without temporal extension.

Let's consider that B is A slave and we want to graphically align B to A at a time t. Let $s = [t_0, t_1[$ be the A segment containing the time t. The corresponding graphic segment is:

$$M_A^-(s) = \{g_i \in Seg(A_g) \mid (g, s) \in M_A\}$$

When $M_A^-(s)$ contains a single segment, B graphic position can be computed by simple linear interpolation i.e.:

$$(x_B, y_B) = (g_{x0} + (g_{x1} - g_{x0}).\delta,\ g_{y0})$$

where g_{x0} and g_{x1} are the graphic segment first and last x coordinates and $\delta = (t - t_0)/(t_1 - t_0)$.

y_B is arbitrary fixed to g_{y0} but it is actually controlled by a synchronization mode (over, above, below).

When s is mapped to several graphic segments, the operation can be repeated for each segment.

2.7.2 Segments graphic alignment

It corresponds typically to the alignment of a performance representation: the performance curve is made of segments (e.g. corresponding to notes) and each segment has to be aligned to the corresponding score location and duration.

The basic principle of segments alignment consists for each of the master graphic segment, to retrieve the corresponding slave segment expressed in the slave local coordinates and to render this slave segment in the space of the master graphic segment. Provided that $Seg(A_t) = Seg(B_t)$, the operation may be viewed as the mapping composition

$$M_A \circ M_B \subseteq Seg(A_g) \times Seg(B_l)$$

Figure 3 gives an example of different alignments obtained using different segmentations.

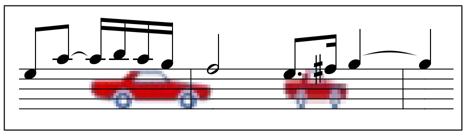

Figure 3. The same car bitmap synchronized to different time positions. The image is made of a single graphic segment and has a quarter note duration. It is stretched to the corresponding score graphic segments.

3. PERFORMANCE REPRESENTATION

Work on the performance representation takes root in previous experiences about the visualization of music instrument playing, made in a pedagogic context [7]. Our previous approach was based on a graphic rendering engine, taking signals and a representation type as input, and producing the corresponding image. The static embedding of the representation types in the rendering engine was one of the main limitations of the approach, implying to modify the engine for any new type.

In the context of the augmented score, our ambition was to develop a dynamically extensible system, avoiding this limitation. To do that, the graphic representation of a signal is viewed as a *graphic signal*, i.e. as a composite signal including all the information required for its graphic rendering.

The resulting performance representation object is a *first order* music score component: it has a date, a duration and thus can be synchronized to any other component.

3.1 Graphic signals

We define a graphic signal as a composite signal made of:

- a y signal: the graphic y coordinates
- a h signal: the graphic thickness at the y position
- a c signal: the graphic color

To make simple, we assume that the color space addressed by c has one dimension. Figure 4 gives an example of these parameters in the graphic space, at a time t.

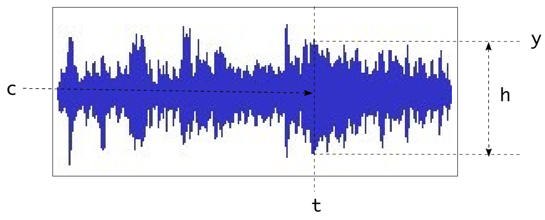

Figure 4. Graphic signal parameters at a time t.

Now, let's consider a signal S defined as a time function:

$$f(t) : \mathbb{R} \to \mathbb{R}^3 = (y, h, c) \mid y, h, c \in \mathbb{R}$$

then this signal may contain everything to be directly drawn i.e. without additional computation.

Such a system may also be viewed as an *oscilloscope* taking the 3 graphic signal components as input.

3.2 Signals composition

In order to build composite signals to be used as graphic signals, we have introduced a signals parallelization operation.

Let's consider $\mathbb{S}$, the set of signals $s : \mathbb{N} \to \mathbb{R}$.
The *parallel* operation '/' is defined as:

$$s_1/s_2/.../s_n : \mathbb{S} \to \mathbb{S}^n \mid s_i \in \mathbb{S} \quad (6)$$

The time function of a parallel signal $s^n \in \mathbb{S}^n$ is the parallelization of each signal's time function:

$$f(t) = (f_0(t), f_1(t), ...f_n(t)) \mid f_i(t) : \mathbb{N} \to \mathbb{R} \quad (7)$$

3.3 Parallel signals types

To implement the system, we have defined several parallel signals types:

- a *color signal* type, based on the HSBA color model [hue, saturation, brigthness, transparency]:

$$c ::= \overrightarrow{(h, s, b, a)} \mid h, s, b, a \in \mathbb{R}$$

- a *graphic signal* type that includes a y signal, a thickness signal th, followed by the 4 components of the color signal:

$$g ::= \overrightarrow{(y, th, h, s, b, a)} \mid y, th, h, s, b, a \in \mathbb{R}$$

- a *parallel graphic signals* type to support several graphic signals in parallel:

$$g^n ::= \overrightarrow{g} \mid g \in \mathbb{R}^6$$

3.4 Graphic signals examples

In order to validate our model, we will describe several representation types that were statically implemented with the previous approach.

3.4.1 Pitch representation

Represents notes pitches on the y-axis using the fundamental frequency (figure 5).

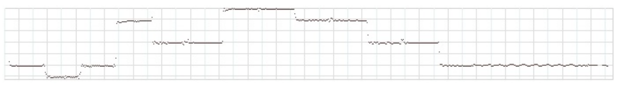

Figure 5. Pitch representation.

The corresponding graphic signal is expressed as:

$$g = S_{f0} \;/\; k_t \;/\; k_c$$

where S_{f0} : fundamental frequency
k_t : a constant thickness signal
k_c : a constant color signal

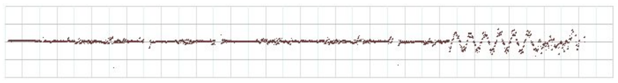

Figure 6. Intonation representation.

3.4.2 *Intonation representation*

Represents the difference between a fundamental frequency and a reference frequency (figure 6).

The corresponding graphic signal is expressed as:

$$g = S_{f0} - S_{fr} \ / \ k_t \ / \ k_c$$

where S_{f0} : fundamental frequency
S_{fr} : reference frequency
k_t : a constant thickness signal
k_c : a constant color signal

3.4.3 *Articulations*

Makes use of the signal RMS values to control the graphic thickness (figure 7).

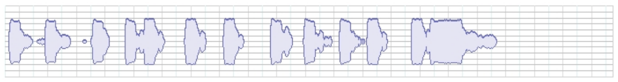

Figure 7. Articulations.

The corresponding graphic signal is expressed as:

$$g = k_y \ / \ S_{rms} \ / \ k_c$$

where k_y : signal y constant
S_{rms} : RMS signal
k_c : a constant color signal

3.4.4 *Pitch and articulation combined*

Makes use of the fundamental frequency and RMS values to draw articulations shifted by the pitches (figure 8).

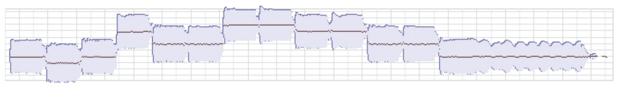

Figure 8. Pitch and articulation combined.

The corresponding graphic signal is expressed as:

$$g = S_{f0} \ / \ S_{rms} \ / \ k_c$$

where S_{f0} : fundamental frequency
S_{rms} : RMS signal
k_c : a constant color signal

3.4.5 *Pitch and harmonics combined*

Combines the fundamental frequency to the first harmonics RMS values (figure 9). Each harmonic has a different color.

We will describe the corresponding graphic signal in several steps. First, we build the fundamental frequency graphic as above (see section 3.4.4) :

$$g0 = S_{f0} \ / \ S_{rms0} \ / \ k_c0$$

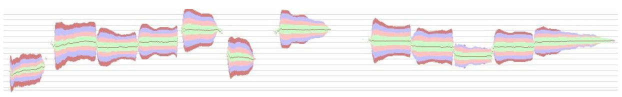

Figure 9. Pitch and harmonics combined.

where S_{f0} : fundamental frequency
S_{rms0} : f0 RMS values
k_c0 : a constant color signal

Next we build the graphic for the harmonic 1:

$$g1 = S_{f0} \ / \ S_{rms1} + S_{rms0} \ / \ k_c1$$

S_{rms1} : harmonic 1 RMS values
k_c1 : a constant color signal

Next, the graphic for the harmonic 2:

$$g2 = S_{f0} / \ S_{rms2} + S_{rms1} + S_{rms0} \ / \ k_c2$$

S_{rms2} : harmonic 2 RMS values
k_c2 : a constant color signal

etc.

And we finally combine them into a parallel graphic signal:

$$g = g2 \ / \ g1 \ / \ g0$$

4. THE AUGMENTED MUSIC SCORE VIEWER

The implementation takes the form of a C++ library - the Interlude library - as well as an augmented score viewer, build on top of this library. This viewer has no user interface since it has been primarily designed to be controlled via OSC messages i.e. using external applications like Max/MSP or Pure Data.

4.1 Messages general format

An Interlude OSC message is made of an OSC address, followed by a message string, followed by 0 to n parameters. The message string could be viewed as the method name of the object identified by the OSC address.

The OSC address is a string or a regular expression matching several objects. The OSC address space includes predefined static nodes:

`/ITL` corresponds to the Interlude viewer application

`/ITL/scene` corresponds to the rendering scene, actually the augmented score address.

The score components have addresses of the form:

`/ITL/scene/anyname` where `anyname` is an arbitrary user defined name.

The score components parameters can be addressed with messages strings like `x`, `y` or `z` to control the x, y or z position. Table 2 gives the main message strings, supported by all the components. Almost these messages have a relative form e.g. `dx` for a relative x displacement.

The next sections present examples of OSC messages setting up a score including synchronized components. Note that the messages list corresponds strictly to the file format of a score. Note also that these examples are static

message strings	component parameters
`x, y, z, scale, angle`	scale and position
`date, duration, clock`	time management
`color, hsb`	color management

Table 2. The main messages, supported by all the score components.

while real-time interaction is always possible, for example to move objects in time by sending `date` or `clock` messages (similar to MIDI clocks).

4.2 A simple cursor example

This example shows a cursor synchronized to a graphic bitmap. Lines beginning with a '#' are comments interleaved with the messages.

```
# creates the score as an image
/ITL/scene/turenas set img "score.png"
# sets the image graphic to time mapping
/ITL/scene/turenas mapf "turenas.map"
# sets the score title and position
/ITL/scene/title set txt "Turenas - John..."
/ITL/scene/title x -0.36
/ITL/scene/title y -0.86
/ITL/scene/title scale 3.0
# creates a rectangle used as cursor
/ITL/scene/cursor set rect 0.004 0.217176
/ITL/scene/cursor z 0.5
/ITL/scene/cursor color 204 0 48 132
# synchronizes the cursor to the score
/ITL/scene/sync cursor turenas v
# moves the cursor in time
/ITL/scene/cursor date 123 4
```

The mapping file *turenas.map* describes the relation betwwen the image segments and the time. The image is segmented in 3 parts corresponding to each lines. For each line, the first segment applies to the graphic space (expressed in pixels intervals) and the second segment to the time space (expressed as rationals).

```
( [27,780[ [15,193[ )   ( [0/4,225/4[ )
( [27,782[ [216,394[ )  ( [225/4,520/4[ )
( [27,511[ [417,594[ )  ( [520/4,594/4[ )
```

The result is given by the figure 10. The cursor is located to the image position corresponding to its date.

4.3 Nested synchronization example

This example uses 3 components; the first one is master of the second, which is master of the third one.

```
# creates a score using an image
/ITL/scene/score set img "score.jpg"
# sets the score graphic to time mapping
/ITL/scene/score mapf "score.map"

# creates a text using a text file
/ITL/scene/text set txtf "comment.txt"
# changes the text scale
/ITL/scene/text scale 3.0
# and the text color
/ITL/scene/text color 0 0 240 255
# put the text in front
/ITL/scene/text z 0.5
# and sets the text to time mapping
/ITL/scene/text mapf "comment.map"
```

```
# creates a ball as vectorial graphic
/ITL/scene/ball set ellipse 0.2 0.2
# puts it in front
/ITL/scene/ball z 0.4
# changes the ball color
/ITL/scene/ball color 250 50 0 255

# sets all the objects date
/ITL/scene/* date 4 1
# sets the text slave of the score
/ITL/scene/sync text score
# sets the ball slave of the text
/ITL/scene/sync ball text
```

Note the use of a wildcard in the OSC address to set all the objects date with a single message. The corresponding result is given by figure 11.

4.4 A signal synchronized to a score

This example show a graphic signal synchronized to a GMN score. Note that a graphic signal is a *first order* music score component: it has a date and a duration and can be synchronized to any other object.

```
# declare a y signal with size 200
/ITL/scene/signal/y size 200
# declare a thickness signal
/ITL/scene/signal/t size 200
# combines y and t + constant color signals
/ITL/scene/signal/sig set y t 0. 1. 1. 1.
# build the corresponding graphic signal
/ITL/scene/myGraph set graph sig
# set its date and duration
/ITL/scene/myGraph date 7 4
/ITL/scene/myGraph duration 2 4
# creates the score
/ITL/scene/score set gmnf "score.gmn"
# synchronize the graphic signal to the score
/ITL/scene/sync myGraph score h
```

The corresponding result is given by figure 12. The signal can move and receive data in real-time. Note that the score graphic to time mapping is automatically computed by the system.

5. CONCLUSION

Our approach for synchronizing arbitrary objects in the graphic space according to their time relations, combines the advantages of simplicity and flexibility: a great variety of behaviors may be obtained depending on the defined segmentations and mappings. This method is independent of any implementation.

The proposed solution to include the performance representation into the music score is also characterized by its simplicity and flexibility. It consists in abstracting the representation computation from the rendering engine, which results in an opened and dynamically extensible system.

The resulting augmented music score supports heterogeneous components and proposes an original music notation approach, opening new spaces to music and performance representation.

There are many potential application domains, including pedagogic applications, games,... But we also hope that new music forms like interactive music, will take advantage of this research and its developments.

The Interlude Augmented Music Score framework is an open source project. The viewer is available from the Interlude web site at http://interlude.ircam.fr.

Acknowledgments
The Interlude project is funded by the Agence Nationale pour la Recherche [ANR-08-CORD-010].

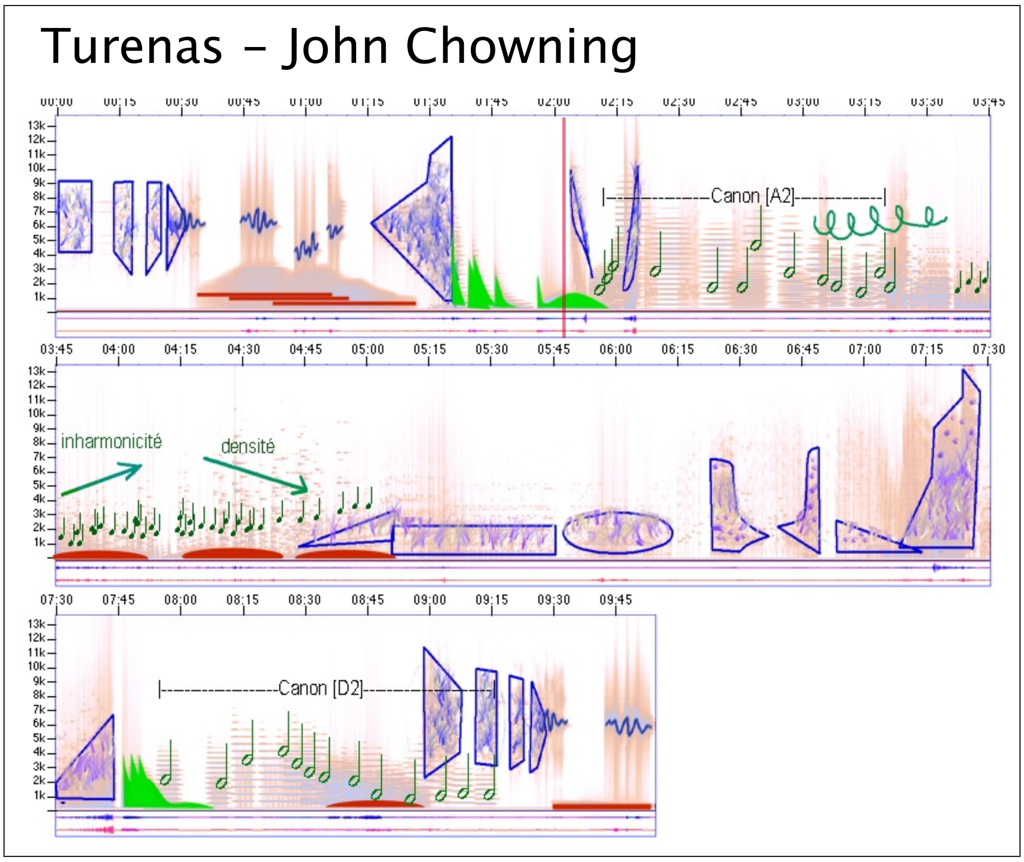

Figure 10. Turenas score: analysis and graphic transcription by Laurent Pottier. The transcription is taken from the INA-GRM "Portraits polychromes" and reimplemented using the augmented score framework.

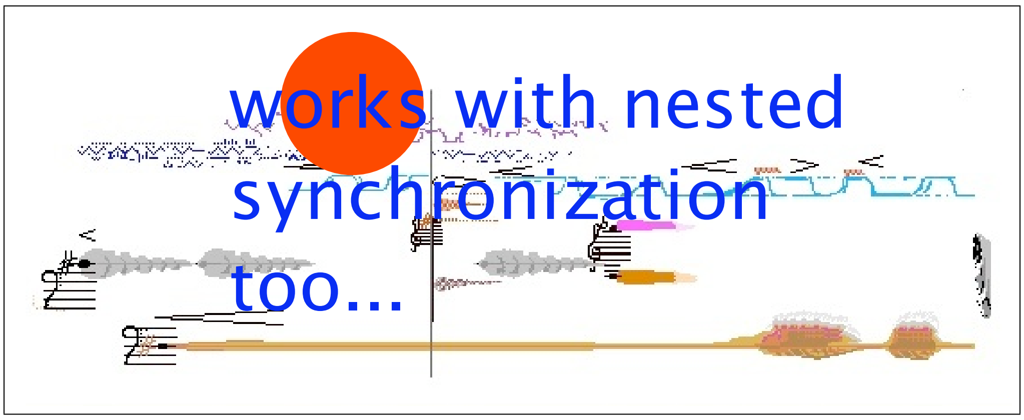

Figure 11. A score with nested synchronization. It includes a bitmap, text and a vectorial graphic. The text is synchronized to the bitmap and the circle to the text. When receiving time messages (e.g. clock), each object moves relatively to its master.

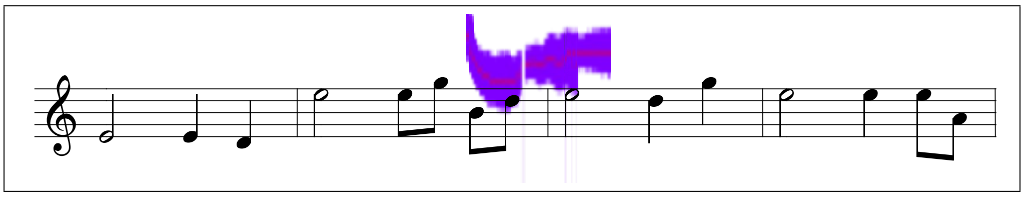

Figure 12. A graphic signal synchronized to a GMN score. The signal can move and receive data in real-time.

6. REFERENCES

[1] R. B. Dannenberg, “Music representation issues, techniques and systems,” *Computer Music Journal*, vol. 17, no. 3, pp. 20–30, 1993.

[2] E. Selfridge-Field, ed., *Beyond MIDI: the handbook of musical codes*. MIT Press, 1997.

[3] W. B. Hewlett and E. Selfridge-Field, eds., *The Virtual Score; representation, retrieval and restoration*. Computing in Musicology, MIT Press, 2001.

[4] Y. Geslin and A. Lefevre, “Sound and musical representation: the acousmographe software,” in *ICMC’04: Proceedings of the International Computer Music Conference*, pp. 285–289, ICMA, 2004.

[5] O. Koechlin and H. Vinet, “The acousmographe, a macinstosh software for the graphical representation of sounds,” in *Proceedings of the International Computer Music Conference (ICMC’91)*, pp. 586–588, ICMA, 1991.

[6] D. Fober, S. Letz, Y. Orlarey, A. Askenfeld, K. F. Hansen, and E. Schoonderwaldt, “Imutus - an interactive music tuition system,” in *Proceedings of the first Sound and Music Computing conference - SMC’04*, pp. 97–103, IRCAM, 2004.

[7] D. Fober, S. Letz, and Y. Orlarey, “Vemus - feedback and groupware technologies for music instrument learning,” in *Proceedings of the 4th Sound and Music Computing Conference SMC’07 - Lefkada, Greece*, pp. 117–123, 2007.

[8] H.-W. Nienhuys and J. Nieuwenhuizen, “LilyPond, a system for automated music engraving,” in *Proceedings of the XIV Colloquium on Musical Informatics*, 2003.

[9] M. Kuuskankare and M. Laurson, “Expressive notation package,” *Computer Music Journal*, vol. 30, no. 4, pp. 67–79, 2006.

[10] K. Hamel, “NoteAbility, a comprehensive music notation system.,” in *Proceedings of the International Computer Music Conference.*, pp. 506–509, 1998.

[11] D. Fober, S.Letz, and Y.Orlarey, “Open source tools for music representation and notation,” in *Proceedings of the first Sound and Music Computing conference - SMC’04*, pp. 91–95, IRCAM, 2004.

[12] J. Bresson and C. Agon, “Scores, programs, and time representation: The sheet object in openmusic,” *Computer Music Journal*, vol. 32, no. 4, pp. 31–47, 2008.

[13] D. Baggi and G. Haus, “IEEE 1599: Music encoding and interaction,” *COMPUTER*, vol. 42, pp. 84–87, March 2009.

[14] C. Cannam, C. Landone, M. Sandler, and J. P. Bello, “The sonic visualiser: A visualisation platform for semantic descriptors from musical signals,” in *Proceedings of the 7th International Conference on Music Information Retrieval*, 2006.

[15] M. Wright, *Open Sound Control 1.0 Specification*, 2002.