INScore
An Environment for the Design of Live Music Scores

D. Fober and **Y. Orlarey** and **S. Letz**
Grame - Centre national de création musicale
9, rue du Garet BP 1185
69202 Lyon Cedex 01,
France,
{fober, orlarey, letz}@grame.fr

Abstract

INScore is an open source framework for the design of interactive, augmented, live music scores. Augmented music scores are graphic spaces providing representation, composition and manipulation of heterogeneous and arbitrary music objects (music scores but also images, text, signals...), both in the graphic and time domains. INScore includes also a dynamic system for the representation of the music performance, considered as a specific sound or gesture instance of the score, and viewed as *signals*. It integrates an *event based* interaction mechanism that opens the door to original uses and designs, transforming a score as a user interface or allowing a score self-modification based on temporal events. This paper presents the system features, the underlying formalisms, and introduces the OSC based scripting language.

Keywords

score, interaction, signal, synchronization

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 [Dannenberg, 1993; Hewlett and Selfridge-Field, 2001], 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 usually notated: a performer has generally a traditional paper score, plus a computer screen displaying a number or a letter to indicate the state of the interaction system. New needs in terms of music representation emerge of this context.

In the domain of electro-acoustic music, analytic scores - music scores made *a postériori*, become common tools for the musicologists but have little support from the existing computer music software, apart the approach proposed for years by the Acousmograph [Geslin and Lefevre, 2004].

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 practicing a traditional music instruments [Fober et al., 2007]. This approach was based on an extended music score, supporting various annotations, including performance representations based on the audio signal, but the system was limited by a monophonic score centered approach and a static design of the performance representation.

Today, new technologies allow for real-time interaction and processing of musical, sound and gestural information. But the symbolic dimension of the music is generally excluded from the interaction scheme.

INScore has been designed in answer to these observations. It is an open source framework[1] for the design of interactive, augmented, live music scores. It extends the traditional music score to arbitrary heterogeneous graphic objects: it supports symbolic music notation (Guido [Hoos et al., 1998] or MusicXML [Good, 2001] format), text (utf8 encoded or html format), images (jpeg, tiff, gif, png, bmp), vectorial graphics (custom basic shapes or SVG), video files, as well as an original performance representation system [Fober et al., 2010b].

Each component of an augmented score has a

[1] http://inscore.sf.net

graphic and temporal dimension and can be addressed in both the graphic and temporal space. A simple formalism, is used to draw relations between the graphic and time space and to represent the time relations of any score components in the graphic space. Based on the graphic and time space segmentation, the formalism relies on simple mathematical relations between segments and on relations compositions. We talk of *time synchronization in the graphic domain* [Fober et al., 2010a] to refer to this specific feature.

INScore is a message driven system that makes use of the Open Sound Control [OSC] [2] format. This design opens the door to remote control and to interaction using any OSC capable application or device. In addition, it includes interaction features provided at score component level by the way of *watchable events*.

All these characteristics make INScore a unique system in the landscape of music notation or of multi-media presentation. The next section gives details about the system features, including the underlying formalisms. Next the OSC API is introduced with a special emphasis on how it is turned into a scripting language. The last section gives an overview of the system architecture and dependencies before some directions are presented for future work.

2 Features

Table 1 gives the typology of the graphic resources supported by the system. All the score elements have graphic properties (position, scale, rotation, color, etc.) and time properties as well (date and duration).

INScore provides a message based API to create and control the score elements, both in the graphic and time spaces. The Open Sound Control [OSC] protocol is used as basis format for these messages, described in section 3.

2.1 Time to graphic relations

All the elements of a score have a time dimension and the system is able to graphically represent the time relationships of these elements. INScore is using segmentations and *mappings* to achieve this *time synchronization in the graphic domain*.

The segmentation of a score element is a set of segments included in this element. A segment may be viewed as a portion of an element. It could be a 2D graphic segment (a rectangle), a time interval, a text section, or any segment describing a part of the considered element in its *local* coordinates space.

[2] http://opensoundcontrol.org/

Mappings are mathematical relations between segmentations. A composition operation is used to relate the graphic space of two arbitrary score elements, using their relation to the time space. Table 2 lists the segmentations and mappings used by the different component types. Mappings are indicated by arrows (↔). Note that the arrows link segments of different types. Segmentations and mappings in *italic* are automatically computed by the system, those in **bold** are user defined.

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

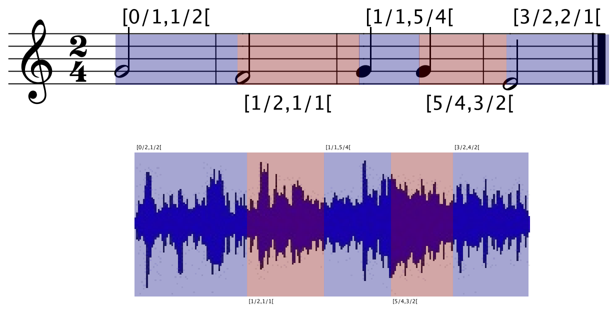

Figure 1: Graphic segments of a score and its performance displayed using colors and annotated with the corresponding time segments.

Figure 1 shows segments defined on a score and an image, annotated with the corresponding time segments. Synchronizing the image to the score will stretch and align the graphic segments corresponding to similar time segments as illustrated by figure 2.

Description of the image graphic to time relation is given in table 3 as a set of pairs including a graphic segment defined as 2 intervals on the x and y axis, and a time segment defined as 2 rationals expressing musical time (where 1 represents a whole note).

Symbolic music description	Guido Music Notation and MusicXML formats
Text	plain text or html (utf8)
Images	jpg, gif, tiff, png, bmp
Vectorial graphics	line, rectangle, ellipse, polygon, curve, SVG code
Video files	using the phonon plugin
Performance representations	see section 2.2

Table 1: Graphic resources supported by the system.

type	segmentations and mappings required
Symbolic music description	*graphic* ↔ *wrapped time* ↔ *time*
Text	*graphic* ↔ **text** ↔ **time**
Images	*graphic* ↔ **pixel** ↔ **time**
Vectorial graphics	*graphic* ↔ **vectorial** ↔ **time**
Performance representations	*graphic* ↔ **frame** ↔ **time**

Table 2: Segmentations and mappings for each component type

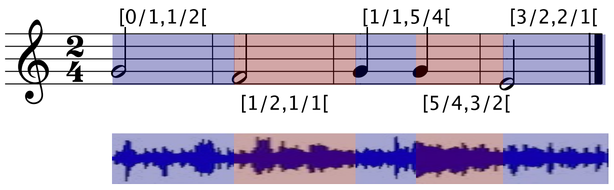

Figure 2: Time synchronization of the segments displayed in figure 1.

([0, 67[[0, 86[ )	([0/2, 1/2[ )
([67, 113[[0, 86[ )	([1/2, 1/1[ )
([113, 153[[0, 86[ )	([1/1, 5/4[ )
([153, 190[[0, 86[ )	([5/4, 3/2[ )
([190, 235[[0, 86[ )	([3/2, 4/2[ )

Table 3: A mapping described as a set of relations between graphic and time segments.

2.2 Performance representation

Music performance representation is based on signals, whether audio or gestural signals. To provide a flexible and extensible system, the graphic representation of a signal is viewed as a *graphic signal*, i.e. as a composite signal made of:

- a y coordinate signal
- a thickness signal h
- a color signal c

Such a composite signal (see figure 3) includes all the information required to be drawn without additional computation.

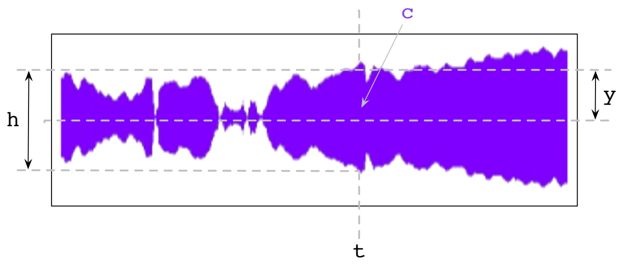

Figure 3: A composite *graphic signal* at time t.

The following examples show some simple representations defined using this model.

2.2.1 Pitch representation

Represents notes pitches on the y-axis using the fundamental frequency (figure 4).

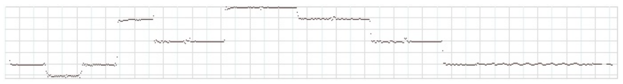

Figure 4: Pitch representation.

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

2.2.2 Articulations

Makes use of the signal RMS values to control the graphic thickness (figure 5). The corresponding graphic signal is expressed as:

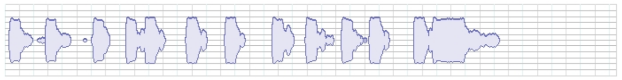

Figure 5: Articulations.

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

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

2.2.3 Pitch and articulation combined

Makes use of the fundamental frequency and RMS values to draw articulations shifted by the pitches (figure 6).

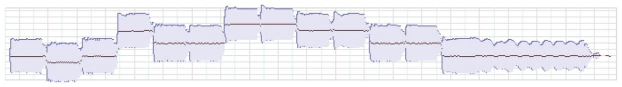

Figure 6: Pitch and articulation combined.

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

2.2.4 Pitch and harmonics combined

Combines the fundamental frequency to the first harmonics RMS values (figure 7). Each harmonic has a different color.

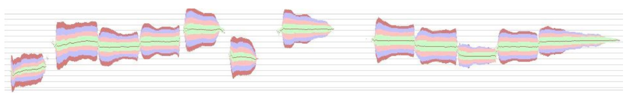

Figure 7: Pitch and harmonics combined.

The graphic signal is described in several steps. First, we build the fundamental frequency graphic as above (see section 2.2.3) :

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

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.
Finally, the graphic signals are combined into a parallel graphic signal:

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

2.3 Interaction

INScore is a message driven system that makes use of Open Sound Control [OSC] format. It includes interaction features provided at individual score component level by the way of *watchable events*, which are typical events available in a graphic environment, extended in the temporal domain. The list of supported events is given in table 4.

mouse events	time events	misc.
mouse enter mouse leave mouse down mouse up mouse move double click	time enter time leave	new element (at scene level)

Table 4: Typology of *watchable* events.

Events are associated to user defined messages that are triggered by the event occurrence. The message includes a destination address (INScore by default) that supports a url-like specification, allowing to target any IP host on any UDP port. The message associated to mouse events may use predefined variables, instantiated at event time with the current mouse position or the current position time.

This simple event based mechanism makes easy to describe for example an *intelligent cursor* i.e. an arbitrary object that is synchronized to the score and that turns the page to the next or previous one, depending on the time zone it enters.

3 INScore messages

The INScore API is an OSC messages API. The general format is illustrated in figure 8: it consists in an address followed by a message string, followed by parameters.

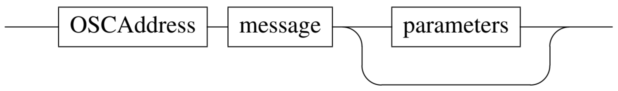

Figure 8: General format of a message.

The address may be viewed as an object pointer, the message string as a method name and the parameters as the method parameters. For example, the message:

```
/ITL/scene/score color 255 128 40 150
```

may be viewed as the following method call:

```
score->color(255 128 40 150)
```

3.1 Address space

The OSC address space is strictly organized like the internal score representation, which is a tree with 4 depths levels (figure 9). Messages can be sent to any level of the hierarchy.

The first level is the application level: a static node with a fixed address that is also used to discriminate incoming messages. An application contains several scenes, which corresponds to different windows and different scores. Each scene contains components and two static nodes: a *sync* node for the components synchronization and a *signal* node, that may be viewed as a special folder containing signals. The name in blue are user defined, those in black are static reserved names.

3.2 Message strings

This section gives some of the main messages supported by all the score components. The list is far from being exhaustive, it is intended to show examples of the system API.

Score components are created and/or modified using a `set` message (figure 10) that takes the object type as argument, followed by type specific parameters.

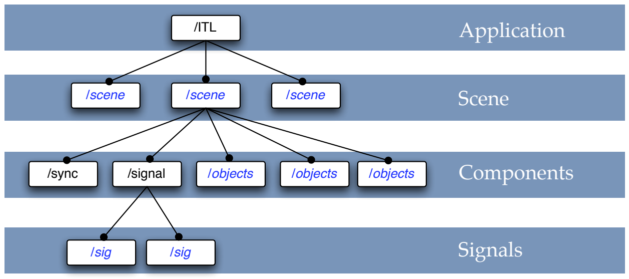

Figure 9: INScore address space.

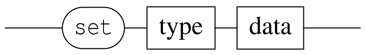

Figure 10: The `set` message.

Messages given in figure 11 are used to set the objects graphic properties.

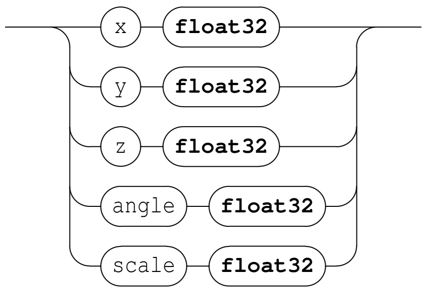

Figure 11: Graphic space management.

Messages given in figure 12 set the time properties. Time is encoded as rational values representing music time (where 1 is a whole note). Graphic space and time management messages have relative positioning forms: the message string prefixed with 'd'.

All the messages that modify the system state have a counterpart `get` form illustrated by figure 13.

Synchronization between components is based on a master / slave scheme. It is described by a message addressed to the static `sync` node as illustrated by figure 14. `syncmode` indicates how to align the slave component to its master: horizontal and/or vertical stretch, etc.

Interactive features are available by requesting to a component to watch an event using

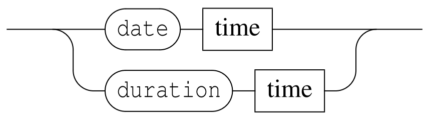

Figure 12: Time management.

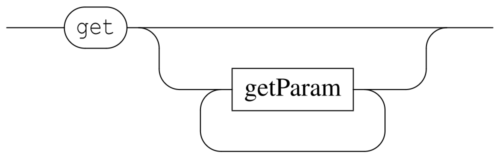

Figure 13: Querying the system state.

a message like figure 15, where **what** indicates the event (mouseUp, mouseDown, mouseEnter, mouseLeave, timeEnter, timeLeave) and `OSCMsg` represents any valid OSC message including the address part.

The example in figure 16 creates a simple score, do some scaling, creates a red cursor and synchronizes it to the score with a vertical stretch to the score height. Output is presented by figure 17.

3.3 INScore scripts

Actually, the example given in figure 16) is making use of the file format of a score, which consists in a list of *textual OSC messages*, separated by a semi-colon. This textual form is particularly suitable to be used as a scripting language and additional support is provided in the form of variables and javascript and/or lua support.

3.3.1 Variables

INScore scripts supports variables declarations in the form illustrated by figure 18. Variables may be used in place of any message parameter prefixed using a $ sign. A variable must be declared before being used.

3.3.2 Scripting support

INScore scripts may include javascript sections, delimited by `<?javascript` and `?>` The javascript code is evaluated at parse time. It is expected to produce valid INScore messages as output. These messages are then expanded in place of the javascript code.

Variables declared before the javascript section are exported to the javascript environment, which make these variables usable in both contexts.

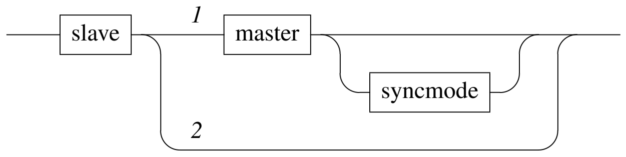

Figure 14: Synchronization message: the second form (2) removes the synchronization from the **slave** object.

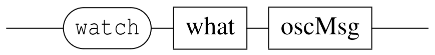

Figure 15: Watching events.

Similarly, INScore may support the lua language in sections delimited by `<?lua` and `?>`. By default, lua support is not embedded in the INScore binary distribution. The engine needs to be compiled with the appropriate options to support lua.

4 Architecture

INScore is both a shared library and a standalone score viewer application without user interface. Some insight of the system architecture is given in this section for a better understanding and an optimal use of the system. The general architecture is a Model View Controller [MVC] designed to handle OSC message streams.

4.1 The MVC Model

The MVC architecture is illustrated in figure 19. Modification of the model state is achieved by incoming OSC messages or by the library C/C++ API, that is actually also message based. An OSC message is packaged into an *internal messages* representation and stacked on a lock-free fifo stack. These operations are synchronous to the incoming OSC stream and to the library API call.

On a second step, messages are popped from the stack by a `Controler` that takes in charge

```
/ITL/scene/score set 'gmn' '[g e f d]';
/ITL/scene/score scale 5.0;
/ITL/scene/cursor set 'rect' 0.01 0.1;
/ITL/scene/cursor color 255 0 0 150;
/ITL/scene/sync 'cursor' 'score' 'v';
```

Figure 16: INScore sample script.

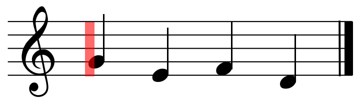

Figure 17: Sample script output.

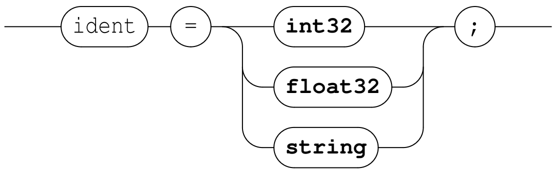

Figure 18: Variable declaration.

address decoding and message passing to the corresponding object of the model. The final operation concerns the *view* update. An `Updater` is in charge of producing a view of the model, which is currently based on the Qt Framework.

This second step of the model modification scheme is asynchronous: it is processed on a regular time base.

4.2 Dependencies

The INScore project depends on external open source libraries:

- the Qt framework[3] providing the graphic layer and platform independence.
- the GuidoEngine[4] for music score layout
- the GuidoQt static library, actually part of the Guido library project
- the oscpack library[5] for OSC support
- and optionally:
  - the MusicXML library[6] to support the MusicXML format.
  - the v8 javascript engine[7] to support javascript in INScore script files.
  - the lua engine[8] to support lua in INScore script files.

[3]`http://qt.nokia.com/`
[4]`http://guidolib.sourceforge.net`
[5]`http://www.rossbencina.com/code/oscpack`
[6]`http://libmusicxml.sourceforge.net`
[7]`http://code.google.com/p/v8/`
[8]`http://www.lua.org/`

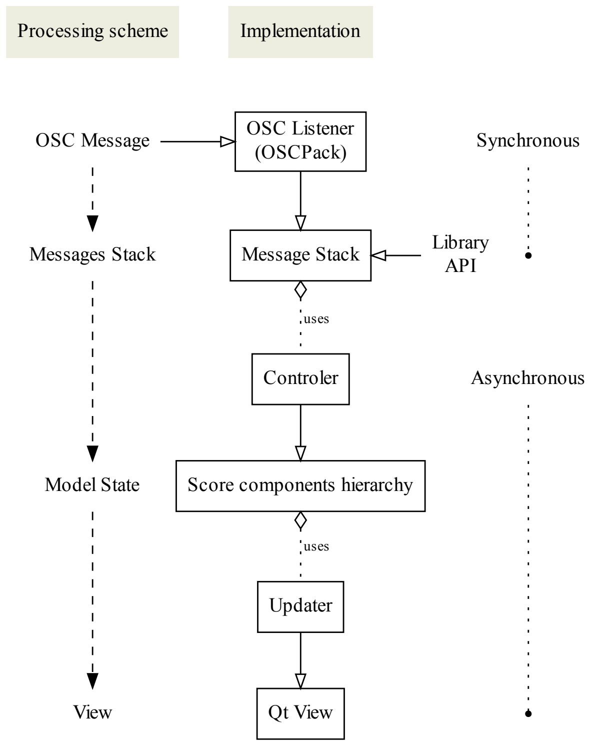

Figure 19: An overview of the MVC architecture

The GuidoEngine, the GuidoQt and oscpack libraries are required to compile INScore.

Binary packages are distributed for Ubuntu Linux, Mac OS and Windows. Detailed instructions are included in the project repository for compiling.

5 Future work

Interactive features described in section 2.3 result from an experimental approach. The formalization and extension of these features are planned.

Time synchronization features are based on a continuous music time. Supporting other time representations like the Allen relations [Allen, 1983] is part of the future work.

Due to its dynamic nature, INScore is particularly suitable to interactive music, i.e. where parts of the music score could be computed in real-time by an interaction system. It could be particularly interesting to provide a musically meaningful visualization of the interaction system state; extensions in this direction are also planned.

6 Acknowledgements

INScore was initiated in the Interlude project funded by the French National Research Agency [ANR- 08-CORD-010].

References

James F. Allen. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26:832–843, November.

R. B. Dannenberg. 1993. Music representation issues, techniques and systems. *Computer Music Journal*, 17(3):20–30.

D. Fober, S. Letz, and Y. Orlarey. 2007. Vemus - feedback and groupware technologies for music instrument learning. In *Proceedings of the 4th Sound and Music Computing Conference SMC'07 - Lefkada, Greece*, pages 117–123.

D. Fober, C. Daudin, S. Letz, and Y. Orlarey. 2010a. Time synchronization in graphic domain - a new paradigm for augmented music scores. In ICMA, editor, *Proceedings of the International Computer Music Conference*, pages 458–461.

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

Yann Geslin and Adrien Lefevre. 2004. Sound and musical representation: the acousmographe software. In *ICMC'04: Proceedings of the International Computer Music Conference*, pages 285–289. ICMA.

M. Good. 2001. MusicXML for Notation and Analysis. In W. B. Hewlett and E. Selfridge-Field, editors, *The Virtual Score*, pages 113–124. MIT Press.

Walter B. Hewlett and Eleanor Selfridge-Field, editors. 2001. *The Virtual Score; representation, retrieval and restoration.* Computing in Musicology. MIT Press.

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