

Programming Interactive Music Scores with INScore

D. Fober, S. Letz, Y. Orlarey

GRAME

Lyon - France

fober, letz, orlarey@grame.fr

F. Bevilacqua

IRCAM

Paris - France

Frederic.Bevilacqua@ircam.fr

ABSTRACT

INSCORE is an environment for the design of interactive music scores that includes an original event-based interaction system and a scripting language for associating arbitrary messages to these events. We extended the previous version by supporting scripting languages offering a great flexibility in the description of scores and in the interactions with scores. The textual format is directly derived from the OSC message format that was defined in the original INSCORE version. This article presents the scripting language and illustrates its ability to describe interactions based on events, while remaining in the temporal space. It also introduces the IRCAM gesture follower and how it is embedded into INSCORE to provide gestural interaction capabilities.

1. INTRODUCTION

INSCORE is a dynamic music score viewer that can be controlled in real-time via OSC messages as well as using OSC based scripts. It supports extended music scores [1], combining symbolic notation with arbitrary graphic objects. All the objects of a score have a time dimension and can be synchronized in a master/slave relationship i.e. any object can be placed in the time space of another object [2]. It can be used in concert, notably for interactive music pieces, for music analysis, for pedagogical applications, etc.

INSCORE has been designed in response to a lack of computer tools for music notation, which did not evolved in proportion to the new forms of musical creation (see eg [3] [4]). In particular, there is a significant gap between interactive music and the way it is statically written.

Music notation generated in interaction with live performance exists for more than a decade. As mentioned by Freeman [5], numerous approaches exist: selection of pre-determined score excerpts [6], mixture of symbolic and graphic elements [7], use of unconventional graphical notation [8], complex staff based notation [9].

These works are based on custom tools, sometimes designed using Max, that are generally specifically suited to a composer approach. Didovsky used JMSL [10] to design interactive scores, but JMSL should be considered more as

a programming language for Java applications developers than an environment for composers. Baird is using Lilypond [11] for audience interaction [12], that can't be considered as a real-time environment for generating music scores, although it works in Baird's context due to relaxed time constraints.

With the recent Bach [13] or MaxScore [14] environments, the symbolic dimension of the music notation starts to be accessible to interaction, first using Max and next the Live environment. However, they are not designed to support unconventional graphical notation, although it could be implemented in Max using Jitter for example.

A unified environment, covering symbolic and graphic notation, opened to real-time interaction is missing and INSCORE aims at fulfilling the needs emerging from the contemporary creation.

Designed to be controlled by OSC messages, INSCORE is naturally turned to an interactive use. The approach to music score programming is also supported by a scripting language based on an extension of the OSC messages, and providing interaction primitives based on *events*. These events are similar to those typically available for user interfaces management (e.g. via Javascript DOM [15]), with an extension in the time domain.

The next section shows two examples of interactive scores, implemented in recent creations using INSCORE. Then it presents the message system and the interaction events, that allow both to describe the music score and to interact with it. Examples of uses are finally given, to illustrate the expressive capabilities of the system.

2. INTERACTIVE MUSIC SCORES

Today, interactive music is subject of convergent artistic and scientific interests. Interaction raises issues for the artistic work composition, description and performance as well. These issues are addressed in the temporal aspects of interactive scores [16] or control [17], and are related to the music piece computation.

For interactive pieces notation, two recent works have used INSCORE to create dynamic scores with original approaches, that also reflect the needs of the contemporary music creation. These works are *Calder's Violin* and *Alien Lands*.

2.1 Calder's Violin

Calder's violin, composed by Richard Hoadley, has been created in Cambridge in October 2011. The piece is de-

fined as a composition for violin and automatic piano. Dynamic symbolic music notation is generated algorithmically and presented to the musician (figure 1) in real-time. This score is played by the musician in parallel to sounds generated by the computer. The technological environment includes SuperCollider for the audio programming and INSCORE for the music notation. For more details, you can refer to [18].



Figure 1. Calder's Violin: sample of music notation.

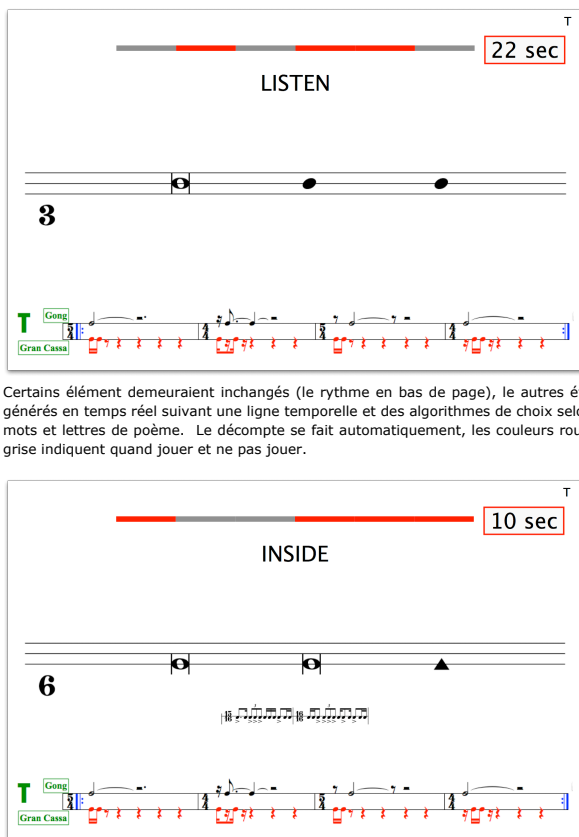


Figure 2. Alien Lands : a complex automatic music score.

3. MUSIC SCORE DESIGN USING MESSAGES

The basic principle for the description of a music score consists in sending OSC messages to the system to create the different score components and to control their attributes, both in graphic and time spaces.

3.1 Format of the messages

The global format of the INSCORE messages is illustrated in figure 3 in a syntax diagram specified in EBNF. It consists in a specialization of the OSC specification that may be viewed as *object oriented*, where the address indicates the target object of the message, *method* indicates a method of the target object and *params*, the method parameters. An INSCORE message could be viewed as a method call of an object of the score.

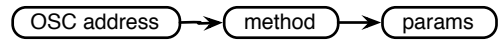


Figure 3. Format of the INScore messages.

The system includes messages to control the objects graphical attributes (position, color, scale, rotation, effects ...) to control their temporal attributes (date, time), to express the relationship between graphic and time spaces, to synchronize different score components, to draw *graphic* signals, and to manage interaction events.

Example 1

Changing the x position of an object named *obj*. The address describes the objects hierarchy: *obj* is embedded in a score named *scene* that is included in the application which address is *ITL*.

```
/ITL/scene/obj x -0.5
```

3.2 Scripting

Although intended to be sent as packets over a network, the OSC messages can be expressed under a textual form, which constitutes the file storage format of a score. This textual form has been extended to enforce the scripting capabilities of the system. The INSCORE viewer supports loading or drag & drop of scripts files, which is equivalent to send the enclosed or evaluated OSC messages to the system.

3.2.1 Extended addresses

The OSC addresses have been extended to support targeting external applications and/or stations (Figure 4). It allows to initialize both the music score and external resources as well using the same script.



Figure 4. Addressing scheme extension.

Example 2

Initializes a score with a Guido Music Notation file [19] and sends a message to an external application listening on port 12000 on a station named *host.adomain.net*. The semicolon (;) is used as a message terminator in a script.

```
/ITL/scene/score set gmnf 'myscore.gmn';  
host.adomain.net:12000/run 1;
```

3.2.2 Variables

Variables have been introduced to allow sharing of parameters between messages. A variable associates an identifier and a parameter list or a list of messages (Figure 5). Variables can be used as message parameter using the form `$identifier`.

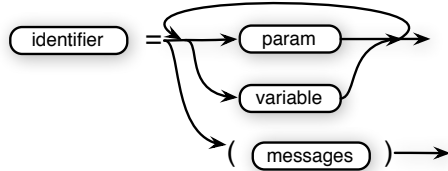


Figure 5. Variables.

Example 3

Variable declaration and use. The exclamation point (!) starts a line comment.

```
color = 200 200 200;
! using the previous color variable
colorwithalpha = $color 100;
/ITL/scene/obj color $colorwithalpha;
```

3.2.3 Languages

INSCORE scripts support programming languages like javascript (default) or lua. The corresponding sections are indicated by angle brackets as in html (Figure 6). The code is evaluated at parse time and the output of the evaluation should be a set of INSCORE messages that will be next parsed in place of the corresponding section.

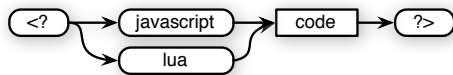


Figure 6. Languages.

4. EVENTS BASED INTERACTION

Interaction is based on associations between events and messages. The messages are sent when the event occurs. The general format of the messages to create such associations is described in Figure 7.

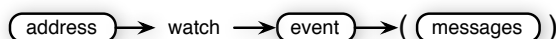


Figure 7. Format of an interaction message.

4.1 Events typologie

Events defined by the system are basically 1) typical user interface events (e.g. mouse click and mouse move) and 2) events defined in the time domain (table 1). This typology

has been extended to *gesture events*, described in section 6.3.

Graphic domain	Time domain
mouseDown	timeEnter
mouseUp	timeLeave
mouseEnter	durEnter
mouseLeave	durLeave
mouseMove	

Table 1. Main events of the system.

In the time domain, an event is triggered when an object date enters (`timeEnter`) or leaves (`timeLeave`) a time interval defined by 2 dates, or when an object duration enters (`durEnter`) or leaves (`durLeave`) an interval bounded by 2 durations.

4.2 Contextual variables

A *contextual variable* is a variable which value depends on an event context (unlike script variables that are evaluated when loading the script). Most of these variables concern the graphic domain and are associated to user interface events; they give the mouse position at the time of the event occurrence and expressed in different reference spaces (`$x` `$y` `$sx` `$sy`). A variable can also give the date corresponding to the current mouse position (`$date`). When an event occurs, the associated messages are evaluated because they may refer to contextual variables.

Example 4

Asking an object to follow the mouse down. The comma (,) is used as separator in a messages list.

```
/ITL/scene/obj watch mouseDown (
  /ITL/scene/obj x '$sx',
  /ITL/scene/obj y '$sy' );
```

4.3 Managing interaction states

Every score component includes a stack to store interaction states. The methods `push` and `pop` are provided to push the current interaction state to the stack and to pop and restore a state from the top of the stack. Examples are given in section 5.3.

5. USE CASES

5.1 Page turning

A simple use case consists in automatic page turning. An object can watch the time intervals corresponding to the different pages and recall a page when it enters its time interval. Time is specified in music time where 1 is a whole note. Note that the `obj` object could be a cursor moving on the score as well.

```
! first page duration is 12 whole notes
/ITL/scene/obj watch timeEnter 0 12
    (/ITL/scene/score page 1);
/ITL/scene/obj watch timeEnter 12 24
    (/ITL/scene/score page 2);
etc.
```

5.2 Sequence of interactions

Interaction messages described in figure 7 accept arbitrary messages to be associated to an event. Thus it is possible to associate an interaction message to an event and to describe sequences of interaction.

Example 5

Description of an interaction sequence based on mouse clicks: the first click changes the object color, the second affects the scaling, the third rotates the object, the fourth modifies the scale too...

```
/ITL/scene/obj watch mouseDown (
  /ITL/scene/obj color 100 100 255,
  /ITL/scene/obj watch mouseDown (
    /ITL/scene/obj scale 1.4,
    /ITL/scene/obj watch mouseDown (
      /ITL/scene/obj angle 45. ,
      /ITL/scene/obj watch mouseDown (
        /ITL/scene/obj scale 0.8 ))) );
```

5.3 Looping a sequence of interactions

A sequence of interactions can be executed n times using the push and pop methods.

Example 6

Executing a sequence of 2 interactions 3 times.

```
/ITL/scene/obj watch mouseDown (
  /ITL/scene/obj color 255 0 0,
  /ITL/scene/obj watch mouseDown (
    /ITL/scene/obj color 0 0 255,
    /ITL/scene/obj pop ))
/ITL/scene/obj push;
/ITL/scene/obj push;
```

Example 7

Executing a sequence of 2 interactions in an infinite loop.

```
/ITL/scene/obj watch mouseDown (
  /ITL/scene/obj push,
  /ITL/scene/obj color 255 0 0,
  /ITL/scene/obj watch mouseDown (
    /ITL/scene/obj color 0 0 255,
    /ITL/scene/obj pop ))
```

5.4 Interaction in the time domain

The sequence of interactions described above (section 5.2) could be defined in the time domain using associations between messages and time events and by moving the object in time. With this approach, it is possible to access the events in a random order but also to control the time flow of the events.

This kind of description combines event based approach, non-sequential access and temporal control.

Example 8

Description of an interaction sequence using time events that are triggered when the object enters consecutive time zones, which duration is a whole note.

```
/ITL/scene/obj watch timeEnter 1 2
    (/ITL/scene/obj color 100 100 255);
/ITL/scene/obj watch timeEnter 2 3
    (/ITL/scene/obj scale 1.4);
/ITL/scene/obj watch timeEnter 3 4
    (/ITL/scene/obj angle 45.);
/ITL/scene/obj watch timeEnter 4 5
    (/ITL/scene/obj scale 0.8);
```

6. INTERACTION WITH GESTURES

INSORE may embed the IRCAM gesture follower as an external plugin. The corresponding objects are similar to signals from input viewpoint. They provide specific interaction events and may also generate streams of messages.

6.1 Principle of the gesture follower

The IRCAM gesture follower is a tool to perform template-based recognition [20, 21]. Technically, the algorithm is available as a C++ library that can be implemented in various environments (up to now the object called *gf* was the most common instantiation of the library in the Max environment). The gestures can be any type of temporal multidimensional times series, that must be regularly time-sampled. Typically, a drawing is a two-dimensional signal, but other signal types can be used such as three, six or nine dimension data obtained from inertial measurement units.

The gesture follower, as most recognition system, is based on two steps. The first step, called *learning*, corresponds to setting a series of "templates". Each template is used to set a Markov Chain modeling the times series. The second step, called *following*, corresponds to "compare" incoming data flow with the stored templates. Technically, the decoding is based on the forward procedure to estimate *likelihoods* of the incoming data to match each templates (note that the forward procedure is incremental compared to a standard Viterbi algorithms). The gesture follower also outputs the *position* (or *temporal index*) that is an estimation of the corresponding current position within the templates, and the estimated *speed* (relative to their templates).

6.2 Gesture follower object

Provided that the corresponding plugin is available, a *gesture follower* object may be embedded in a score. It is created with a fixed set of named gestures to be recognized and thus, its address space is automatically extended to the set of named gestures.

Example 9

Address space of a gesture follower named *myFollower* created to handle 2 gestures named *gestureA* and *gestureB*

```
/ITL/scene/myFollower
/ITL/scene/myFollower/gestureA
/ITL/scene/myFollower/gestureB
```

A gesture follower may take 3 states: a learning state, a following state and an idle state. It receives values that are stored to the corresponding gesture when in learning state, analysed to recognize a gesture when in following state and ignored when idle. Each time the follower receives data in the following state, it produces a set of likelihood, position and speed for each of the gestures.

6.3 Gestures events

Specific events are available from gestures and depends on the gesture state. A gesture may be *active* or *idle*: it is active when its likelihood is greater or equal than a given threshold, otherwise it is idle (figure 8).

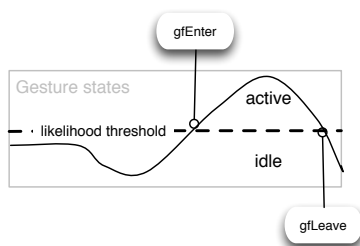


Figure 8. A gesture states and events.

Two specific events are associated to gestures :

- **gfEnter**: triggered when a gesture state moves from idle to active,
- **gfLeave**: triggered when a gesture state moves from active to idle.

6.4 Gesture streams

A gesture supports messages streaming, depending on its state. Figure 9 presents the `send` method that associates a list of messages to the *active* or *idle* state of a gesture. The messages are sent when the gesture follower state is refreshed i.e. when it is in following mode and each time it receives data.

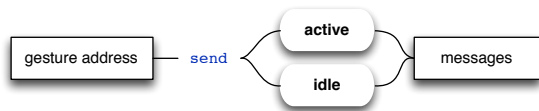


Figure 9. Associating messages to gesture states.

6.5 Variables defined in the context of gestures

Specific variables may be used by messages associated to gesture events or streams:

- **\$likelihood**: gives the current gesture likelihood,
- **\$pos**: indicates the current position in the gesture,

- **\$speed**: indicates the current speed of the gesture.

These variables support scaling and translation of their values when suffixed using an interval. The values denoted by `$pos[1, 5]` represents the current position scaled between 1 and 5.

Example 10

Using a gesture to move a cursor date from 0 to 1.

```
/ITL/scene/gf/gesture send active
(/ITL/scene/cursor date $pos);
```

7. CONCLUSION

Using the OSC protocol to design a scripting language constitutes an original approach which is simple to apprehend for people familiar with OSC. While none of classical programming languages constructs exists in INSCORE scripts, programming capabilities emerge from the objects behavior and leads to new conceptions of music score design.

The association of messages to events reveals to be a simple, powerful and homogeneous way to describe dynamic music scores. A single textual script serves the need of both the static and dynamic parts of the score, leading to new kind of programming e.g. moving of objects in the time domain using an external application when these objects are designed using behaviors linked to time intervals.

This system opens a new dimension to the score components that were previously passive objects: they could react to messages but didn't send messages by themselves. While becoming active and able to send messages, autonomous dynamic behaviors emerge and since each object may embed its own behavior, the system may be viewed as a parallel programmable music score.

However, an external application or the user interaction is necessary to move objects in time. This is currently not considered as a limitation since external applications remain also necessary for the music itself.

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-009-03].

8. REFERENCES

- [1] 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*, 2010, pp. 233–240.
- [2] D. Fober, C. Daudin, S. Letz, and Y. Orlarey, "Time synchronization in graphic domain - a new paradigm for augmented music scores," in *Proceedings of the International Computer Music Conference, ICMA*, Ed., 2010, pp. 458–461.
- [3] T. Magnusson, "Algorithms as scores: Coding live music," *Leonardo Music Journal*, vol. 21, pp. 19–23, 2011.

- [4] J. Freeman, "Bringing instrumental musicians into interactive music systems through notation," *Leonardo Music Journal*, vol. 21, no. 15-16, 2011.
- [5] —, "Extreme sight-reading, mediated expression, and audience participation: Real-time music notation in live performance," *Comput. Music J.*, vol. 32, no. 3, pp. 25–41, Sep. 2008. [Online]. Available: <http://dx.doi.org/10.1162/comj.2008.32.3.25>
- [6] D. Kim-Boyle, "Musical score generation in walses and etudes," in *Proceedings of the 2005 conference on New interfaces for musical expression*, ser. NIME '05. Singapore, Singapore: National University of Singapore, 2005, pp. 238–239. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1085939.1086007>
- [7] G. E. Winkler, "The realtime score. A missing link in computer music performance." in *Proceedings of the Sound and Music Computing conference - SMC'04*, 2004.
- [8] J. Gutknecht, A. Clay, and T. Frey, "Goingpublik: using realtime global score synthesis," in *Proceedings of the 2005 conference on New interfaces for musical expression*, ser. NIME '05. Singapore, Singapore: National University of Singapore, 2005, pp. 148–151. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1085939.1085980>
- [9] N. Didkovsky, "Recent compositions and performance instruments realized in the java music specification language," in *Proceedings of the 2004 international computer music conference*, 2004, pp. 746–749.
- [10] N. Didkovsky and P. Burk, "Java music specification language, an introduction and overview," in *Proceedings of the International Computer Music Conference*, 2001, pp. 123–126.
- [11] H.-W. Nienhuys and J. Nieuwenhuizen, "LilyPond, a system for automated music engraving," in *Proceedings of the XIV Colloquium on Musical Informatics*, 2003.
- [12] K. C. Baird, "Real-time generation of music notation via audience interaction using python and gnu lilypond," in *Proceedings of the 2005 conference on New interfaces for musical expression*, ser. NIME '05. Singapore, Singapore: National University of Singapore, 2005, pp. 240–241. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1085939.1086008>
- [13] 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.
- [14] N. Didkovsky and G. Hajdu, "Maxscore: Music notation in max/msp," in *Proceedings of International Computer Music Conference*, ICMA, Ed., 2008.
- [15] B. Höhrmann, P. Le Hégarret, and T. Pixley, "Document object model (dom) level 3 events specification," World Wide Web Consortium, Working Draft WD-DOM-Level-3-Events-20071221, December 2007.
- [16] A. Allombert, "Aspects temporels d'un système de partitions musicales interactives pour la composition et l'exécution," Ph.D. dissertation, Université Bordeaux 1, Oct. 2009. [Online]. Available: <http://tel.archives-ouvertes.fr/tel-00516350>
- [17] A. Cont, "Antescofo: Anticipatory synchronization and control of interactive parameters in computer music," in *Proceedings of International Computer Music Conference*, ICMA, Ed., 2008.
- [18] R. Hoadley, "Calder's violin: Real-time notation and performance through musically expressive algorithms," in *Proceedings of International Computer Music Conference*, ICMA, Ed., 2012, pp. 188–193.
- [19] H. Hoos, K. Hamel, K. Renz, and J. Kilian, "The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music," in *Proceedings of the International Computer Music Conference*. ICMA, 1998, pp. 451–454.
- [20] F. Bevilacqua, B. Zamborlin, A. Sypniewski, N. Schnell, F. Guédry, and N. Rasamimanana, "Continuous realtime gesture following and recognition," in *In Embodied Communication and Human-Computer Interaction, volume 5934 of Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2010, pp. 73–84.
- [21] F. Bevilacqua, N. Schnell, N. Rasamimanana, B. Zamborlin, and F. Guédry, "Online gesture analysis and control of audio processing," in *Musical Robots and Interactive Multimodal Systems*, ser. Springer Tracts in Advanced Robotics, J. Solis and K. Ng, Eds. Springer Berlin Heidelberg, 2011, vol. 74, pp. 127–142. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-22291-7_8