INScore

AN ENVIRONMENT FOR THE DESIGN OF LIVE MUSIC SCORES

D. Fober, Y. Orlarey, S. Letz
Interactive Music Scores
Interactive Music Scores

- Alien Lands - Sandeep Bhagwati
  [Montréal - February 2011]
  Music performance in four movements for four spatially dispersed percussionists with interactive scores.

- Calder’s Violin - Richard Hoadley
  [Cambridge - October 2011]
  Automatic music for violin and computer.
The Interlude Project

New Digital Paradigms for Exploration and Interaction of Expressive Movement with Music.
The Interlude Project
INScore

- Music score extension
- Graphic & time spaces relationship
- Performance representation
- Interaction
INScore

INScore supports
- Symbolic music notation [GMN, MusicXML]
- Textual elements
- Bitmaps [jpg, gif, tiff, png,...]
- Vectorial graphics (rectangles, ellipses, SVG,...)
- Video files
- Sound and gesture graphic representations

INScore is
- a standalone score viewer
- an open source C/C++ library
- multi-platform
- an Open Sound Control API
INScore
Relations between graphic and time space

Hypothesis
Approach the problem with segmentation and relations between segments
Relations between graphic and time space

Hypothesis
Approach the problem with segmentation and relations between segments

Segments
Defined as a list of intervals:
- generalizable to n dimensions
- intersection operation

Segmentation
A set of disjoined segments
Relations between graphic and time space

Hypothesis
Approach the problem with segmentation and relations between segments
Relations between graphic and time space

Hypothesis
Approach the problem with segmentation and relations between segments

Mapping
Relation between two segmentations:
• operations to query the mapping
• operations to compose mappings
# Relations between graphic and time space

Segmentations and mappings for each component type:

<table>
<thead>
<tr>
<th>type</th>
<th>segmentations and mappings required</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>graphic ↔ text ↔ relative time</td>
</tr>
<tr>
<td>score</td>
<td>graphic ↔ wrapped relative time ↔ relative time</td>
</tr>
<tr>
<td>image</td>
<td>graphic ↔ pixel ↔ relative time</td>
</tr>
<tr>
<td>vect. graphics</td>
<td>vectorial ↔ relative time</td>
</tr>
<tr>
<td>signal</td>
<td>graphic ↔ frame ↔ relative time</td>
</tr>
</tbody>
</table>
Performance representation

The VEMUS approach

- a mirror metaphor
- feedback based pedagogy
- score annotation with performance representations
Performance representation

The VEMUS approach

- a mirror metaphor
- feedback based pedagogy
- score annotation with performance representations

- static design,
- tricky to extend,
- awkward to experiment.
Performance representation

Hypothesis
Approach the graphic of a signal as a graphic signal.
Performance representation

Hypothesis
Approach the graphic of a signal as a graphic signal.

A graphic signal
A composite signal made of:
• a $y$ signal
• a thickness signal
• a color signal
Performance representation

Hypothesis
Approach the graphic of a signal as a graphic signal.

A graphic signal
A composite signal made of:
• a \( y \) signal
• a thickness signal
• a color signal

Consider a signal \( S \) defined as a time function:
\[
f(t) : \mathbb{R} \rightarrow \mathbb{R}^3 = (y, h, c) \mid y, h, c \in \mathbb{R}
\]
This signal could be directly drawn (i.e. without additional computation)
Hypothesis
Approach the graphic of a signal as a graphic signal.

A graphic signal
A composite signal made of:
- a $y$ signal
- a thickness signal
- a color signal

Consider a signal $S$ defined as a time function:
$$f(t) : \mathbb{R} \rightarrow \mathbb{R}^3 = (y, h, c) \mid y, h, c \in \mathbb{R}$$

This signal could be directly drawn (i.e. without additional computation)
Performance representation
System expressivity

Examples

\[ g = S_{f0} / k_t / k_c \]

- \( S_{f0} \): fundamental frequency
- \( k_t \): constant thickness signal
- \( k_c \): constant color signal
Examples

\[ g = \frac{k_y}{S_{rms}} / k_c \]

- \( S_{rms} \): RMS signal
- \( k_y \): constant y signal
- \( k_c \): constant color signal
Examples

\[ g = \frac{S_{f0}}{S_{rms}} \div k_c \]

\( S_{rms} \): RMS signal

\( S_{f0} \): fundamental frequency

\( k_c \): constant color signal
Performance representation
System expressivity

Examples

\[
g_0 = \frac{S_{f_0}}{S_{rms_0}} / k_{c_0} \\
S_{f_0} : \text{fundamental frequency} \\
S_{rms_0} : \text{f0 RMS values} \\
g_1 = \frac{S_{f_0}}{S_{rms_1} + S_{rms_0}} / k_{c_1} \\
S_{rms_1} : \text{f1 RMS values} \\
g_2 = \frac{S_{f_0}}{S_{rms_2} + S_{rms_1} + S_{rms_0}} / k_{c_2} \\
S_{rms_2} : \text{f2 RMS values} \\
... \\
g = \frac{g_2}{g_1} / g_0
\]
INScore OSC Messages

An «object oriented» approach
INScore OSC Messages

An «object oriented» approach

- The OSC address is like an object pointer.
- An OSC message is similar to an object method call.
- The OSC address space is dynamic.

OSC message general format
INScore OSC Messages

An «object oriented» approach

- The OSC address is like an object pointer.
- An OSC message is similar to an object method call.
- The OSC address space is dynamic.

OSC message general format

Example

```
/ITL/scene/score  color  255 128 40 150
score->color(255, 128, 40, 150)
```
INScore OSC Address Space

OSC address
message string
parameter

/ITL

/scene1
/sync
/sig1

/scene2
/signal
/sig2

/scene3
/object1
/object2
/object3

/param1
/param2
/param3
INScore OSC Address Space

Application

/ITL

/scene1
  /sync
    /sig1
  /signal
    /sig2

/scene2
  /object1

/scene3
  /object2
  /object3
INScore OSC Address Space

Application

Scene

/ITL

/scene1
/sync
/sig1

/scene2
/signal
/sig2

/scene3
/object1
/object2
/object3

OSC address
message string
parameter
INScore OSC Address Space

- OSC address
- Message string
- Parameter

Application

Scene

Components

/ITL

/scene1
/scene2
/scene3

/sync
/signal
/object1
/object2
/object3

/sig1
/sig2
INScore OSC Address Space

Application

Scene

Components

Signals

/ITL

/scene1

/scene2

/scene3

/sync

/signal

/object1

/object2

/object3

/sig1

/sig2
Messages Strings

OSC address → message string → parameter
Messages Strings

**Graphic space control**
- **Position:**
  x, y, z, angle, scale, dx, dy, dz, dangle, dscale
- **Color:**
  color, dcolor, red, green, blue, 
  dred, dgreen, dblue, alpha, dalpha, 
  hue, saturation, brightness, dhue, 
  dsaturation, dbrightness

**Time space control**
- **Time position:**
  date, ddate, clock
- **Duration:**
  duration, dduration

**Constructor**
- set

**Query message**
- get

**Time and graphic spaces relations**
- map

**Signals and graphic signals messages**
Messages Parameters

Direct use of basic OSC types
- `int32`
- `float32`
- `OSC-string`

Relaxed types but strict parameters count
Interaction Messages

Basic principle

OSC address → watch → event → message
Interaction Messages

Basic principle

- mouse up, mouse down, mouse move, mouse enter, mouse leave ...
- time enter, time leave
Interaction Messages

Basic principle

OSC address | watch | event | message

address | parameters
Interaction Messages

Basic principle

OSC address \(\xrightarrow{\text{watch}}\) event \(\xrightarrow{\text{message}}\) address \(\xrightarrow{\text{parameters}}\) OSC address
Interaction Messages

Basic principle

- OSC address
- watch
- event
- message
- address
- parameters
- IP address
- port
- host name
- OSC address
Interaction Messages

Basic principle

OSC address → watch → event → message

Examples

/ITL/scene/myObject watch mouseDown

/ITL/scene/myObject show 0
Interaction Messages

Basic principle

Examples

/ITL/scene/myObject watch mouseDown /ITL/scene/myObject show 0

/ITL/scene/myObject watch mouseDown host.domain.org:12100/an/address start
Interaction Messages

Variables

- $x, y, absx, absy, sx, sy
- $date

Address variables

- $self
- $scene

Message based variables

- $(a valid INScore ‘get’ message)

Scaling variable values

- $x[min, max], y[min, max]

Date quantification

- $date[n/d]
Interaction Messages

Variables
- $x, y, absx, absy, sx, sy
- $date

Address variables
- $self
- $scene

Message based variables
- $(a valid INScore 'get' message)

Scaling variable values
- $x[min, max], y[min, max]

Date quantification
- $date[n/d]

0.5

0.5

-1

1

0.5
Interaction Messages

Variables
- $x$, $y$, $absx$, $absy$, $sx$, $sy$
- $date$

Address variables
- $self$
- $scene$

Message based variables
- $(a \text{ valid INScore\ 'get' message)}$

Scaling variable values
- $x[\text{min, max}]$, $y[\text{min, max}]$

Date quantification
- $date[n/d]$
Scripting

- INScore files as script files.
- Supports variables
- Javascript support (embedded by default)
  
  ```javascript
  ... any javascript code ... 
  ```

- optional Lua support (not embedded by default)
  
  ```lua
  ... any lua code ... 
  ```
http://inscore.sourceforge.net