Towards dynamic and animated music notation using INScore

Dominique Fober, Yann Orlarey and **Stéphane Letz**
GRAME - Centre national de création musicale
11 cours de Verdun Gensoul
69002 Lyon
France,
{fober, orlarey, letz}@grame.fr

Abstract

INScore is an environment for the design of augmented interactive music scores opened to conventional and non-conventional use of the music notation. The system has been presented at LAC 2012 and has significantly evolved since, with improvements turned to dynamic and animated notation. This paper presents the latest features and notably the dynamic time model, the events system, the scripting language, the symbolic scores composition engine, the network and Web extensions, the interaction processes representation system and the set of sensor objects.

Keywords

INScore, music score, dynamic score, interaction.

1 Introduction

Contemporary music creation poses numerous challenges to the music notation. Spatialized music, new instruments, gesture based interactions, real-time and interactive scores, are among the new domains that are now commonly explored by the artists. Common music notation doesn't cover the needs of these new musical forms and numerous research and approaches have recently emerged, testifying to the maturity of the music notation domain, in the light of computer tools for music notation and representation. Issues like writing spatialized music [Ellberger et al., 2015], addressing new instruments [Mays and Faber, 2014] or new interfaces [Enström et al., 2015] (to cite just a few), are now subject of active research and proposals.

Interactive music and real-time scores are also representative of an expanding domain in the music creation field. The advent of the digital score and the maturation of the computer tools for music notation and representation constitute the basement for the development of this musical form, which is often grounded on non-traditional music representation [Smith, 2015] [Hope et al., 2015] but may also use the common music notation [Hoadley, 2012; Hoadley, 2014].

In order to address the notation challenges mentioned above, INScore [Fober et al., 2010] has been designed as an environment opened to non-conventional music representation (although it supports symbolic notation), and turned to real-time and interactive use [Fober et al., 2013]. It is clearly focused on music representation only and in this way, differs from tools integrated into programming environments like Bach [Agostini and Ghisi, 2012] or MaxScore [Didkovsky and Hajdu, 2008].

INScore has been already presented at LAC 2012 [Fober et al., 2012a]. It has significantly evolved since and this paper introduces the set of issues that have been more recently addressed. After a brief recall of the system and of the programming environment, we'll present the scripting language extensions and the symbolic scores composition engine that provides high level operations to describe real-time and interactive symbolic scores composition. Next we'll describe how interaction processes representations can be integrated into the music score and how remote access is supported using the network and/or Web extensions. Tablet and smartphone support have led to integrate gestural interaction with a set of *sensor* objects that will be presented. Finally, the time model, recently extended, will be described.

2 The INScore environment

INScore is an environment to design interactive augmented music scores. It extends the music representation to arbitrary graphic objects (symbolic notation but also images, text, vectorial graphics, video, signals representation) and provides an homogeneous approach to manipulate the score components both in the graphic and time spaces.

It supports *time synchronization in the* *graphic space*, which refers to the graphic representation of the temporal relations between components of a score - via a synchronization mechanism and using *mappings* that express relations between time and graphic space segmentations (Fig. 1).

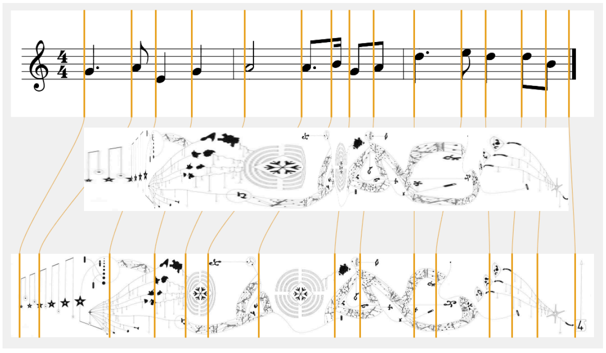

Figure 1: A graphic score (Mark Applebaum's graphic score *Metaphysics of Notation*) is synchronized to a symbolic score. The picture in the middle is the result of the synchronization. The vertical lines express the graphic to graphic relationship, that have been computed by composing the objects common relations with the time space.

INScore has been primarily designed to be controlled via OSC[1] messages. The format of the messages consists in an OSC address followed by a message string and 0 to n parameters (Fig. 2).

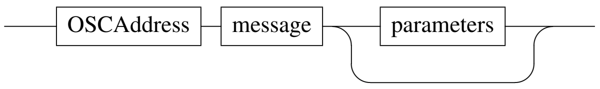

Figure 2: INScore messages general format.

Compared to object oriented programming, 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:

`ITL[scene[score]]->color(255 128 40 150)`

The system provides a set of messages for the graphic space control (`x`, `y`, `color`, `space`, etc.), for the time space control (`date`, `duration`, etc.), and to manage the environment. It includes two special messages:

- the `set` message that operates like a constructor and that takes the object type as parameter, followed by type specific parameters (Fig. 3).
- the `get` message provided to query the system state (Fig. 4).

```
/ITL/scene/obj set txt "Hello world!";
```

Figure 3: A message that creates a textual object, which type is `txt`, with a text as specific data.

```
/ITL/scene/obj get;
  -> /ITL/scene/obj set txt "Hello
world!";

/ITL/scene/obj get x y;
  -> /ITL/scene/obj x 0;
  -> /ITL/scene/obj y 0.5;
```

Figure 4: Querying an object with a `get` message gives messages on output (prefixed with `->`). These messages can be used to restore the corresponding object state.

The address space is dynamic and not limited in depth. It is hierarchically organized, the first level `/ITL` is used to address the application, the second one `/ITL/`*scene* to address the score and the next ones to address the components of a score (note that *scene* is a default name that can be user defined). Arbitrary hierarchy of objects is supported.

3 The scripting language

The OSC messages described above have been turned into a textual version to constitue the INScore scripting language. This language has been rapidly extended to support :

- variables, that may be used to share parameters between messages (Fig. 5).
- message based variables and/or parameters that consists in querying an object to retrieve one of it's attributes value (Fig. 6).
- an extended OSC addressing scheme that allows to send OSC messages to an external application for initialization of control purposes (Fig. 7).

[1]`http://opensoundcontrol.org/`

- JavaScript sections that can be evaluated at parsing and/or run time. A JavaScript call is expected to produce INScore messages as output (Fig. 8).
- mathematical expressions (`+ - / *`, `conditionals`, etc.) that can be used for arguments computation (Fig. 9).
- symbolic scores composition expressions that are described in section 4.

```
greylevel = 140;
color = $greylevel $greylevel $greylevel;
/ITL/scene/obj1 color $color;
/ITL/scene/obj2 color $color;
```

Figure 5: Variables may be used to share values between messages.

```
ox = $(/ITL/scene/obj get x);
/ITL/scene/obj2 x $(/ITL/scene/obj get x);
```

Figure 6: The output of `get` messages can be used by variables or as another message parameter.

```
/ITL/scene/obj set txt "Hello world!";
localhost:8000/start;
```

Figure 7: This script initialises a textual object and sends the `/start` message to an external application listening on UDP port 8000.

4 Symbolic scores composition

Rendering of symbolic music notation makes use of the Guido engine [Daudin et al., 2009]. Thus the primary music score description format is the Guido Music Notation format [Hoos et al., 1998] [GMN]. The MusicXML format [Good, 2001] is also supported via conversion to the GMN format.

The Guido engine provides a set of operators for scores level composition [Fober et al., 2012b]. These operators consistently take 2 scores as argument to produce a new score as output. They allow to put scores in sequence `(seq)`, in parallel `(par)`, to cut a score in the time dimension `(head, tail)`, in the polyphonic dimension `(top, bottom)`, to transpose `(transpose)`, to stretch `(duration)` a score and to apply the

```
<?javascript
  function randpos(address) {
    var x = (Math.random() * 2) - 1;
    return address + " x " + x + ";";
  }
?>
/ITL/scene/javascript run
        'randpos("/ITL/scene/obj")';
```

Figure 8: The JavaScript section defines a `randpos` function that computes an `x` message with a random value, addressed to the object given as parameter. This function may be next called at initialization or at any time using the static JavaScript node embedded into each score.

```
/ITL/scene/o x ($shift ? $x + 0.5 : $x);
```

Figure 9: A mathematical expression is used to compute the position of an object depending on 2 previously defined variables.

rhythm or the pitch of a score to another one `(rhythm, pitch)`.

The INScore scripting language includes *score expressions*, a simple language providing score composition operations. The novelty of the proposed approach relies on the dynamic aspects of the operations, as well as on the persistence of the score expressions. A score may be composed as an arbitrary graph of score expressions and equipped with a fine control over the changes propagation.

4.1 Score expressions

A score expression is defined as an operator followed by two scores (Fig. 10). The leading `expr` token is present to disambiguate parenthesis in the context of INScore scripts.

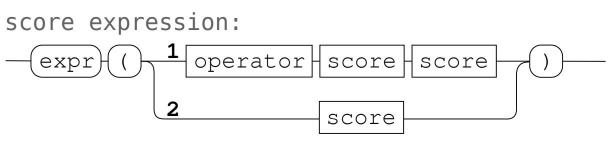

Figure 10: Score expressions syntax.

The score arguments may be:

- a literal score description string (GMN or MusicXML formats),
- a file (GMN or MusicXML formats),
- an existing score object,

- a score expression.

An example is presented in Fig. 11.

```
expr( par score.gmn (seq "[c]" score))
```

Figure 11: A score expression that puts a score file (`score.gmn`) in parallel with the sequence of a literal score and an existing object (`score`). Note that the leading `expr` token can be omitted inside an expression.

4.2 Dynamic score expression trees

The score expressions language is first transformed into an internal tree representation. In a second step, this representation is evaluated to produce GMN strings as output, that are finally passed to the INScore object as specific data.

Basically, the tree is reduced using a depth first post-order traversal and the result is stored in a cache. However, the score expressions language provides a mechanism to make arbitrary parts of a tree variable using an ampersand (&) as prefix of an argument, preventing the corresponding nodes to be reduced at cache level (Fig. 12).

```
expr( par score.gmn (seq "[c]" &score))
```

Figure 12: A score expression that includes a reference to a `score` object. Successive evaluations of the expression may produce different results, provided that the `score` object has changed.

INScore events system (described in section 8.2) provides a way to automatically trigger the re-evaluation of an expression when one of it's variable parts has changed. These mechanisms open the door to dynamic scores composition within the INScore environment. More details about the score expressions language can be found in [Lepetit-Aimon et al., 2016].

5 Musical processes representation

INScore includes tools for the representation of musical processes within the music score. In the context of interactive music and/or when a computer is involved in a music performance, it may provide useful information regarding the state of the musical processes running on the computer. This feedback can notably be used to guide the interaction choices of the performer.

On INScore side, a process state is viewed as a signal. Signals are part of a score components and can be combined into *graphic signals* to become first order objects of a score. They may be notably used for the representation of a performance [Fober et al., 2012a].

Regarding musical processes representation, a signal can be connected to any attribute of an object (Fig. 13), which makes the signal variations visible (and thus the process activity) with the changes of the corresponding attributes.

```
/ITL/scene/signal/sig size 100;
/ITL/scene/obj set rect 0.5 0.5;
/ITL/scene/img set img 'file.png';
/ITL/scene/signal connect sig
    "obj:scale" "img:rotatez[0,360]";
```

Figure 13: A signal `sig` is connected to the `scale` attribute of an object and to the `rotatez` attribute of an image. Note that for the latter, the signal values (expected to be in [-1,1]) are scaled to the interval [0,360].

6 Network and Web dimensions

INScore supports the aggregation of distributed resources over Internet, as well as the publication of a score via the HTTP and/or WebSocket protocols. In addition, a score can also be used to control a set of remote scores on the local network using a *forwarding* mechanism.

6.1 Distributed score components

Most of the components of a score can be defined in a literal way or using a file. All the file based resources can be specified as a simple file path, using absolute or relative path, or as an HTTP url (Fig 14).

```
/ITL/scene/obj1 set img 'file.png';
/ITL/scene/obj2 set img
      'http://www.adomain.org/file.png';
```

Figure 14: File based resources can refer to local or to remote files.

When using a relative path, an absolute path is built using the current path of the score, that may be set to an arbitrary location using the `rootPath` attribute of the score (Fig 15).

The current rootpath can also be set to an arbitrary HTTP url, so that further use of a relative path will result in an url (Fig. 16).

```
/ITL/scene rootPath '/users/me/inscore';
/ITL/scene/obj set img 'file.png';;
```

Figure 15: The rootPath of a score is equivalent to the current directory in a shell. With this example, the system will look for the file at '/users/me/inscore/file.png'

```
/ITL/scene rootPath
              'http://www.adomain.org';
/ITL/scene/obj set img 'file.png';;
```

Figure 16: The rootPath supports urls. With this example, the system will look for the file at 'http://www.adomain.org/file.png'

This mechanism allows to mix local and remote resources in the same music score, but also to express local and remote scores in a similar way, just using a rootPath change.

6.2 HTTPd and WebSocket objects

A music score can be published on the Internet using the HTTP or the WebSocket protocols. Specific objects can be embedded in a score in order to make this score available to remote clients (Fig. 17).

```
/ITL/scene/http set httpd 8000;
/ITL/scene/ws set websocket 8100 200;
```

Figure 17: This example creates an httpd server listening on the port 8000 and a WebSocket server listening on the port 8100 with a maximum notification rate of 200 ms.

The WebSocket server allows bi-directional communication between the server and the client. It sends notifications of score changes each time the graphic appearance of the score is modified, provided that the notification rate is lower than the maximum rate set at server creation time.

The communication scheme between a client and an INScore Web server relies on a reduced set of messages. These messages are protocol independent and are equally supported over HTTP or WebSocket :

- `get`: requests an image of the score.
- `version`: requests the current version of the score. The server answers with an integer value that is increased each time the score is modified.
- `post`: intended to send an INScore script to the server.
- `click`: intended to allow remote mouse interaction with the score.

More details are available from [Fober et al., 2015].

6.3 Messages forwarding

Message forwarding is another mechanism provided to distribute scores over a network. It operates at application and/or score levels when the forward the message is send to the application (/ITL) or to a score (/ITL/scene). The message takes a list of destination hosts specified using a host name or an IP number, and suffixed with a port number. All the OSC messages may be forwarded, provided they are not filtered out (Fig. 18). The filtering strategy is based on OSC adresses and/or on INScore methods (i.e. messages addressing specific objects attributes).

```
/ITL forward 192.168.1.255:7000;
/ITL/filter reject
    '/ITL/scene/javascript';
```

Figure 18: The application is requested to forward all messages on INScore port (7000) to the local network using a broadcast address. Messages addressed to the JavaScript engine are filtered out in order to only forward the result of their evaluation.

7 The *sensor* objects

INScore runs on the major operating systems including Android and iOS. Tablet and smartphone support have led to integrate gestural interaction with a set of sensor (Table 1).

Sensors can be viewed as objects or as signals. When created as a signal node, a sensor behaves like any signal but may provide some additional features (like calibration). When created as a score element, a sensor has no graphical appearance but provides specific sensor events and features.

All the sensors won't likely be available on a given device. In case a sensor is not supported, an error message is generated at creation request and the creation process fails.

8 The time model

INScore time model has been recently extended to support *dynamic* time. Indeed and with the

name	values
accelerometer	x, y, z
ambient light	light level
compass	azimuth
gyroscope	x, y, z
light	a level in lux
magnetometer	x, y, z
orientation	device orientation
proximity	a boolean value
rotation	x, y, z
tilt	x, y

Table 1: The set of sensors and associated values

initial design, the time attributes of an object are fixed and don't change unless a time message `(date, duration)` is received, which can only be emitted from an external application or using the *events* mechanism. The latter (defined very early) introduced another notion of time: the *events* time, which takes place when an event occurs. The *events* system has also been extended for more flexibility.

8.1 The musical time

Regarding the time domain, any object of a score has a date and a duration. A new tempo attribute has been added, which has the effect of moving the object in the time dimension when non null, according to the tempo value and the absolute time flow. Let t_0 be the time of the last tempo change of an object, let v be the tempo value, the object date d_t at a time t is given by a time function f:

$$f(t) \rightarrow d_t = d_{t0} + (t - t_0) \times v \times k, \quad t \geqq t_0 \quad (1)$$

where d_i is the object date at time t_i and k a constant to convert absolute time in musical time. In fact, absolute time is expressed in milliseconds and the musical time unit is the whole note. Therefore, the value of k is $1/1000 \times 60 \times 4$.

Each object of a score has an independent tempo. The tempo value is a signed integer, which means that an object can move forward in time but backward as well.

From implementation viewpoint and when its tempo is not null, an object sends `ddate` (a relative displacement in time) to itself at periodic intervals (Fig. 19).

This design is consistent with the overall system design since it is entirely message based. It is thus compatible with all the INScore mechanisms such as the forwarding system.

```
/ITL/scene/obj tempo 60

-> /ITL/scene/obj ddate f(r_i)
-> /ITL/scene/obj ddate f(r_{i+1})
-> /ITL/scene/obj ddate f(r_{i+2})
-> ...
```

Figure 19: A sequence of messages that activate the time of an object `obj`. Messages prefixed by `->` are generated by the object itself. r_i is the value of the absolute time elapsed between the task i and $i-1$.

8.2 The events system

The event-driven approach of time in INScore preceded the musical time model and has been presented in [Fober et al., 2013]. The event-based interaction process relies on messages that are associated to events and that are sent when the corresponding event occurs. The general format of an interaction message is described in Fig. 20.

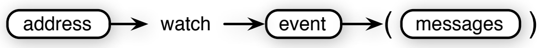

Figure 20: Format of an interaction message: the `watch` request installs a messages list associated to the event `event`.

Initially, the events typology was limited to classical user interface events (e.g. mouse events), extended in the time domain (see Table 2).

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

Table 2: Main INScore events in the initial versions.

This typology has been significantly extended to include:

- touch events `(touchBegin, touchEnd, touchUpdate)`, available on touch screens and supporting multi-touch.
- any attribute of an object: modifying an object attribute may trigger the corresponding event, that carries the name of the attribute (e.g. `x, y date`, etc.).

- an object specific data i.e. defined with a `set` message. The event name is `newData` and has been introduced for the purpose of the symbolic score composition system.
- user defined events, that have to comply to a special naming scheme.

Any event can be triggered using the `event` message, followed by the event name and event's dependent parameters. The `event` message may be viewed as a function call that generates OSC messages on output. This approach is particularly consistent for user events that can take an arbitrary number of parameters, which are next available to the associated messages under the form of variables named $1...$$n$ (Fig. 21).

```
/ITL/scene/obj watch MYEVENT (
  /ITL/scene/t1 set txt $1,
  /ITL/scene/t2 set txt $2
);
/ITL/scene/obj event MYEVENT
              "This text is for t1"
              "This one is for t2";
```

Figure 21: Definition of a user event named MYEVENT that expects 2 arguments referenced as $1 and $2 in the body of the definition. This event is next triggered with 2 different strings as arguments.

The time dimension of the events system allows to put *functions* in the time space under the form of events that trigger messages that can modify the score state and/or be addressed to external applications using the extended OSC addressing scheme (Fig. 22).

Combined with the dynamic musical time, this events system allows to describe autonomous animated score. The example in Fig. 23 shows how to describe a cursor that moves forward and backward over a score by watching the time intervals that precedes and follows a symbolic score and by inverting the tempo value.

9 Conclusion

INScore[2] is an ongoing open source project that crystallizes a significant amount of research addressing the problematics of the music notation and representation in regard of the contemporary music creation. It is used in artistic

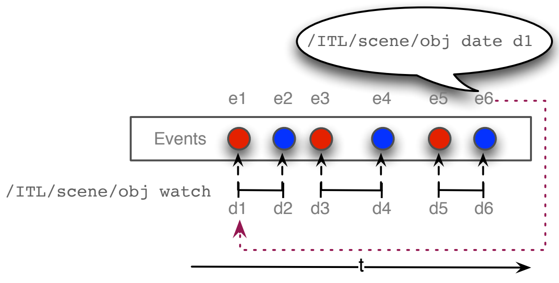

Figure 22: Exemple of events placed in the time space. These events are associated to time intervals (`timeEnter` and `timeLeave`) and are triggered when entering (in red) of leaving (in blue) these intervals. The last event (e6) emits a `date` message that creates a loop by putting the object back at the beginning of the first interval.

```
# first clear the scene
/ITL/scene/* del;

# add a simple symbolic score
/ITL/scene/score set gmn '[c d e f g a h
c2 ]';

# add a cursor synchronized to the score
/ITL/scene/cursor set ellipse 0.1 0.1;
/ITL/scene/cursor color 0 0 250;
/ITL/scene/sync cursor score syncTop;

# watch different time zones
/ITL/scene/cursor watch timeEnter 2 3
  ( /ITL/scene/cursor tempo -60 );

/ITL/scene/cursor watch timeEnter -1 0
  ( /ITL/scene/cursor tempo 60 );

# and finally start the cursor time
/ITL/scene/cursor tempo 60;
```

Figure 23: A cursor that moves forward and backward over a symbolic score.

projects and many of the concrete experiences raised new issues that are reflected into some of the system extensions. The domain is quite recent and there are still a lot of open questions that we plan to address in future work and in particular:

- turning the scripting language into a *real* programming language would provide a more powerful approach to music score description. The embedded JavaScript en-

[2]`http://inscore.sf.net`)

gine may already be used for an algorithmic description of a score, but switching from one environment (INScore script) to another one (JavaScript) proved to be a bit tedious.

- extending the score components to give a time dimension to any of their attributes could open a set of new possibilities, including arbitrary representations of the passage of time.

Finally, migrating the INScore native environment to the Web is part of the current plans and should also open new perspectives, notably due to the intrinsic connectivity of Web applications.

References

Andrea Agostini and Daniele Ghisi. 2012. Bach: An environment for computer-aided composition in max. In ICMA, editor, *Proceedings of International Computer Music Conference*, pages 373–378.

C. Daudin, Dominique Fober, Stephane Letz, and Yann Orlarey. 2009. The guido engine – a toolbox for music scores rendering. In LAC, editor, *Proceedings of Linux Audio Conference 2009*, pages 105–111.

Nick Didkovsky and Georg Hajdu. 2008. Maxscore: Music notation in max/msp. In ICMA, editor, *Proceedings of International Computer Music Conference*.

Emile Ellberger, Germán Toro-Perez, Johannes Schuett, Linda Cavaliero, and Giorgio Zoia. 2015. A paradigm for scoring spatialization notation. In Marc Battier, Jean Bresson, Pierre Couprie, Cécile Davy-Rigaux, Dominique Fober, Yann Geslin, Hugues Genevois, François Picard, and Alice Tacaille, editors, *Proceedings of the First International Conference on Technologies for Music Notation and Representation - TENOR2015*, pages 98–102, Paris, France. Institut de Recherche en Musicologie.

Warren Enström, Josh Dennis, Brian Lynch, and Kevin Schlei. 2015. Musical notation for multi-touch interfaces. In Edgar Berdahl and Jesse Allison, editors, *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 83–86, Baton Rouge, Louisiana, USA, May 31 – June 3. Louisiana State University.

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

Dominique Fober, Yann Orlarey, and Stephane Letz. 2012a. Inscore – an environment for the design of live music scores. In *Proceedings of the Linux Audio Conference – LAC 2012*, pages 47–54.

Dominique Fober, Yann Orlarey, and Stéphane Letz. 2012b. Scores level composition based on the guido music notation. In ICMA, editor, *Proceedings of the International Computer Music Conference*, pages 383–386.

Dominique Fober, Stéphane Letz, Yann Orlarey, and Frederic Bevilacqua. 2013. Programming interactive music scores with inscore. In *Proceedings of the Sound and Music Computing conference – SMC'13*, pages 185–190.

Dominique Fober, Guillaume Gouilloux, Yann Orlarey, and Stéphane Letz. 2015. Distributing music scores to mobile platforms and to the internet using inscore. In *Proceedings of the Sound and Music Computing conference – SMC'15*, pages 229–233.

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.

Richard Hoadley. 2012. Calder's violin: Real-time notation and performance through musically expressive algorithms. In ICMA, editor, *Proceedings of International Computer Music Conference*, pages 188–193.

Richard Hoadley. 2014. December variation (on a theme by earle brown). In *Proceedings of the ICMC/SMC 2014*, pages 115–120.

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.

Cat Hope, Lindsay Vickery, Aaron Wyatt, and Stuart James. 2015. The decibel scoreplayer - a digital tool for reading graphic notation. In Marc Battier, Jean

Bresson, Pierre Couprie, Cécile Davy-Rigaux, Dominique Fober, Yann Geslin, Hugues Genevois, François Picard, and Alice Tacaille, editors, *Proceedings of the First International Conference on Technologies for Music Notation and Representation - TENOR2015*, pages 58–69, Paris, France. Institut de Recherche en Musicologie.

Gabriel Lepetit-Aimon, Dominique Fober, Yann Orlarey, and Stéphane Letz. 2016. Inscore expressions to compose symbolic scores. In Richard Hoadley, Chris Nash, and Dominique Fober, editors, *Proceedings of the International Conference on Technologies for Music Notation and Representation - TENOR2016*, pages 137–143, Cambridge, UK. Anglia Ruskin University.

Tom Mays and Francis Faber. 2014. A notation system for the karlax controller. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 553–556, London, United Kingdom, June. Goldsmiths, University of London.

Ryan Ross Smith. 2015. An atomic approach to animated music notation. In Marc Battier, Jean Bresson, Pierre Couprie, Cécile Davy-Rigaux, Dominique Fober, Yann Geslin, Hugues Genevois, François Picard, and Alice Tacaille, editors, *Proceedings of the First International Conference on Technologies for Music Notation and Representation - TENOR2015*, pages 39–47, Paris, France. Institut de Recherche en Musicologie.