SYMBOLIST: AN OPEN AUTHORING ENVIRONMENT FOR USER-DEFINED SYMBOLIC NOTATION

Rama Gottfried
CNMAT, UC Berkeley, USA
IRCAM, Paris, France / ZKM, Karlsruhe, Germany
HfMT Hamburg, Germany
rama.gottfried@berkeley.edu

Jean Bresson
IRCAM – CNRS – Sorbonne Université
UMR STMS, Paris, France
jean.bresson@ircam.fr

ABSTRACT

We present SYMBOLIST, a graphic notation environment for music and multimedia. SYMBOLIST is based on an Open Sound Control (OSC) encoding of symbols representing multi-rate and multidimensional control data, which can be streamed as control messages to audio processing or any kind of media environment. Symbols can be designed and composed graphically, and brought in relationship with other symbols. The environment provides tools for creating symbol groups and stave references, by which symbols maybe timed and used to constitute a structured and executable multimedia score.

1. INTRODUCTION

Contemporary art and music productions frequently rely on automated computer processes with huge sets of data and control parameters; and as in other large-scale data-driven situations, the authoring tools, storage and performance of the data are key design factors which have a marked influence on the aesthetic framework used to compose the artwork [1, 2]. Unlike pen and paper, commercial software authoring tools have been designed based on a set of use-cases and decisions about the composition format and rendering, selected and put forward by different actors in their development process. This situation prompts the question: If tools are a shaping factor in art production, how should authoring environments for artistic production be designed? In what ways can a computational process or mechatronic movement be represented in a score so that it is freely "composable", without presupposing a specific use context, or grammar?

While computational tools for creating and parsing symbolic graphic information are readily available, composition environments which support visualizing, editing, and synchronously executing multimedia control data streams are few to none. There exist no actual notational convention on how to represent control data for computerized automation systems [3, 4]. In electronic music production, most often the "score" is authored in a *digital audio workstation* (DAW) with MIDI note events and breakpoint function automations; while in theater, a *show control* system is typically used to step through a series of cues which send control messages to stage and lighting mechanisms. These tools have proved useful through their longevity over 30+ years, however as compositional frameworks, they proscribe specific ways of thinking about data. Breakpoint function automation works well for situations where you want to control *one* parameter over time, but in multivariate situations, for example spatial location where a position is a vector $\{x, y, z\}$, splitting the values into three separate automation lanes obscures the meaning of the values. [1] In contrast, a well designed symbolic notation could allow users to represent many parameters simultaneously [5].

The SYMBOLIST project addresses these issues by providing composers and media artists with a context-free environment for the authoring of graphical symbolic notation, with tools for displaying, editing and generating arbitrary streams of OSC-encoded data. After a general presentation of the project (Section 2), we will describe the design features and user interface of the software (Section 3), and then detail the execution mechanisms behind its score structure (Section 4). In continuation we will present some use cases and integration in host environments (Section 5), and conclude with an open discussion and some considerations about future work directions (Section 6).

2. FOUNDATIONS

SYMBOLIST was designed to address the practical need of visually representing parameters of electronic performances involving dense streams of control data, first conceived in the context of composing for spatial audio systems [6, 7]. High-dimensional symbolic representation is common in contemporary instrumental writing, and so for many composers it is intuitive to also apply symbolic notation approaches to new kinds of "multimedia instruments".

A first working prototype was implemented using Scalable Vector Graphics (SVG) authored with graphic design software (Adobe Illustrator), which could then be interpreted and performed as a stream of OSC data (Open Sound Control [8]) in the Max environment [9]. By leveraging the tools of a professional graphic design program in connection with the widely supported networking capabilities of OSC, the SVG-OSC project [10] provided a functional model of how graphic objects could be labeled and grouped

[1] The mathematical representation has the same perceptual problem.

semantically in order to be processed by an interpretive engine and used to control multimedia renderers. Building on the SVG-OSC project research, SYMBOLIST integrates the editing and semantic assignment functions into a single workspace specifically designed for maximum flexibility, through a minimum number of predefined object definitions.

SYMBOLIST considers a *score* as a structured set of graphical symbols, where each *symbol* (basic, or compound group of symbols) exists internally as an OSC *bundle* (i.e. a set of OSC messages describing a consistent data structure) which potentially includes both graphical attributes and other musical or control parameters. Although structured on the surface through staves, groupings and nested symbols (as we will see in the next section), the score is therefore viewed (and stored) as a simple, flat and *executable* sequence of OSC bundles.

The SYMBOLIST environment was implemented as a C++ application and built using the Juce framework. [2] It can run as a standalone editor or as an embedded component in another programming environment such as Max (where it constitutes a persistent container — a score — to display, edit and monitor control data streams) or OpenMusic [11] (where scores can be generated and processed through visual programs and algorithms).

3. WORKING IN SYMBOLIST

From the user point of view, the current SYMBOLIST prototype essentially implements a set of utilities for symbol authoring and composition following standard vector-graphic editing techniques.

Symbols. Graphical symbols and their associated semantics are defined by the user through interactive graphic and text-based OSC editing tools. Figure 1 shows a sample view of the main SYMBOLIST window. The left sidebar displays a number of default atomic symbol models (circle, rectangle, triangle, text characters...) which the user can pick and use as templates for the creation of symbols in the score page. On Figure 1, a single, big triangle symbol was added to the score. Score symbols are editable interactively using standard graphic transforms (translation, scaling, rotation, copy/paste, etc.). Their attributes may also be edited directly in the inspector view at the right of the window.

As mentioned above, each symbol is stored as an OSC bundle (i.e. a set of OSC messages), which reflects the set of attributes visible on the inspector view. The basic attributes shared by all symbols are: the *name*, symbol *type*, position (*x*, *y*), size (*w*, *h*), *color*, *staff* assignment, and *id*, a unique identifier of the symbol within the score. [3] Symbols may also include additional attributes. For example, the triangle symbol in Figure 1 includes *fill*, *stroke thickness*, and *rotation* attributes. The listing below displays the OSC representation corresponding to this symbol.

[2] https://juce.com/

[3] By default the *name* value is same as the *type*, and the *id* is the *name* followed by a unique instance number. Once a user-defined *name* is given, the *id* is updated.

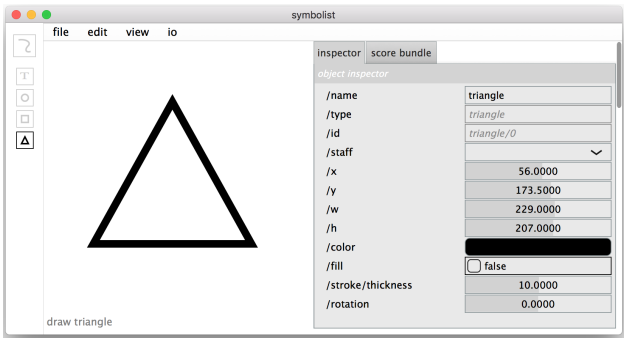


Figure 1. A single triangle symbol in the SYMBOLIST window. The inspector on the right side displays the attribute values of the symbol.

```
{
        /name : "foo",
        /type : "triangle",
        /id : "foo/0",
        /staff : "",
        /x : 47.,
        /y : 134.5,
        /w : 123.,
        /h : 120.,
        /color : [0., 0., 0., 1.],
        /fill : 0,
        /stroke/thickness : 2.,
        /rotation : 0.
}
```

Custom shapes can be drawn and edited using control point handles, and are encoded as *paths*, defined as a sequence of linear, quadratic or cubic bézier curve segments (see Figure 2). The SVG standard is used for storing path drawing commands in string format [12].

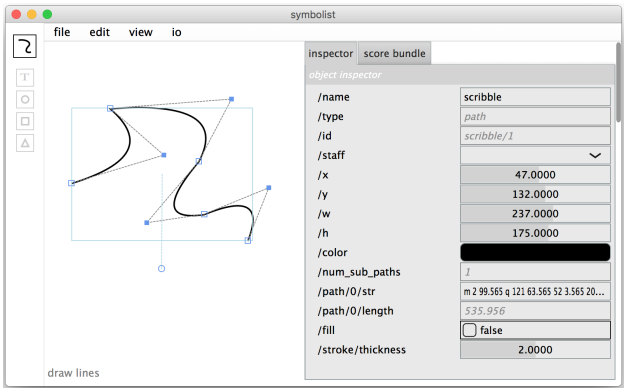


Figure 2. Drawing a custom (*path*) symbol.

The symbol in Figure 2 is represented in OSC as follows:

```
{
    /name : "scribble",
    /type : "path",
    /id : "scribble/1",
    /staff : "",
    /x : 33.,
    /y : 83.,
    /w : 237.,
    /h : 175.,
    /color : [0., 0., 0., 1.],
    /path/str : "m 2 99.565 q 121 63.565 52 [...]",
    /path/length : 535.956,
    /fill : 0,
    /stroke/thickness : 2.
}
```

Templates. Any symbol in the score can be turned into a template via a simple keyboard shortcut. Newly created templates appear in the symbol palette of the left sidebar (see Figure 3). They can then be stored in the application data and potentially shared between scores and projects.

In addition to the set of atomic symbols mentioned previously in this section, user-defined template symbols may therefore be selected and copied anywhere in the score as a new symbol, with all of the same editing and transformation possibilities.

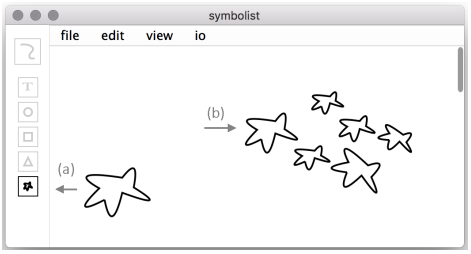


Figure 3. (a) Storing a user symbol as template in the SYMBOLIST palette toolbar (left side of the window). (b) Using this template as a model for creating new symbols.

Compound symbols can be created by graphical composition of simpler ones, through the *grouping* command. Symbols (atomic, custom, or compound) selected for grouping are gathered and converted into a single new symbol (see Figure 4), which can then be positioned, edited, transformed individually, and/or turned into a template in the symbol palette.

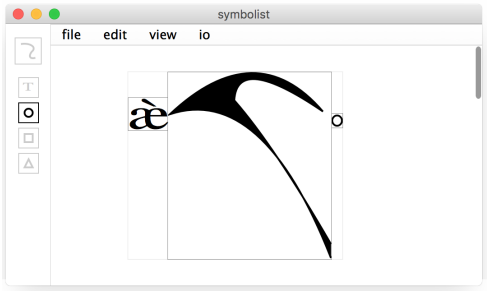


Figure 4. Grouping symbols.

Grouping is a hierarchical operation of unlimited depth and complexity. After grouping, sub-group symbols can still be accessed, recomposed and edited individually at any time, using simple user operations to step through the hierarchy of compound symbols. Below is an excerpted example of OSC representation of a SYMBOLIST *group* symbol, corresponding to the symbol in Figure 4:

```
{
    /name : ”group”,
    /type : ”group”,
    /id : ”group/0”,
    /staff : ””,
    /x : 78.,
    /y : 46.,
    /w : 240.,
    /h : 209.,
    /color : [0., 0., 0., 1.],
    /numsymbols : 3,
    /subsymbol/1/name : ”path”,
    /subsymbol/1/type : ”path”,
    /subsymbol/1/id : ”path/0”,
    /subsymbol/1/staff : ””,
    /subsymbol/1/x : 45.,
    /subsymbol/1/y : 0.,
    [...]
    /subsymbol/2/name : ”text”,
    /subsymbol/2/type : ”text”,
    /subsymbol/2/id : ”text/0”,
    /subsymbol/2/staff : ””,
    /subsymbol/2/x : 0.,
    /subsymbol/2/y : 51.5,
    [...]
    /subsymbol/3/name : ”circle”,
    /subsymbol/3/type : ”circle”,
    /subsymbol/3/id : ”circle/0”,
    /subsymbol/3/staff : ””,
    /subsymbol/3/x : 225.,
    /subsymbol/3/y : 54.5,
    [...]
}
```

Staves and score structure. In order to structure symbols into a temporal score, *staff* symbols can be created from any existing symbol (simple or compound). A *staff* symbol is considered as a reference which can be used for global manipulations and creation of polyphonic scores. It is wrapped in a special *staff* OSC bundle, and is automatically assigned time values (see Figure 5).

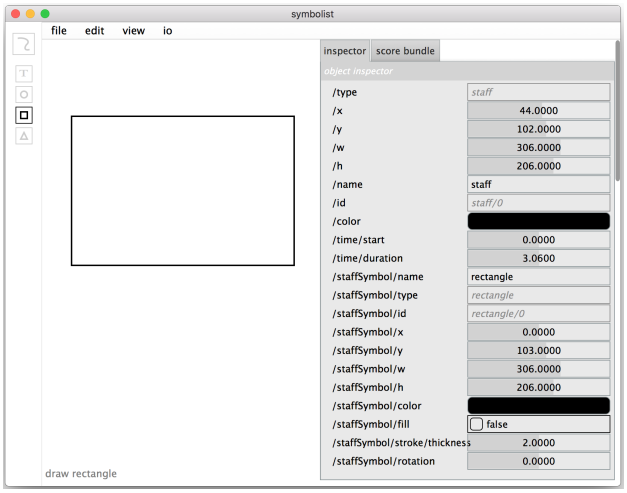


Figure 5. Converting a symbol to a *staff*.

Any symbol can be attached to a stave by setting the */staff* attribute to link to an existing *staff* symbol *id* value. All non-*staff* type symbols include the */staff* attribute in their corresponding OSC bundle. Once a symbol has been linked to a *staff*, this symbol becomes *timed*: it is given a start and duration, to its position and size relative to the stave origin and the stave numbering (see Figure 6).

Stave start and duration values are currently determined by their sequential order on in the score, also following traditional stave system format, reading left to right in lines down the page, and then continuing at the top of the following page. For traditional left to right, top to bottom reading, the symbol's start time and end times are calculated using the left and right edges of the object's bounds. [4]

[4] In the future, we envisage time direction could be a user-definable parameter in the score, for example to facilitate the use of Labanotation [13] or other types of graphic time arrangements such as trajectories through the score, and so on.

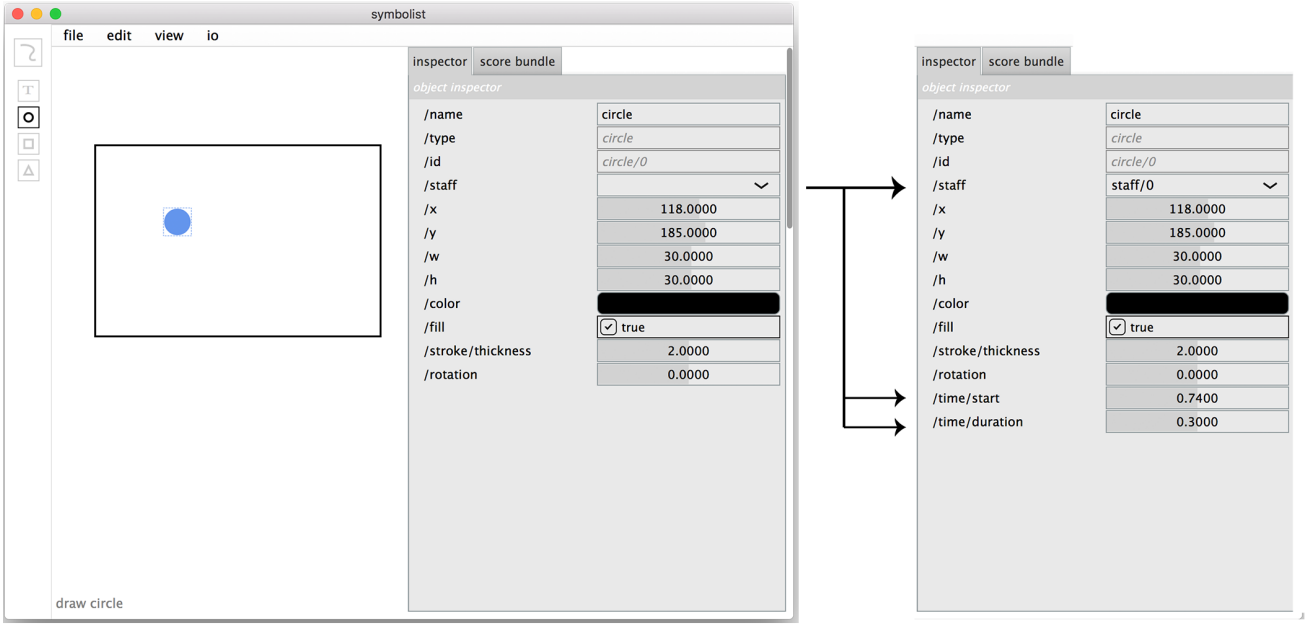


Figure 6. After attaching a symbol to a staff, time information is added to the symbol's OSC bundle.

Depending on the context, other internal parameters could also be effected by the symbol's horizontal and vertical coordinates in the stave reference, for example effecting pitch values for notes on traditional scores.

4. TIME AND SCORE PERFORMANCE

Through the creation of time relationships between symbols and staves, the score becomes "executable", or "performable". As described above, staves are the key temporal marker for score performance: they embed a time referential and a time-map allowing the computation of absolute time from relative graphical distances.

For the performance of the score, SYMBOLIST provides methods for outputting control values as OSC, and includes visual feedback information such as highlighted display for play-heads or cursors, etc.

SYMBOLIST actually does not include its own scheduling engine, but functions by responding to external time requests — e.g. from host environments — in order to retrieve the active symbol(s) at a given time. In response to a time location query, SYMBOLIST outputs an OSC bundle containing the values of all symbol "events" existing at that time in the score (see Figure 7).

To aid with mapping, the output OSC bundle is formatted using the symbol's *name* attribute as user defined identifier. For example in Figure 7, the staff name is "foo" and the group symbol name is "glissnote", which contains "glissando" and "notehead" sub-symbols. Whereas the score is a flat array of symbols/bundles, the contents of the output bundle are formatted in a hierarchical representation, where events are located in the OSC namespace of their associated stave. For example, in Figure 7 note that the active voices in the bundle, are in prefixed by */staff/foo*.

Each event is output with the relative time position within the symbol called the */time/ratio*, where 0 is the beginning of the symbol and 1 is the end.

To assist in handling overlapping polyphonic symbols, which may start and stop independently, a *voice* identifier is assigned to each symbol which stays constant between lookup queries. A */state* value is also provided which identifies the symbol's status: 1 for a new *voice*, 0 when it is continuing from the last lookup, and -1 to identify when a *voice* is no longer present, which can be used for "note off" messages.

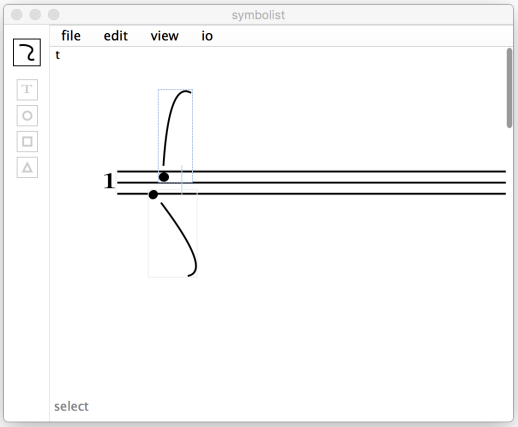
Symbols can also have internal timing and time referential — for example imagine a curve, or another graphic symbol which could represent the evolution of one or several parameters over a given amount of time. In *path*-symbols the relative time position is used to lookup the $\{x, y\}$ location on the path, output at the address */lookup/xy*.

For example, in Figure 8, a "frame notation" is used to control the spatialization of a sequence of events. In this case, a compound symbol is used, consisting of: (1) a 2D spatial region defined by the rectangle frame, (2) a *path* depicting a trajectory moving through the 2D space, and (3) a horizontal line which is used to define the duration of the symbol on the stave. The circle symbols below are sound events which are positioned using the frame notation above.

In order to optimize the processing of the time requests (which can occur at a relatively high rate in playback or score execution contexts), an internal "time-point array" is constructed and maintained along with score editing operations, which stores a sorted reference map of the score symbols' start and end points.

5. HOST ENVIRONMENTS

SYMBOLIST currently exists as a standalone application, and as a static or dynamic library. The main entry points of the application programming interface (API) are read/write




```
{
   /time/lookup : 0.7,
   /time/end : 0.89,
   /staff/foo/voice/1/glissnote/state : 0,
   /staff/foo/voice/1/glissnote/time/ratio : 0.631579,
   /staff/foo/voice/1/glissnote/name : "glissnote",
   /staff/foo/voice/1/glissnote/type : "group",
   /staff/foo/voice/1/glissnote/id : "glissnote/1",
   /staff/foo/voice/1/glissnote/staff : "foo/palette",
   /staff/foo/voice/1/glissnote/x : 46.,
   /staff/foo/voice/1/glissnote/y : −87.,
   [...]
   /staff/foo/voice/1/glissnote/numsymbols : 2,
   /staff/foo/voice/1/glissnote/time/start : 0.46,
   /staff/foo/voice/1/glissnote/time/duration : 0.38,
   /staff/foo/voice/1/glissnote/subsymbol/1/notehead/name : "notehead",
   /staff/foo/voice/1/glissnote/subsymbol/1/notehead/type : "circle",
   /staff/foo/voice/1/glissnote/subsymbol/1/notehead/x : 0.,
   [...]
   /staff/foo/voice/1/glissnote/subsymbol/2/glissando/name : "glissando",
   /staff/foo/voice/1/glissnote/subsymbol/2/glissando/type : "path",
   /staff/foo/voice/1/glissnote/subsymbol/2/glissando/x : 4.,
   [...]
   /staff/foo/voice/1/glissnote/subsymbol/2/glissando/lookup/xy : [0.268815, 0.234897],
   /staff/foo/voice/0/glissnote/state : 0,
   /staff/foo/voice/0/glissnote/time/ratio : 0.648148,
   /staff/foo/voice/0/glissnote/name : "glissnote",
   /staff/foo/voice/0/glissnote/type : "group",
   /staff/foo/voice/0/glissnote/id : "glissnote/2",
   /staff/foo/voice/0/glissnote/staff : "foo/palette",
   [...]
   /staff/foo/voice/0/glissnote/numsymbols : 2,
   /staff/foo/voice/0/glissnote/time/start : 0.35,
   /staff/foo/voice/0/glissnote/time/duration : 0.54,
   /staff/foo/voice/0/glissnote/subsymbol/1/notehead/name : "notehead",
   /staff/foo/voice/0/glissnote/subsymbol/1/notehead/type : "circle",
   [...]
   /staff/foo/voice/0/glissnote/subsymbol/2/glissando/name : "glissando",
   /staff/foo/voice/0/glissnote/subsymbol/2/glissando/type : "path",
   [...]
   /staff/foo/voice/0/glissnote/subsymbol/2/glissando/lookup/xy : [0.657142, 0.553749]
}
```

Figure 7. An example SYMBOLIST OSC output stream for a time point containing multiple timed symbols.

accessors which allow to build, store, process the score symbols in host environments, and perform time point look-up as described above in Section 4. All the data is transferred back and forth through OSC-encoded bundles. Two main host environment are currently supported.

Max. SYMBOLIST was embedded in an object for the Max environment [9], where the score editor can be used to store, generate and monitor timed streams of data (see Figure 8). Score readers can be easily implemented to browse through the score via time requests which output the corresponding symbols and associated data.

OpenMusic. SYMBOLIST was also integrated in the O7 prototype implementation of the OpenMusic computer-aided composition environment [11, 14]. OpenMusic programs can generate scores (sequences of OSC bundles representing staves and timed symbols), which can be connected to interactive, personalized graphical display and editing (see Figure 9). SYMBOLIST in this context offers alternative graphical representations for musical data reaching far beyond the expressive potential of traditional music notation editors or more neutral automation controllers. The editor here also can be easily connected to OpenMusic's embedded scheduling engines through the timed-request function of the SYMBOLIST API, which allows the score be "played", just as any other musical object of the environment, via timed transfer of OSC data.

6. DISCUSSION: TOWARDS EMBEDDED SCORES

An important challenge to be considered for the SYMBOLIST framework is how, if possible, to integrate these new notation tools with the current predominant practices in media art programming. Interactive computer-music and multimedia artists often make use of programming environments such as Max, Pure Data, SuperCollider, Processing, Arduino, Grasshopper, Blender, VVVV, OpenFrameworks, et al., where the compositional thought is directly integrated into the program that renders or performs the work. In these cases, the artist composes the piece directly in the code itself, embedding the artistic intention into the computational process which produces the piece [15].

For example in the "circuit scores" of David Tudor, the "composition" takes the form of an instrument: the instrument's behavior is composed as the result of an interaction between electronic components [16]. Chadabe, di Scipio, Leman, Wessel, and others have discussed this embedded nature of artistic intentions in interactive instrument systems, and its relation to cybernetics, systems theory, and embodied cognition studies [17, 18, 19, 20, 21, 2]. This merging of "instrument" and "composition" can also be observed in the "process scores" of Cage, Feldman, Stockhausen, et al., and all the way back to the *Musikalisches Würfelspiel* pieces by Mozart and Kirnberger, where the score describes a sequence of musical-cognitive processes which led to the production of the piece, rather than describing the results themselves.

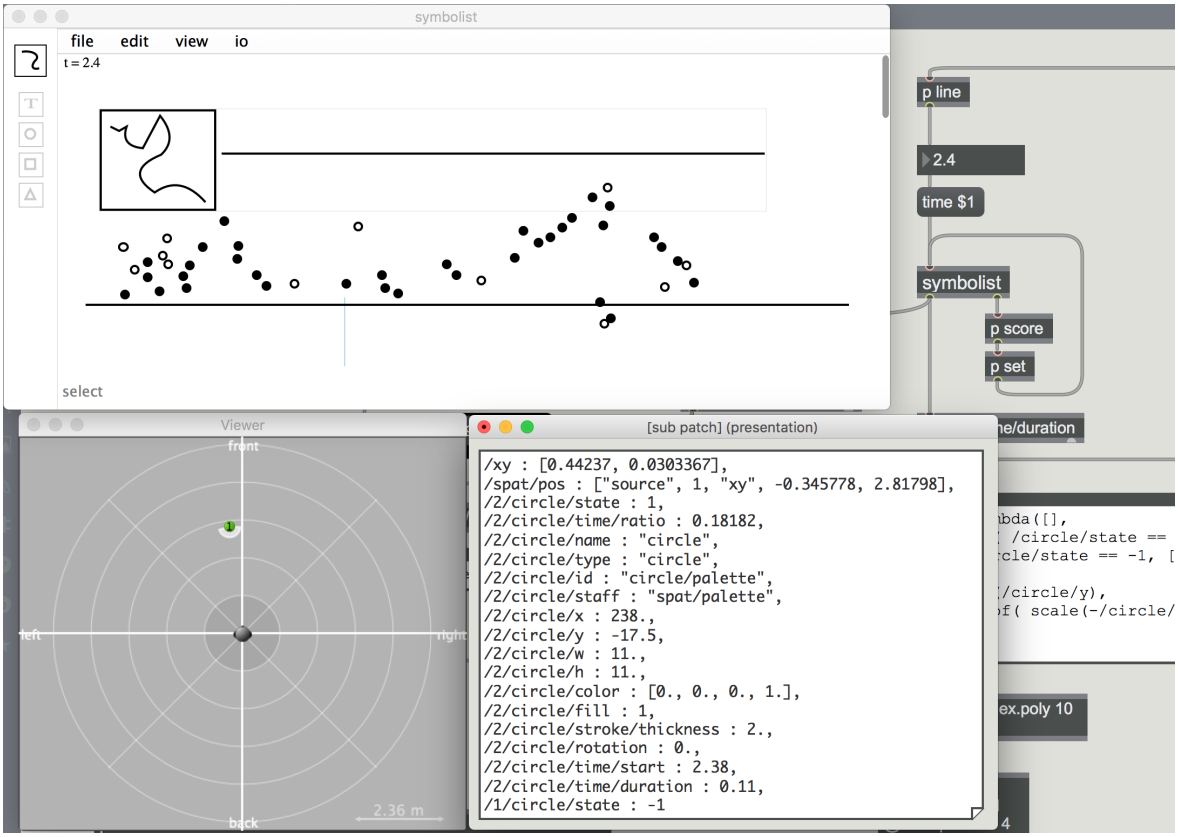


Figure 8. An example using SYMBOLIST in Max. Sending a time value into the SYMBOLIST Max object causes output of an OSC bundle containing the symbol values at that time-point. Spatial location is composed with a frame notation symbol group where the frame represents a given region in space, and the path is the trajectory distributed over the time of the horizontal line. Separately, circular symbols are used to notate sound events.

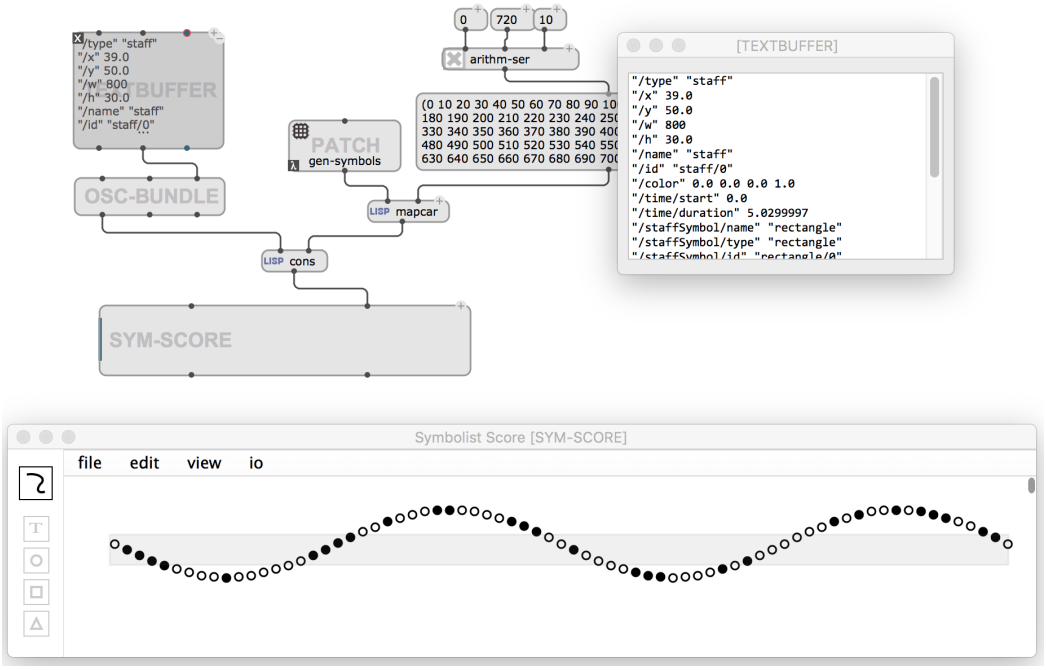


Figure 9. SYMBOLIST integration in OpenMusic (O7). OSC bundles describing symbols are written and generated algorithmically in the computer-aided composition environment (here to produce a sine-shaped sequence of small circle symbols), then displayed and edited in the SYMBOLIST editor.

In all of the above historical examples, the scores were notated to be read and performed by *humans*, which naturally requires them to be readily understood and interpreted by humans in terms of their learned and embodied cultural knowledge. This is no longer necessarily the situation when working with digital performance systems. Control parameters for digital processes need to be in a *computation*-friendly format which can be parsed and interpreted by the *program*, which invisibly transforms the algorithm and score into executable machine-code.

In this context where compositional processes are embedded into an interactive system, there is rarely a "score" separate from the instrument itself. This may well be the most natural approach for this situation, where the code, the instrument, and the score are all intertwined. However, it may be limiting as well, since as we discussed in the introduction, the affordances of a system have a strong influence on the uses of the system. A number of recent projects, such as INScore [22], *bach* [23], or PWGL's Expressive Notation Package (ENP) [24], are similarly navigating this hybrid zone between score and programmatic media generation. The question then for SYMBOLIST is, in what ways could notation function within the context of the embedded score?

A potential route of development could be to include interpretive expressions inside a symbol's OSC bundle which could be evaluated at performance time. Computational expressions could be composed in SYMBOLIST, either symbolically or as text, which could then be transcoded into another environment. The *odot* expression language would be a natural choice since it is specifically designed to operate on OSC messages and is well suited to transcoding between applications [25]. In the simplest case, a symbol could include anonymous functions which when evaluated would map the symbol data to the target rendering system format (spatial audio system, video, motors, etc.).

Attaching expressions to symbols could also be a way for users to create their own custom designed interaction tools. In this case, the expression could be evaluated while editing within SYMBOLIST, to provide additional information relevant to the intended output context (e.g. contextual displays), or used to create interactive drawing tools which could generate other types of symbolic/graphic information.

7. CONCLUSION AND PERSPECTIVES

We presented the first prototype of SYMBOLIST, a software developed for visualizing, editing, and executing control data streams for music and media encoded as OSC bundles. The project was conceived in response to the lack of efficient tools currently available to perform these tasks, and to expand the possibilities for multimedia and electroacoustic scores, which, when they exist, are most often incomplete, non-executable and/or non-editable: there is generally little support to *symbolically notate* computerized music and media control material.

SYMBOLIST aims at completing contemporary artists' and composers' toolboxes with a simple tool used to realize and execute such multimedia scores, and joins a burgeoning landscape of computer platforms for computer aided composition and multimedia notation [22, 23, 26, 27, 28]. As compared to IanniX's 3D timeline orientation [28], or to advanced sequencing tools such as i-Score [29] or Antescofo's Ascograph editor [30], which provide advanced means to program and visualize timing and interactions, SYMBOLIST emphasizes symbolic, graphical drawing/editing for new music and media notation.

The OSC foundation for the SYMBOLIST score data structure is not an arbitrary choice: it is today an established and widely supported format used for media data encoding and interchange, and we believe in the potential for its future development — especially through CNMAT's *odot* library — to greatly improve the expressivity of our software functionality. The planned future work in this project will feature the integration of and embedded OSC server, in order to fully support interaction with external software, as well as advanced embedded expression programming in OSC-encoded symbols, as discussed in Section 6.

Other future work directions are to continue development on the graphical display and rendering of scores, through a number of features related to page formatting and layout, printing, export to graphical formats, etc. Finally, in order to constitute a fully-workable score environment, the software will need to embed the possibility to integrate, edit and merge common music notation with the user-defined staves and symbols of the SYMBOLIST scores.

Acknowledgments

This work was realized in the context of a joint artistic research residency at IRCAM and ZKM.

8. REFERENCES

[1] J. Greeno, "Gibon's Affordances," *Psychological Review*, vol. 101, no. 2, pp. 336–342, 1994.

[2] T. Magnusson, "Of Epistemic Tools: Musical instruments as cognitive extensions," *Organised Sound*, vol. 14, no. 2, pp. 168–176, 2009.

[3] M. Battier, "Describe, Transcribe, Notate: Prospects and problems facing electroacoustic music," *Organised Sound*, vol. 20, no. 1, pp. 60–67, 2015.

[4] K. Stone, "Problems and Methods of Notation," *Perspectives of New Music*, vol. 1, no. 2, pp. 9–31, 1963.

[5] E. R. Tufte, *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[6] R. Gottfried, "Studies on the Compositional Use of Space," IRCAM, Paris, France, Tech. Rep., 2013.

[7] T. Carpentier, N. Barrett, R. Gottfried, and M. Noisternig, "Holophonic Sound in IRCAM's Concert Hall: Technological and Aesthetic Practices," *Computer Music Journal*, vol. 40, no. 4, pp. 14–34, 2017.

[8] M. Wright, "Open Sound Control: an enabling technology for musical networking," *Organised Sound*, vol. 10, no. 3, pp. 193–200, 2005.

[9] M. Puckette, "Combining Event and Signal Processing in the MAX Graphical Programming Environment," *Computer Music Journal*, vol. 15, no. 3, pp. 68–77, 1991.

[10] R. Gottfried, "SVG to OSC Transcoding: Towards a Platform for Notational Praxis and Electronic Performance," in *Proceedings of the International Conference on Technologies for Notation and Representation (TENOR'15)*, Paris, France, 2015.

[11] J. Bresson, D. Bouche, T. Carpentier, D. Schwarz, and J. Garcia, "Next-generation Computer-aided Composition Environment: A New Implementation of OpenMusic," in *Proceedings of the International Computer Music Conference (ICMC'17)*, Shanghai, China, 2017.

[12] World Wide Web Consortium, "Scalable Vector Graphics (SVG) 1.1 (Second Edition)," *W3C Candidate Recommendation*, 2011. [Online]. Available: https://www.w3.org/TR/SVG11/

[13] A. H. Guest, *Labanotation: The System of aAalyzing and Recording Movement*. Routledge, 2005.

[14] J. Bresson, C. Agon, and G. Assayag, "OpenMusic: Visual Programming Environment for Music Composition, Analysis and Research," in *Proceedings of the ACM international conference on Multimedia – Open-Source Software Competition*, Scottsdale, AZ, USA, 2011, pp. 743–746.

[15] T. Magnusson, "Algorithms as Scores: Coding Live Music," *Leonardo Music Journal*, vol. 21, pp. 19–23, 2011.

[16] R. Kuivila, "Open Sources: Words, Circuits and the Notation-Realization Relation in the Music of David Tudor," *Leonardo Music Journal*, vol. 14, pp. 17–23, 2004.

[17] J. Chadabe, "Interactive Composing: An Overview," *Computer Music Journal*, vol. 8, no. 1, pp. 22–27, 1984.

[18] D. Wessel, "An enactive approach to computer music performance," in *Le Feedback – Acte des Rencontres Musicales Pluridisciplinaires*. Lyon, France: Grame, 2006, pp. 93–98.

[19] M. Leman, *Embodied Music Cognition and Mediation Technology*. MIT Press, 2008.

[20] A. Di Scipio, "'Sound is the interface': from interactive to ecosystemic signal processing," *Organised Sound*, vol. 8, no. 3, pp. 269–277, 2003.

[21] N. Schnell and M. Battier, "Introducing Composed Instruments, Technical and Musicological Implications," in *Proceedings of the conference on New Interfaces for Musical Expression (NIME'02)*. Dublin, Ireland, 2002.

[22] D. Fober, S. Letz, Y. Orlarey, and F. Bevilacqua, "Programming interactive music scores with INScore," in *Proceedings of the Sound and Music Computing conference (SMC'13)*, Stockholm, Sweden, 2013, pp. 185–190.

[23] A. Agostini and D. Ghisi, "A Max Library for Musical Notation and Computer-Aided Composition," *Computer Music Journal*, vol. 39, no. 2, pp. 11–27, 2015.

[24] M. Kuuskankare, "ENP: A System for Contemporary Music Notation," *Contemporary Music Review*, vol. 28, no. 2, pp. 221–235, 2009.

[25] J. MacCallum, R. Gottfried, I. Rostovtsev, J. Bresson, and A. Freed, "Dynamic Message-Oriented Middleware with Open Sound Control and Odot," in *Proceedings of the International Computer Music Conference (ICMC'15)*, Denton, TX, USA, 2015.

[26] N. Didkovsky and G. Hajdu, "MaxScore: Music Notation in Max/MSP," in *Proceedings of the International Computer Music Conference (ICMC'08)*, Belfast, Northern Ireland / UK, 2008.

[27] C. Hope, L. Vickery, A. Wyatt, and S. James, "The Decibel Scoreplayer – A digital tool for reading graphic notation," in *International conference on Technologies for Music Notation and Representation (TENOR'15)*, Paris, France, 2015.

[28] T. Coduys and G. Ferry, "Iannix-aesthetical/symbolic visualisations for hypermedia composition," in *Proceedings of the Sound and Music Computing conference (SMC'04)*, Paris, France, 2004, pp. 18–23.

[29] J.-M. Celerier, M. Desainte-Catherine, and J.-M. Couturier, "Graphical Temporal Structured Programming for Interactive Music," in *Proceedinsg of teh International Computer Music Conference (ICMC'16)*, Utrecht, Netherlands, 2016.

[30] G. Burloiu, A. Cont, and C. Poncelet, "A visual framework for dynamic mixed music notation," *Journal of New Music Research*, vol. 46, no. 1, pp. 54–73, 2017.