

SVG TO OSC TRANSCODING: TOWARDS A PLATFORM FOR NOTATIONAL PRAXIS AND ELECTRONIC PERFORMANCE

Rama Gottfried

Center for New Music and Audio Technologies (CNMAT)

University of California, Berkeley

rama.gottfried@berkeley.edu

ABSTRACT

In this paper we present a case study for the creation of an open system for graphically developing symbolic notation which can function both as professional quality print or online documentation, as well as a computer performable score in electro-acoustic music and other computer aided contexts. Leveraging Adobe Illustrator's graphic design tools and support for the Scalable Vector Graphics (SVG) file format, the study shows that SVG, being based on Extensible Markup Language (XML), can be similarly used as a tree-based container for score information. In the study, OpenSoundControl (OSC) serves as middleware used to interpret the SVG representation and finally realize this interpretation in the intended media context (electronic music, spatial audio, sound art, kinetic art, video, etc.). The paper discusses how this interpretive layer is made possible through the separation of visual representation from the act of rendering, and describes details of the current implementation, and outlines future developments for the project.

1. BACKGROUND

The twentieth century was a time full of notational experimentation and development. Due to the explosion of performance technologies, new materials entered the scope of composition that previously were considered outside the realm of "music." [1] Works dealing with complex rhythmic and microtonal inflections, chance, and stochastic processes each exposed new compositional parameters that were not easily addressed by traditional music notation. Developments in purely mechanical music production such as piano rolls, automata, optical synthesizers, and eventually digital audio workstations allowed composers to pre-

cisely sculpt the resulting sounds by manipulating the many parameters of sound creation. [2,3]

In some works, such as Stockhausen's *Plus-Minus*, Feldman's *Projection* series, Riley's *In C*, and others¹, the score is designed as a description of structural processes which require the performer to interpret the instructions and create their own performance score. In other works, such as Cardew's *Treatise*, the score contains no instruction, but only graphic symbols to be interpreted by the performer. Some, like Lachenmann, began to draw from tablature notation where the score describes the actions of the performers on their instruments rather than the results, while others like Kagel began to compose physical spatial movements. Similarly, in the dance world choreographers were composing movements based on the work of Laban and others [4].

Key to all of the above works is that they were all conceived and printed on paper and were designed to be read and performed by humans. Thus, the symbolic information contained in them is expressive to a human intelligence, who then performs their interpretation with an instrument or physical action. In the fields of electronic music, kinetic, video, and other types of mechanical and digital arts, the output of the instrument is via an electronically mediated system, or *rendering*, which often is unnotated [5]. With the ubiquity of powerful personal computers we now have immense rendering capabilities at our fingertips, along with many specialized tools to control these various media. These systems provide new ways for artists to incorporate many new types of media into their practice that were not previously available, however there remains a dearth of symbolic notation tools to compose with these new medias while still providing the richness of symbolic representation designed to be interpreted by a human intelligence.

The following study looks at what a more open framework for notation might look like for composing and performing scores designed for new and existing types of rendering contexts.

Copyright: ©2015 Rama Gottfried et al. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 Unported License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

¹ Including Mozart's *Musikalisches Würfelspiel*



Figure 1. Screenshot from Max for Live environment showing breakpoint functions used to control flocking behaviors of virtual sources in a piece using spatial audio. On the right side of the screen are OpenGL visualization of point locations generated by the many parameters contained within the algorithm. In a more symbolic notation environment, the parameters of the resulting rendering would be representational of their function.

2. CONTEXTUAL EXAMPLE: COMPOSING FOR SPATIAL AUDIO SYSTEMS

Mixed works for live acoustic instruments and real-time spatial audio rendering systems present significant challenges for relating the graphic visualizations of spatial processing with the traditional musical notations used in a score.

For example, Ircam’s “Spat”² MaxMSP³ library for spatial audio processing comes equipped with several useful forms of representation for visualizing spatial processing. The simplest visualization tools in Spat display the placement of a point source in two dimensions, viewed from either “top” (XY) or “front” (XZ) vantage points. The 2D representation makes the location of points very clear and minimizes occlusion issues. For 3D visualization Spat’s viewer may be easily integrated with OpenGL tools in Max’s Jitter library. These are valuable methods for visualizing and developing a conceptual basis for spatial design, but as interactive graphic user interfaces they are time-variant, and so do not provide a clear mechanism for relating spatial processing with score notated actions to be performed by the instrumentalist.

Part of this problem is that there is no widely adopted notation system for incorporating spatial movement within a musical score. Interactive UIs are an intuitive way to experiment and learn the expressive capabilities in new media contexts, however when seeking to compose temporal structures, time must be represented as well. Traditional music notation symbolically represents the articulation of

sound over metric time, and composers trained in this tradition learn to silently hear through the spatial organization of symbols on paper. Similarly, we might develop inner spatial perception by drawing from a long history of dance notation to describe spatial movement [4], which could be used to control a rendering system like Spat. What is needed is a symbolic graphic environment to explore ways of composing for these new types of media contexts, we have many new tools for controlling media, but very few ways of utilizing symbolic notation practice in these contexts.

2.1 Perceptual representations and breakpoint functions

Composition for spatial processing systems typically occurs in media programming environments or digital audio workstations, where the compositional approach must be contextualized within the types of controls provided by the software tools. As with interactive UIs, real-time processing is time-variant, and so the control parameters of a given process need to be contextualized in time if a score is desired⁴. This type of system has a natural affordance towards the triggers and breakpoint functions, which are extremely useful for fine control over the movement of one value (Y) over time (X).

This 2D representation, however, requires our spatial perception to be fragmented into three separate parameters (X-Y-Z or azimuth-elevation-distance). Working in this perforated situation the user must compose each param-

² <http://forumnet.ircam.fr/product/spat/>

³ <https://cycling74.com>

⁴ Keeping in mind that the score does not necessarily need to describe all events as “fixed” in time.

ter individually, so there is a natural tendency to focus on a smaller number of dimensions (e.g. a tendency to focus on azimuth over distance). This computationally friendly representation comes at the expense of a more intuitive data manipulation, which is always one step removed in univariate control over multi-dimensional spaces. Figure 1 shows an example of this, where many lines of automation are composed to describe the spatial behavior of the three dimensional space shown on the right.

The strength of a well-developed notation system is in the way layers of contextual meanings are signaled by a combination of symbols. For example, a staccato dot above a note head is immediately heard and physically felt in the mind of a musician. There is an interpretive act that accompanies a notation symbol that is bound up in a cultural history of practice and experience. This interpretive act may also be present for electronic musicians who have worked with breakpoint functions for many years, however, the breakpoint function is fundamentally a concrete control over a *single* parameter as a function of time, where a symbolic representation is an aggregate of *many* parameters, functioning through abstract, contextual implications for how it should be interpreted.

In order to take advantage of the expressivity contained in symbolic notation into other media contexts, we need a way to experiment with different notational systems and strategies outside of music notation.

3. WHY NOT USE MUSIC NOTATION SOFTWARE?

Software has built-in affordances that simplify certain uses, while making other approaches more difficult [6]. As music notation has become increasingly digitalized over the last 20 years, software applications designed for music notation have also become increasingly specialized tools focusing on a specific context at the exclusion of others. Music notation software tools expose the author(s)'s idea of what "music" is, through the types of functions they provide their users.

The most used music notation programs, Finale⁵, LilyPond⁶, NoteAbility⁷, and Sibelius⁸, all target the production of traditional music scores, and also provide mechanisms for MIDI playback of these scores. Many of these applications provide APIs for manipulating the musical pa-

rameters computationally, in particular LilyPond's Scheme based scripting language.

As we have been discussing, traditional notation was not designed to handle sound's relationship to advanced instrumental techniques, spatial audio, dance, installation art, and so on. Most traditional notation software has predefined interpretations of the symbolic information contained in the score, a prime example being the ubiquitous MusicXML⁹ format, which is designed specifically for "music," and so does not provide an optimal encoding for symbolic notation for contexts traditionally thought of as outside "music." While most of the above musical notation programs *do* allow the user to create custom symbols, the user is bound to an underlying assumption that the score is either to be read only by humans or to be performed as MIDI within a specifically musical context.

Computer aided composition tools such as Abjad¹⁰, Bach¹¹, INScore¹², MaxScore¹³, OpenMusic¹⁴, and PWGL¹⁵, provide environments for algorithmically generating musical scores, as well as providing connectivity to the types of new media outputs mentioned above. However, these tools rely on text input or visual programming, requiring the artist to formalize their thought process to function within the confines of a computational structure. In some cases basic drawing tools are available, however they are limited in flexibility.

Other experimental notation programs (e.g. GRM's Acousmographie [7], IanniX¹⁶, MaxScore's Picster¹⁷, Pure Data's Data Structures [8], etc.) provide new ways of performing graphic information, but they also contain symbolic limitations, which are not found in a graphic design environment, either through a forced method of playback, or through a limitation of graphic flexibility. Thus, at the moment, purely graphic design tools seem to provide a more flexible option for developing – and composing with – appropriate notation systems. For this reason, many contemporary composers use Adobe Illustrator for creating their scores.

The UPIC (Unit Polyagogique Informatique CEMAMu) project was one of the first to connect the act of human drafting with digital sound resources [9]. This integration of the drawing gesture is related to the working method in discussion here, the design of the UPIC was however very much tied to specific rendering contexts (amplitude

⁵<http://www.finalemusic.com>

⁶<http://www.lilypond.org>

⁷<http://debussy.music.ubc.ca/NoteAbility> – note: NoteAbility provides some support for communication with MaxMSP through the use of breakpoint functions and qlist max messages, as well as an option to export to Antescofo (<http://repmus.ircam.fr/antescfofo>) score following format. This is useful for traditional music, but does not solve the problem of symbolic notation of the processes occurring in Max.

⁸<http://www.sibelius.com>

⁹<http://www.musicxml.com>

¹⁰<http://abjad.mbrsi.org>

¹¹<http://www.bachproject.net>

¹²<http://inscore.sourceforge.net>

¹³<http://www.computermusicnotation.com>

¹⁴<http://repmus.ircam.fr/openmusic/home>

¹⁵<http://www2.siba.fi/PWGL>

¹⁶<http://www.iannix.org>

¹⁷http://www.computermusicnotation.com/?page_id=314

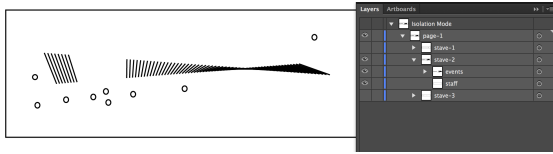


Figure 2. An example of using Adobe Illustrator’s grouping mechanism to create a hierarchy of graphic data

envelopes, waveforms, etc.) and so required specific kinds of symbolic composition, which make it not extendable to other types of interpretation.

3.1 Sketching and babbling

The recent MusInk [10] and InkSplore [11] projects have shown that Livescribe¹⁸ pen technology may also be a way to connect symbolic thinking on paper with digital rendering capabilities. The MusInk project also provides the capability to assign a type to an arbitrary symbol, which is closely related to the present study, however since these Livescribe projects are designed for paper, they forgo some of the possibilities offered by graphic design environments. These studies point to the importance of sketching in developing new graphic ideas.

Sketches are by definition incomplete, and provide the mind with an image to reflect on and continually refine through iteration [12]. David Wessel describes this type of *enactive* engagement in his discussion of *babbling* as a method for free experimentation in sensory-motor development in language and instrument learning, leading towards the development of a “human-instrument symbiosis” [13]. Such a symbiosis should also be possible with symbolic thought and computer controlled rendering systems.

3.2 Performing digital graphic scores

Graphic design applications like Adobe Illustrator, InkScape, OmniGraffle are created to have the basic affordances of a drafting table: a piece of paper, pen, stencils, and ruler – with the end goal of creating publication ready documents. There are no built-in musical functions, no button for transposition, no MIDI playback, etc. What these applications provide are the basic tools for visually creating whatever it is that you want to draw, generally in two dimensions. The user is left to decide what the meaning of the graphics might be.

Composers who choose to work in graphic design programs rather than music notation programs are silently stating that they do not expect to be able to render their score with the computer in the way that a typical musical notation program will have built in MIDI playback. Rather, it

is implied that they accept that due to software constraints their work is graphic, and either meant to be performed only by humans who will be able to interpret the score, or that the score is a descriptive notation of electronic results rather than proscriptive notation of how to perform the material. However this does not need to be the case. As a preliminary study implementation, the SVG output of Illustrator was used as a container for performable graphic information, leveraging Illustrator’s layer panel as a control for hierarchical grouping.

4. IMPLEMENTATION

Scalable Vector Graphic (SVG)¹⁹ is an XML-based open standard developed by the World Wide Web Consortium (W3C) for two dimensional graphics. In addition to being widely supported in software applications, the SVG Standard provides several key features that make it an attractive solution for digital graphic notation: (1) it is human readable which makes it easy to open an SVG file in a text editor and understand how the data is structured; (2) the SVG format provides a palette of primitive vector shapes that are the raw building blocks for most notations (and also provides tags for adding further types); (3) inheriting XML’s tree structure model, SVG provides standardized grouping and layer tags allowing users to create custom hierarchies of graphic objects; and (4) the header for SVG files includes the canvas information for contextualizing the content of the file.

In this paper, we propose replacing the graphics renderer with a new type of rendering interpretation, be it sonic, spatial, kinetic, or any other possible output type. Thought of this way, the SVG file functions as hierarchical input data, to be rendered, or performed by an electronic instrument. In our implementation, OpenSoundControl (OSC) [14] serves as a transcoding layer used for processing an interpretation of the SVG file structure.

4.1 SVG → Odot → Performance

As a first test to interpret and perform the SVG score within the Max environment, the LibXML2²⁰ library was used to parse the SVG file created in Adobe Illustrator (figure 2), and convert the SVG tree (figure 3), into an OSC bundle (figure 4). For convenience, this was implemented in C and put in a Max object called “o.svg”. The SVG graphic data was then reformatted, and interpreted for performance utilizing the “odot” OSC expression language developed at CNMAT over the last few years [15].

Based on the OSC protocol, CNMAT’s new odot library provides tools for handing OSC bundles as time-synchronized

¹⁸ <http://www.livescribe.com>

¹⁹ SVG Standard: <http://www.w3.org/TR/SVG>

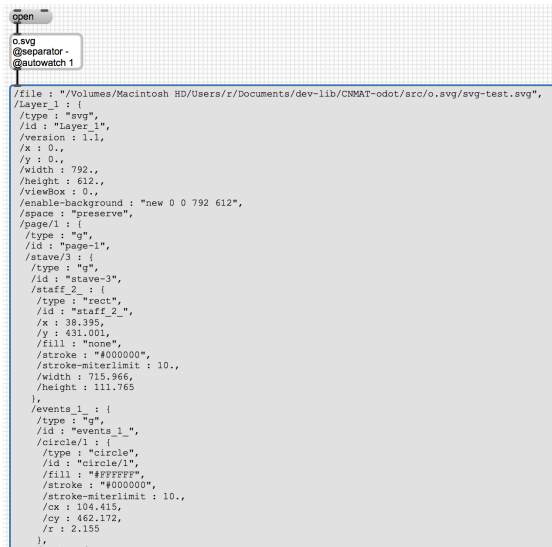
²⁰ <http://xmlsoft.org>

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <!-- Generator: Adobe Illustrator 16.0.4, SVG Export Plug-In . SVG Version: 6.00 Build 0) -->
3 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
4 <svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
5 width="792px" height="612px" viewBox="0 0 792 612" enable-background="new 0 0 792 612" xml:space="preserve">
6 <g id="page-1">
7 <g id="stave-1">
8 <rect id="staff" x="38.395" y="249.535" fill="none" stroke="#000000" stroke-miterlimit="10" width="715.966" height="111.765"/>
9 <g id="events">
10 <line fill="none" stroke="#000000" stroke-miterlimit="10" x1="72.605" y1="287.212" x2="83.5" y2="314.356"/>
11 <line fill="none" stroke="#000000" stroke-miterlimit="10" x1="75.656" y1="287.444" x2="86.09" y2="314.146"/>
12 <line fill="none" stroke="#000000" stroke-miterlimit="10" x1="78.708" y1="287.677" x2="88.682" y2="313.935"/>
13 <line fill="none" stroke="#000000" stroke-miterlimit="10" x1="81.761" y1="287.909" x2="91.273" y2="313.724"/>
14 <line fill="none" stroke="#000000" stroke-miterlimit="10" x1="84.814" y1="288.141" x2="93.864" y2="313.514"/>

```

Figure 3. The contents of the SVG file, showing the hierarchical graphic information designed in Figure 2.



```

file : "/Volumes/Macintosh HD/Users/z/Documents/dev-lib/CNAT-odot/src/o.svg/svg-test.svg",
/layer : {
  /type : "svg",
  /id : "Layer_1",
  /version : "1.1",
  /x : 0,
  /y : 0,
  /width : 792,
  /height : 612,
  /viewBox : 0,
  /enable-background : "new 0 0 792 612",
  /space : "preserve",
  /page/1 : {
    /type : "g",
    /id : "page-1",
    /stave/3 : {
      /type : "g",
      /id : "stave-3",
      /staff_2 : {
        /type : "rect",
        /id : "staff_2",
        /x : 38.395,
        /y : 249.535,
        /fill : "none",
        /stroke : "#000000",
        /stroke-miterlimit : 10,
        /width : 715.966,
        /height : 111.765
      },
      /events : {
        /type : "g",
        /id : "events_1",
        /circle/1 : {
          /type : "circle",
          /id : "circle_1",
          /fill : "#FFFFFF",
          /stroke : "#000000",
          /stroke-miterlimit : 10,
          /cx : 104.415,
          /cy : 287.212,
          /r : 3.76
        },

```

Figure 4. The contents of the SVG file designed in Figure 2, transcoded to Odot in the MaxMSP programming environment.

hierarchical data structures within data-flow programming environments like Max, Pure Data (Pd) ²¹, and NodeRed ²². Through odot's expression language, the OSC bundle becomes a local scope for time-synchronized variables in which functions may be applied referencing the named contents of the bundle [16]. Thus, by transcoding the SVG file contents into an OSC bundle, it is possible to process the data in Max/Pd, interpreting the values intended for graphic rendering as control parameters for synthesis, spatialization, and any other parameter controllable with the computer.

The graphic content and grouping relations within an SVG file are described by the organization of XML elements, and graphic primitives as specified by the SVG Standard. For example, a group of SVG elements might look like this:

```

<g id="note-duration-event">
  <circle id="notehead" cx="100" cy="300" r="3.76"/>
  <line id="duration" fill="none" stroke="#000000" stroke-width=
    "3" stroke-miterlimit="10" x1="100" y1="300" x2="200" y2=
    "300"/>
</g>

```

Each SVG element follows a similar structure, the element tag name is followed by a list of attributes to the

element. The `<g>` tag indicates a group which is closed by the `</g>` tag, and has an `id` attribute with the value "note-duration-event". The `o.svg` object creates an OSC bundle, mirroring the structure of the SVG file, and creating OSC addresses for each attribute name of a given SVG element, using the `id` attribute as the element name for example:

```

/note_duration_event/notehead/type : "circle",
/note_duration_event/notehead/cx : 100,
/note_duration_event/notehead/cy : 300,
/note_duration_event/notehead/r : 3.76,
...

```

After transcoding the SVG file into OSC, the SVG data may be interpreted, and performed in Max through the odot library, allowing us to sort and iterate over the items, and to apply interpretive functions (figure 5).

4.2 Grouping strategies

With the transcoding from SVG to OSC in place it becomes possible to begin composing within a graphic design program in a way that facilitates the interpretation and performance downstream in OSC. Using the `id` attribute to identify groups and graphic objects, it is then possible to use the SVG tree structure as a framework for developing grammars which can be used later to interpret the graphic information for the generation of control parameters.

Taking traditional musical notation as a starting point, a logical structural design to facilitate rendering might be something like the one illustrated in figure 6. With the root `<svg>` tag understood as the global container for a full score, the next largest container would be the *page*, followed by a *system* which might contain many instrument *staves* (or other output types), each with their own *staff* and *clef*. Within the individual staff group, there might be graphic information providing the *bounds* of the staff (e.g. lines marking different qualities within the vertical range of the staff as described by the clef; where the X dimension usually (but not necessarily) representing time). Within this grammar structure, the bounds of the staff provide a context for interpreting *event objects* contained within the staff group. Further, each event object grouping may contain any number of graphic objects. For example a *note-duration-event* might contain a shape identified as a *note-head*, with other graphic objects representing the event's

²¹ <http://puredata.info>

²² <http://nodered.org>

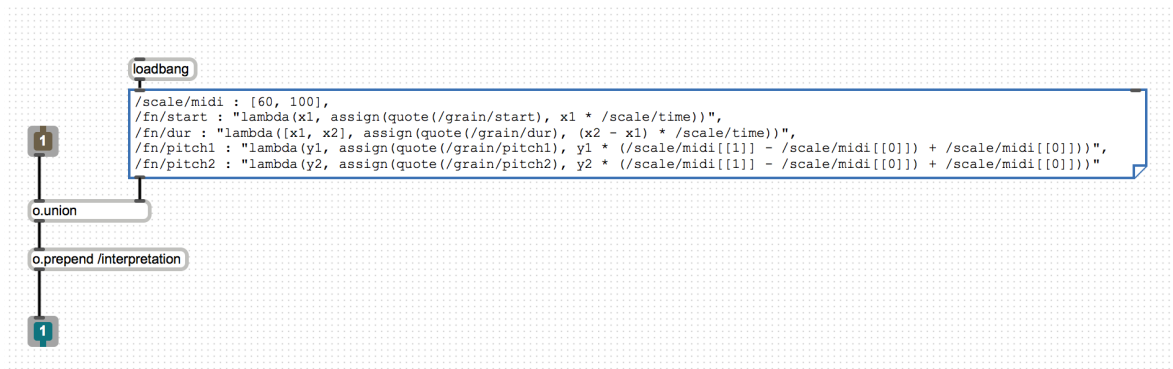


Figure 5. Example of storing interpretations of graphic information contained in the SVG file in Odot lambda functions. Here, the function describes a process of interpretation where the $x1$ value indicates the start time, scaled to a given time constant, and the duration is the horizontal span of the object.

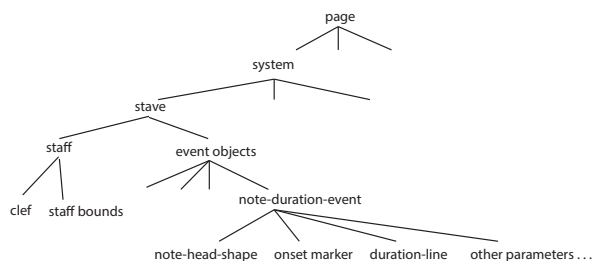


Figure 6. Example namespace hierarchy for identifying score elements. Event object could be any user defined graphic symbol (e.g. a circle, line, paths, gradients, etc.).

onset, *duration*, and any other parameters.

Figure 7 shows a potential expansion of the note-duration-event object to include with a second *frame-staff* placed above the *pitch-staff* used to notate the spatial trajectory of the sound source in a 2D frame. In this example, a dotted vertical line identified as a *stem* is used to coordinate the beginning of the trajectory with the beginning of a *col legno battuto jeté glissando*, with a duration indicated by the length of the beam line identified as *duration*. This type of trajectory is the simplest type of spatial processing, in cases with more complex treatments, such as spherical harmonic manipulation, other types of notation would be more appropriate.

5. CRITIQUE AND FUTURE WORK

The initial results of the work are encouraging, however, there are many areas that could be developed further. The study shows that it is possible to increase the rendering context flexibility by separating the score editing environment from pre-conceived ideas about how the score might be interpreted or rendered. The flip-side to the current implementation which uses only the odot/OSC expression language for parsing the assigned meaning of the notation,

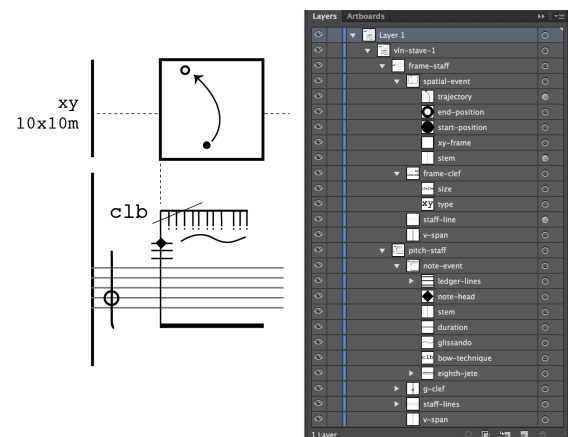


Figure 7. A more developed example showing a 2D trajectory in frame notation in Adobe Illustrator. The hierarchy structure is shown on the right side.

is that while the possibilities of the system are extremely large, this flexibility comes at the price of the parsing programming that must be written to interpret and perform the score. For example, if a user wants to also use traditional musical notation formatting rules, the entire mechanism for traditional score interpretation must be built using the odot expression language. As daunting as this might sound, the symbolic flexibility of the graphic design environment plus the many rendering media accessible through OSC may make the development of multiple rendering systems worth the effort.

Illustrator's editing environment has been very well developed for many years to become the standard for graphic design, which in addition to extensive support for print output, provides a large number graphic functions that can be leveraged for temporal media composition. However, since Illustrator is designed for graphic art not "renderable scores," there are noticeable limitations on the amount of data that can be contained in a graphics file before it begins to effect the application's responsiveness. This eventually points to the fact that a *specialized* tool for notation might

indeed be useful, providing a database and Model View Controller (MVC) architecture for interacting with score data.

Adobe's recently announced support for Node.js²³ opens up several new options for working within the Illustrator application. For example, with odot's SWIG²⁴ based JS bindings, a Node plug-in could be created to stream OSC score data directly from Illustrator without the need to save the file to disk and reload it with the `o.svg` object. With the addition of Node as a plug-in backend this means that Paper.js²⁵ could be used to create custom interactive GUIs for handling data. Paper.js is developed by the same team who wrote Scriptographer²⁶ which was a powerful JS based drawing tool building suite for Illustrator, and was one of the initial tools for an earlier version of our study. Unfortunately, Adobe drastically changed their plugin design in Illustrator CS6, which broke Scriptographer. This is an important point to be considered for any future work in the present study, and is one indicator that possibly an independent design environment might be a more reliable long-term solution. A future iteration of the study might be in the form of a Node and Paper.js based editor with a stripped-down toolkit for symbolic graphic notation. The Paper.js front-end would allow users to easily create their own interactive tools, and either export a rendering of the score to SVG for printing with a program like Illustrator, or the score could be streamed via odot/SWIG. There is some possibility that INScore's V8²⁷ integration might provide a suitable platform for these developments, this would also allow the editor to take advantage of INScore's MVC design, and traditional notation tools.

Other improvements might include a more intuitive system for defining meaning for symbols. In the process of sketching and developing a notation, it was time consuming to constantly keep objects nicely grouped and labeled using Illustrator's layer and grouping tools. This issue can be mitigated through the use of search algorithms to auto-detect symbol patterns (i.e. containing similar types of graphic objects, gestures, etc.) which would allow the artist to later apply semantic structuring rule to different members of these symbolic groupings.

6. CONCLUSION

The authoring of data in computer music systems is predominately done through graphical representations of univariate functions, whereas symbolic notation systems like

music notation are aggregate and contextual. A symbol in a notation system is given meaning through the interpretation of a human or computerized intelligence based on contextual understanding, for example the nature of a staccato string articulation is different for different dynamic ranges. Complex rendering systems incorporating digital signal processing and/or other electronic media often have a large number of parameters that artists wish to control expressively. Due to the affordances of the programming environments in which these pieces are created, there is typically a focus on control of many *single* parameters. However, the symbolic representation of information such as spatial location becomes fragmented in these systems, forcing a point in space to be represented with three separate coordinates, which in many ways obscures its perceptual simplicity.

The SVG format provides a useful method for defining meanings of symbols leveraging Illustrator's grouping and layering tools, while the graphic editing environment provided by graphic design programs like Illustrator provide a flexible vector graphic drafting environment for symbolic experimentation. Since Illustrator was designed without musical applications in mind, there are no pre-conceived playback limitations based on the application developer's idea of what "music" is, or how graphic symbols on a page should be organized. This lack of meaning leaves room for the user to sketch and experiment, as well as requiring extra effort to create meaning through an interpretive algorithm if the score is meant to be performed by the computer. Transcoding SVG format into OSC facilitates the interpretation of notation through the use of the odot expression language in the Max media programming environment, providing digital artists a mechanism to perform graphic symbolic notation with any electronic media accessible with Max, Pd, or any other application that can interpret OSC.

Preliminary work on developing an interpretation and performance system for notation stored in SVG format has proven feasible, however there is still significant work needed to bring the system to a point where it would be competitive with existing rendering systems that are specifically designed for a given medium. On the other hand the openness of the SVG format, combined with its compatibility with OSC points towards a myriad of new ways to expressively controlling new media formats with symbolic notation. Looking towards the future, the above plans for a new symbolic graphic notation editor discussed in section 5 seem to be a promising direction for the creation of notation software that is capable of being used to render new media forms that have proven difficult to notate (such as spatial audio), as well as those that have yet to be thought of.

²³ <http://www.adobe.com/devnet/cs-extension-builder/articles/extend-adobe-cc-2014-apps.html>

²⁴ <http://www.swig.org>

²⁵ <http://paperjs.org>

²⁶ <http://scriptographer.org>

²⁷ <https://code.google.com/p/v8>

Acknowledgments

I would like to thank John MacCallum and Adrian Freed for their valuable feedback in developing this study, and in developing the odot system without which this work would not exist. I would also like to thank Olivier Warusfel, Markus Noisternig, and Thibaut Carpentier for their mentorship during my spatial composition residency at Ircam where many of these ideas were developed, and to Jean Bresson for his continued feedback and great work in the field of musical representation.

7. REFERENCES

- [1] K. Stone, "Problems and methods of notation," *Perspectives of New Music*, vol. 1, no. 2, pp. 9–31, 1963.
- [2] P. Manning, "The oramics machine: From vision to reality," *Organised Sound*, vol. 17, no. 2, pp. 137–147, August 2012.
- [3] P. L. Smirnov A., "1917-1939. son z / sound in z." *PALAIS / Palais de Tokyo Magazine, Paris*, no. 7, pp. pp. 66–77, 2008.
- [4] B. Farnell, "Movement notation systems," in *The world's writing systems*, O. U. Press, Ed. New York, NY: Weidenfeld and Nicholson, 1996, pp. 855–879.
- [5] M. Battier, "Describe, transcribe, notate: Prospects and problems facing electroacoustic music," *Organised Sound*, vol. 20, no. Special Issue 01, pp. 60–67, April 2015.
- [6] J. Greeno, "Gibon's affordances," *Psychological Review*, vol. 101, no. 2, pp. 336–342, 1994.
- [7] Y. Geslin and A. Lefevre, "Sound and musical representation: the acousmographe software," in *ICMC*. Miami, USA: International Computer Music Conference, 2004.
- [8] F. Barknecht, "128 is not enough - data structures in pure data," in *Linux Audio Conference*, Karlsruhe, Germany, 2006.
- [9] H. Lohner, "The upic system: A user's report," *Computer Music Journal*, vol. 10, no. 4, pp. 42–49, 1986.
- [10] T. Tsandilas, C. Letondal, and W. E. Mackay, "Musink: Composing music through augmented drawing," in *CHI*. Boston, Massachusetts, USA: Conference on Human Factors in Computing Systems, 2009.
- [11] J. Garcia, T. Tsandilas, C. Agon, and W. Mackay, "Inksplorer: Exploring musical ideas on paper and computer," in *NIME*. Oslo, Norway: New Interfaces for Musical Expression, 2011.
- [12] M. Tohidi, W. Buxton, R. Baecker, and A. Sellen, "User sketches: A quick, inexpensive, and effective way to elicit more reflective user feedback," in *NordiCHI*. Oslo, Norway: Nordic conference on Human-computer interaction, 2006.
- [13] D. Wessel, "An enactive approach to computer music performance," in *Le Feedback dans la Creation Musical*, Y. Orlarey, Ed. Lyon, France: Studio Gramme, 2006, pp. 93–98.
- [14] A. Freed and A. Schmeder, "Features and future of open sound control version 1.1 for nime," in *NIME*. New Interfaces for Musical Expression, 2009.
- [15] A. Freed, J. MacCallum, and A. Schmeder, "Composability for musical gesture signal processing using new osc-based object and functional programming extensions to max/msp," in *NIME*. Oslo, Norway: New Interfaces for Musical Expression, 2011.
- [16] J. MacCallum, A. Freed, and D. Wessel, "Agile interface development using osc expressions and process migration," in *NIME*, Daejeon Korea, 2013.