[NOTES]: INTERACTIVE COMPUTER ASSISTED NOTATION WITH PURE DATA AND LILYPOND

Jaime E. Oliver La Rosa

New York University
Music Department, New York City, USA
Waverly Labs for Music and Computing
la.rosa@nyu.edu

ABSTRACT

The [notes] external library for Pure Data and LilyPond is presented. Section 1 provides a brief overview of music notation in computer applications. Software for music engraving, computer assisted composition, as well as Max/MSP and Pd libraries for notation visualization are introduced and some of their features and limitations are explored. Section 2 introduces the [notes] object as a new alternative suited specifically for algorithmic composition in Pd and music engraving in LilyPond. Section 3 explores the choice of software packages as well as aesthetic considerations on the process of composing music interactively and algorithmically. Section 4 shows simple example patches and output scores, and outlines some of the features and processes involved. Section 5 presents the challenges of engraving input that may not necessarily be aware of bars, beats, or other subdivisions and subdivision dependent expressions such as tuplets and proposes a feedback feature that can be used in compositional processes. Finally, potential future work and a score example are presented, followed by conclusions.

1. INTRODUCTION

Computer assisted musical notation has become an important tool in the work of musicians in a large variety of practices such as musical composition, documentation, performance, visualization, dictation, and education. Some computer programs specialize in music notation and the objective of its users is to produce a music score generally intended for human performance - we'll refer to these packages as *music engraving* software. Other computer programs are intended primarily for composition and use music notation as a temporary display to see the result of a compositional process.

Software packages for music engraving can be divided into those that allow for graphical manipulation of music symbols and those that compile text files using a specialized syntax. Two of the most popular graphical editing packages include the commercial products *Finale* and *Sibelius*, but there is also a robust open source alternative called *Musescore*. Open source packages *LilyPond* [10] and *Guido* [7], represent music in a specialized syntax in text files which are then compiled into musical scores. Text files such as those used in LilyPond not only control the placement of musical symbols in a default score layout, but allow the user to manipulate the layout itself resulting in larger control over the resulting score.

Computer Assisted Composition (CAC) packages such as *Open Music* [2] and *PWGL* [9] offer musical notation as a central feature of their software. The aim of these packages is to provide composers with a framework to design compositional algorithms. Using the Lisp programming language, these graphical environments allow the user to generate expressions that are evaluated and optionally visualized in musical notation. However, such notation is not intended for musical engraving, but as a temporary visualization tool. Therefore, if a final score is to be produced with material generated in these environments, this material must then be exported to one of the engraving programs mentioned above for editing. However, these two packages are not the only CAC alternatives. The *Abjad* [3] library extends the Python programming language and creates LilyPond scores. With *Abjad*, the user can create compositional algorithms in Python and produce notation in LilyPond syntax which can then be edited directly to produce a final score.

Real Time Interactive environments such as *Pure Data* (Pd) [12], *Max / MSP* or *Supercollider* allow the user to graphically edit and connect boxes or objects to design real time processes for sound generation and manipulation as well as integration with video, graphics, sensors, motors, other software, etc. These software package also allow for CAC, however, they don't support musical notation natively and it has been the responsibility of users to generate their own libraries for this purpose.

At least four packages allow for visualizing musical notation in *Max / MSP*: *Maxscore* [4], *inscore* [5], *LOLC* [6], and the *bach* library [1]; and at least one allows musical notation in *Pd*: *gemnotes* [8]. All of these packages allow for visualizing musical notation inside their programming envi-

ronment as temporary displays for CAC or as a display for a live performer to read from a screen. However, if a final score is to be produced with material generated in these environments, this material must then be exported to an engraving program of choice[1].

There are several advantages and flexibilities to several of these languages that [notes] does not intend to provide such as a library of objects for CAC as in [1], a graphic display native to GEM as in [8], interactive displays for live performers as in [6] or [4], or graphic notation resources, live interactive score displays, and OSC messaging as in [5].

Fomus [11] deserves a specific mention as it provides the ability to work with CM/Grace, Pure Data and Lisp and produce output of various kinds including midi and Lilypond files. It can be loaded as an external object for Pure Data though there are several differences with [notes]. The package is still in alpha version and its last release was in 2011.

2. MAIN FUNCTIONALITIES OF [NOTES]

The *[notes]* library of external objects for *Pd* is written in the C programming language and allows the user to bridge a gap found in some of the notation packages mentioned above and generate musical notation directly in the engraving program *LilyPond*. LilyPond is therefore not only the software where the score will eventually be edited, but also a temporary display. In this sense, it shares this feature with *Abjad* and *Fomus*.

The main objective of this library is to produce music notation in LilyPond syntax - that can be further edited - to produce a final score. It is therefore the job of the user to develop or adapt algorithms in Pd that produce input for the object to display it in music notation, or to create live interactive musical score systems, or design any other applications that need to display data in musical notation. For this reason, I am using the term *Computer Assisted Notation* or CAN to describe the process of converting input data to musical notation.

The way in which [notes] "assists" you in converting data into music notation is in parsing a sequence of durations into beats and bars according to meter data and then encode this parsed data into a syntax that LilyPond can then compile into a score. [notes] gives the user the option of choosing a few basic layout options such as paper size, vertical and landscape orientation, and then it uses LilyPond's powerful default layout settings.

Some of [notes]' most important features include the ability to specify articulations, dynamics, clefs, tuplets, meter, tempo, tremolo, note heads, grace notes, clusters, arpeggio, as well as expressions that span multiple notes such as crescendos, several kinds of texts optionally connected by lines and arrows, glissandi, pedal indications, and several others. Finally, [notes] is able to have multiple voices per stave and several staves per score by connecting several [notes] objects together with other objects in the library.

The software is still in its design stage, although the alpha version has already been downloaded multiple times and several users have reported using it successfully. There are numerous features that will be added over time. Currently however, some pieces of music can be automatically rendered each time, producing different versions of a work ready to be performed. An excerpt form the score for the piece *flexura* for cello and MANO controller, written by the author in 2015 has been included in Figure 1 as an example of a score generated with the notes external. The patch used for this section contains small generative cells that are governed by a larger probabilistic system. These cells contain not only pitch and duration information, but also articulation and spans, leaving a score that can be further edited. As can be seen, scores of significant complexity, in notational terms, can be generated automatically.

In what follows, I'll explain the rationale for the design choices made in developing this package, followed by some of the challenges presented by its design and the solutions that were adopted as well as the next steps in its development.

3. DESIGN CHOICES

3.1. Choice of Software Packages

The [notes] external runs in the *Pd* graphical programming environment and produces scores in *LilyPond* syntax, which can be compiled into .pdf, .ps, or .png formats. If LilyPond is installed properly, [notes] will load a .pdf version of the score automatically. *Pd* was chosen for its open source license, its ability to produce live interactive software, and for its support of multiple media. *LilyPond* was also chosen for its open source license as well as for its specialization in music engraving. This specialization allows for temporary displays during the design of the program, but it also allows for the further editing of the file to produce a final score in any of the aforementioned formats without having to export musical material into a separate software package.

Although Pd might be seen as a limited or inadequate tool for CAC compared to text based approaches [3], this is largely dependent on style and choice of techniques. However, Pd can allow the user to hear interactive processes live before notation and coordinate an electronic or multimedia part with the score in ways that would be harder in other platforms.

3.2. Live Interaction and the Score as a Document

[notes] is designed to receive input that is generated live in real time. While this design choice presents multiple chal-

[1] For a detailed review of musical notation in non real time and real time platforms and their design philosophies consult [13], [3], and [1]

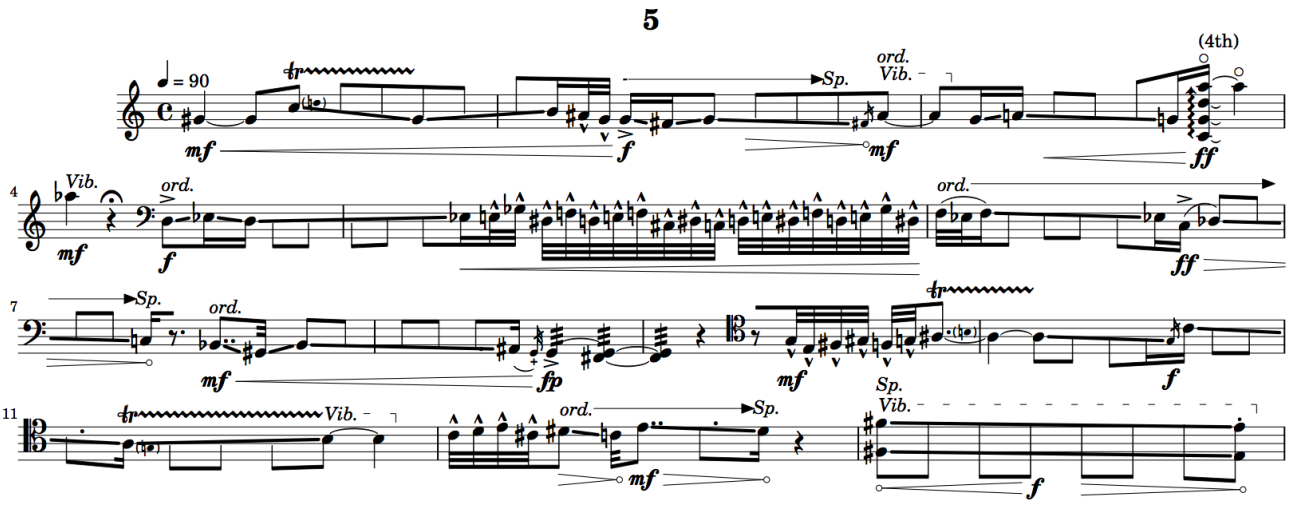

Figure 1. An excerpt of the cello part for the piece *flexura* showing the kind of output generated by the [notes] library.

lenges[2] associated with the contingent nature of this approach, it also provides the user with the choice of implementing algorithms that may or may not be aware of meter, bars, or other basic assumptions of traditional music notation. More importantly, working in an interactive manner allows a composer to sculpt an algorithm through her experience of its outcome over time, editing and tweaking it to achieve a desired musical result. In this sense, the final notation becomes a document of the work - rather than the work itself - and the work can potentially render different results if executed at a different time or in different circumstances.

3.3. Software Barriers and The Point of No Return

When composing using algorithms, there are often several layers of work that begin with the design of the algorithm and end with the final editing of the score, a step which is often done manually. Particularly in the initial phases of the compositional process it is common to modify the algorithm based on evaluation of its output and then generate new output. Another strategy is to build "scores in an iterative and incremental way" [3] as proposed by *abjad*, which is also possible in [notes] by gradually appending features to a sequence of input messages. This gradual sculpting process can continue until the beginning of manual work or until it is not longer possible to operate on previous output, which is what I call "the point of no return".

For example, if one choses (or is forced by limitations in the software) to add articulations manually to the output of a generative process, then once these articulations are added, the algorithm cannot be used again to generate output unless one is willing to discard the already invested manual labor. As an example, in the score shown in Figure 1, most if not all articulations were generated by the algorithm, but dynamics were added manually in the final editing of the score. A "point of no return" in this case is also the point where composition as a formalized process ends and arbitrary creative decisions begin.

"Points of no return" refer not only to the beginning of manual labor, but also sometimes to the switching of software packages such as moving form a Pd patch to editing LilyPond Scores. Once a LilyPond score is being edited it cannot be read or transformed by Pd anymore, so a point of no return is created if it is not possible to formalize a method to achieve these manual edits or if it isn't possible to algorithmically operate on that score anymore or if information is lost in the translation process. Furthermore, a composer might use several software packages to generate and transform musical material potentially creating multiple points of no return. [notes] attempts to postpone and reduce the point(s) of no return, by increasing the specificity and complexity of available musical expressions before changing software packages or initiating manual labor, the composer thus retains the ability to further sculpt the algorithm or operate on its output for a longer period of time, but is also responsible for the complexity and specificity of the algorithm that generates her material.

4. OPERATION

[notes] receives input in the form of messages that contain note expressions with at least a duration, and a pitch or silence. *input* messages have the format: "input -pit 60 -dur 16". In this case, 60 refers to middle c in a "midi" scale and 16 refers to a global parameter called "minimum reference duration". If "minimum reference duration" is equal to a 32nd note (which is the default), then the duration entered

[2] Some of which are outlined in Section 5.

in the input message is 16 32nd notes, or a half note. [notes] does not do any quantization for pitch or rhythm. All pitches must be specified as integers for half tones and with .5 for quarter tones. The user can specify whether to use sharp or flat symbols. All durations must be integers.

Once all input messages have been entered, the message "write filename" will fit the sequence of input note expressions into beat and bar subdivisions whenever necessary, encode them in *LilyPond* readable syntax in a file called *filename.ly*, then call *LilyPond* to compile *filename.ly* into .pdf format, and finally automatically open the .pdf file in a pdf viewer native to the operating system used[3].

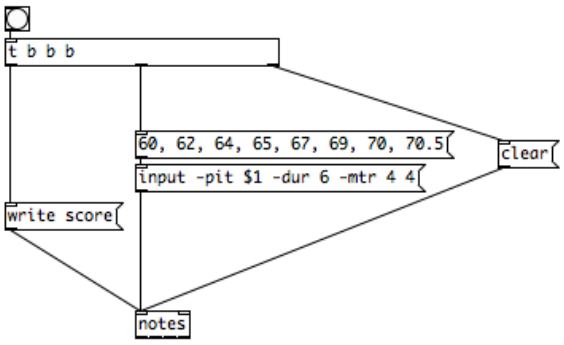

Figure 2. A simple patch using the [notes] external object.

Figure 2 shows a simple patch using the [notes] external object. The first message sent to the object is "clear" which erases any input it may have stored in memory similar to the way the *textfile* object works in *Pd*. Following the "clear" message, the first six pitches of the major scale starting in middle c and two more pitches are entered with a duration of 6 each. As mentioned earlier the default reference duration is a 32nd note, and therefore all of these notes will have a duration of 6 32nd notes or its equivalent, a dotted 8th note. Finally, the "write" message, followed by a filename, produces the output score shown in Figure 3.

As can be seen in Figure 3 durations are automatically split and tied to fit beat and bar divisions. This is done to conform with the notion of bar needed in western musical notation, but also to conform with conventions on beat divisions and subdivisions that allow human performers to read the score better. Furthermore, if a musical expression such as a glissando, spans multiple bars, it will also be adjusted.

Several parameters such as meter or clef have default values, however, they can all be changed at any time in the sequence and are always attached to a note expression. All note expressions can be described further by adding ties, dynamics, articulation, text spans, slurs, pedal or tremolo markings, alternate note heads, or specifying rests, grace

[3]Only OS X and Linux are currently supported, although it should be trivial for an experienced windows programmer to contribute and/or compile a windows version.

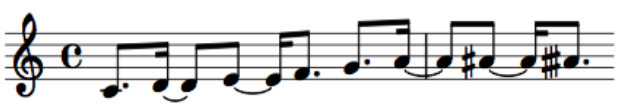

Figure 3. Score produced with the patch showed in Figure 2

notes, indicating tuplets, chords, fingering, arpeggios, clusters, and a growing number of features described in the help file.

Given the orientation towards music engraving and to allow for easier viewing and documentation, the external allows the user to specify other global settings such as page size and orientation; title, subtitle and author; and construct patches that can contain multiple staves and multiple voices per stave using information produced in the outlets of the [notes] object and combining it with the [mainscore] object provided with the library as exemplified in the help file and demonstrated below.

5. DESIGN CHALLENGES

As briefly mentioned earlier, [notes] receives contingent input, that is, input which is unpredictable. Because of this, several problems may arise for which [notes] needs strategies. Some of these issues are outlined below.

5.1. Meter Changes

A simple yet not trivial example is a change in meter. Meter determines the length of a bar and its internal subdivision so that, as expected, in 4/4 meter there are four quarter note beats in each bar. Consider the patch shown in Figure 4. In this patch, the default pitch is middle c and the default meter is 4/4, and a series of varying durations succeed each other with the request of a meter change at a point in the sequence. Since the program or the user generating the input is not necessarily aware of where the bar subdivisions lie in the score, a request for a meter change, as in this case, may not fall at the beginning of a new bar, but during an incomplete bar. Due to this, [notes] will automatically calculate the meter of the truncated bar - in this case is 7/8 in bar two as shown in Figure 5 - and then display the following note expressions with the requested 3/4 meter.

5.2. Multiple Voices and Staves

When two or more sequences of notes happen in parallel one might want to input them as separate voices in the same staff or as multiple parts in different staves. If the composer wants a score with parts that have independent meter and bar structures it is possible to combine each of these individual parts into a global score. However, if one wants these parts

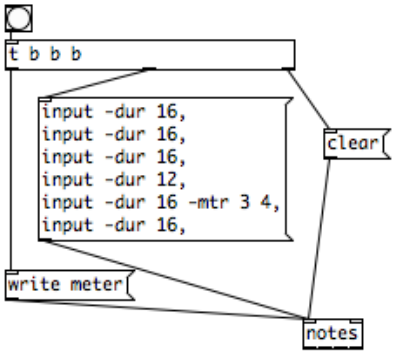

Figure 4. A patch showing a meter change.

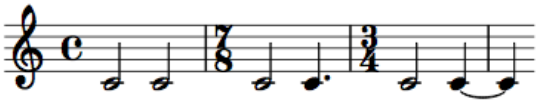

Figure 5. Score produced with the patch showed in Figure 4 showing how the request for a 3/4 meter automatically generates a 7/8 meter.

to share meter and bar structure, of one of the voices must determine the bar structure of the other voices. In this case, multiple [notes] objects can be interconnected and fed into a [mainscore] object to produce a global score, as shown in Figures 6 and 7. Finally, all parts are created on individual files and can be opened manually by the user.

5.3. Tuplets

Another challenge is the specification of tuplets. A tuplet expression should ideally not cross bar lines and beat divisions as it makes it harder for performers to read. LilyPond will do its best to compile such expressions in its default settings often leading to incorrect musical expressions, but [notes] will consider these expressions invalid. If the user or algorithm is not aware of its position with respect to bars, beats and sub-beats, there is a significant risk of specifying tuplets in the invalid positions mentioned above. In these cases, [notes] will report warning and error messages in the Pd console and often LilyPond will fail to compile. In contrast to the meter problem described in the previous section, I've made a design choice to not re-bar and re-meter in order to make a valid tuplet expression as this could be done directly with a meter change. Instead, I have opted to leave the responsibility of being aware of tuplet positioning to the user and the job of compiling (or not) to [notes] and LilyPond until a good solution is found.

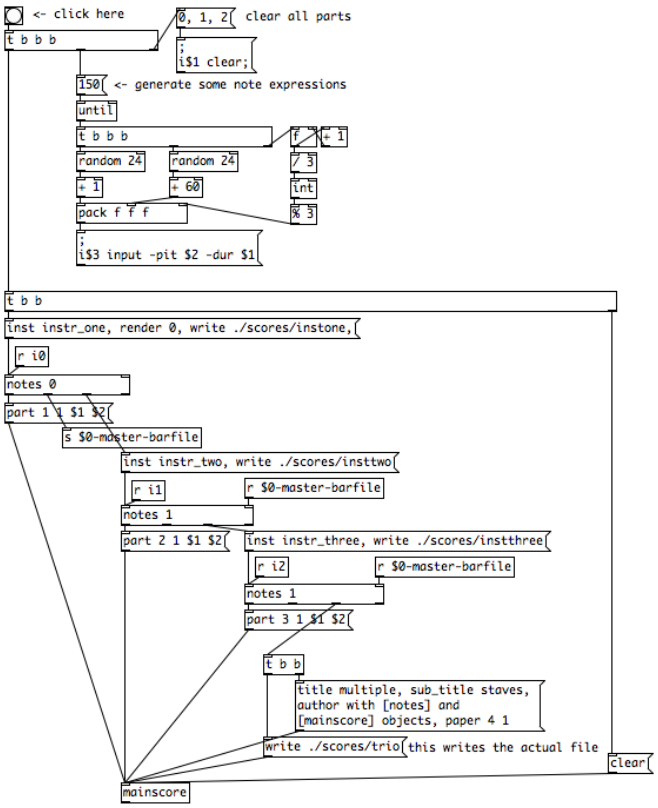

Figure 6. A patch showing how to use multiple notes objects and a [mainscore] object to produce a score with multiple staves. This particular patch generates random pitches and durations for each voice and uses the bar structure of the first staff to subdivide all other staves.

5.4. Reporting Position in Bar Live

While it is ideal that when requiring bar or beat dependent notations such as tuplets, the program or user generating the input is aware of its position in the bar, this is not always the case and an alternative strategy must be developed. In order to allow an algorithm to determine whether a tuplet is a valid expression at any moment in the process of generating input, the fourth and last outlet of [notes] provides the number of reference durations already inputted to the object.

For example, if the object reports a 0 in its fourth outlet, this means that the current position is at the beginning of a bar and several tuplet expressions would be valid. Similarly, if the current meter is 3/4 and the minimum reference duration is 32, then we'll know that a bar has a length of 24 and that a beat has a length of 8, so if the same outlet is now reporting 23 there are no valid tuplet expressions available as only one 32nd note is left to the bar's end.

This strategy is not a compensation for bad design. Rather than a fix, it should be seen as a feature in the sense proposed in section 3.2, namely, that it allows for the design of compositional algorithms that can become aware of their posi-

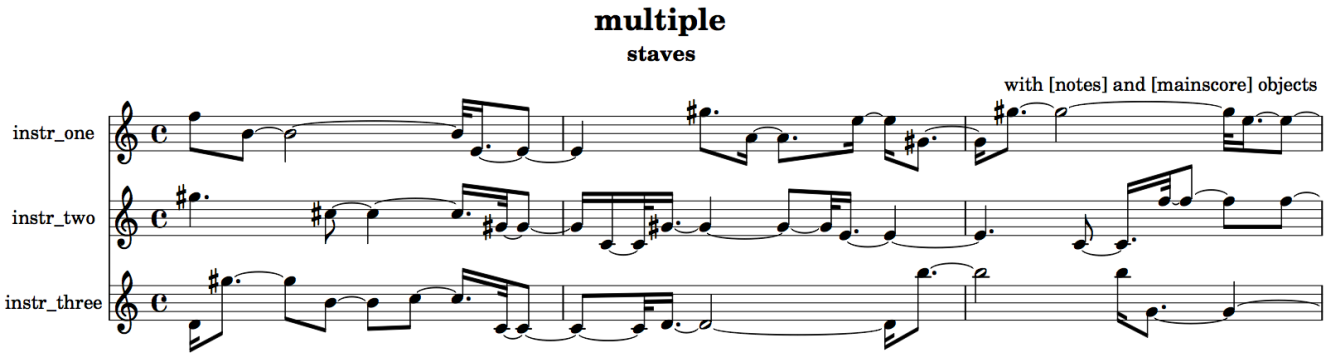

Figure 7. First bars of the score produced with the patch showed in Figure 6 demonstrating title, subtitle, author, and instrument name features. All parts share the same meter and bar structure.

tion, structure, and past input. This feature contributes to the interactive properties of a live generative system which can be designed to take feedback from the Computer Assisted Notation process to determine its next output.

6. FUTURE WORK AND CHALLENGES

While Pd has remained significantly committed to backward compatibility, a significant challenge lies in the updates provided to LilyPond on an ongoing basis. LilyPond has a version declaration in its score to control changes that occur from version to version, however, it is necessary to remain up to date with changes for the external to work properly.

While LilyPond has been chosen as the notation program for its orientation towards music engraving therefore reducing the need to export work into another editing software, the same output provided by [notes] could easily be encoded in other syntaxes. The GUIDO [7] notation format could be an alternative notation package. Some limitations to using LilyPond and GUIDO include their limitations in generating some of the graphic symbols used in contemporary music and of graphic notation in general, but no notation package provides a complete solution to this problem and it is not the intention of [notes] to do so. Lilypond is certainly moving in the correct direction and I expect it to continue improving the availability of contemporary music symbols.

Finally, [notes] is a personal project and most notation features added so far correspond to my own compositional needs. The code for [notes] is a little over 3000 lines long and thus far, development, maintenance, and testing relies exclusively on the author. As the [notes] external becomes robust enough for a full release it will be hosted in github or other collaborative code management packages so other developers may contribute to it.

The code for the [notes] library of externals is available under a GNU Public license and therefore it can be modified by anyone. Contributions and bug reports and fixes by other users can be made in order to increase the capabilities of the library. The implementation of GUIDO syntax or windows and Max/ MSP versions are interesting possibilities for other people to contribute to the project.

7. CONCLUSIONS

As presented in this paper, the [notes] external is an addition to the various options already available to provide musical notation in Pure Data that offers the ability to generate scores of considerable complexity directly in an engraving program such as LilyPond. The way in which [notes] "assists" you in converting data into music notation is in parsing a sequence of durations into beats and bars according to meter data and then encode this parsed data into a syntax that LilyPond can then compile into a score. Notes is restricted to *computer assisted notation* and it is the responsibility of the user to develop any generative algorithms, quantization methods, etc. [notes] attempts to postpone and reduce the "point(s) of no return", by increasing the specificity and complexity of available musical expressions - and of the algorithm that generates them before changing software packages or initiating manual labor, the composer thus retains the ability to further sculpt the algorithm or operate on its output for a longer period of time, but is also responsible for the complexity and specificity of the algorithm that generates her material. [notes] receives contingent, unpredictable input, and provides a feedback mechanism that allows algorithms to be aware of the current position with respect to bars and beats. Finally, significant challenges remain in the development, maintenance, and testing of [notes], however, hosting the code in a collaborative code management package should allow other developers to contribute.

8. REFERENCES

[1] A. Agostini and D. Ghisi, "Real-time computer-aided composition with bach," *Contemporary Music Review*,

vol. 32, no. 1, pp. 41–48, 2013.

[2] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue, "Computer-assisted composition at ircam: from patchwork to openmusic," *Computer Music Journal*, vol. 23, no. 3, pp. 59–72, 1999.

[3] T. Bača, J. W. Oberholtzer, and V. Adán, "Abjad: An open-source software system for formalized score control," in *Proceedings of the International Conference on Technologies for Music Notation and Representation*, 2015.

[4] N. Didkovsky and G. Hajdu, "Maxscore: Music notation in max/msp," in *Proceedings of the International Computer Music Conference*, 2008, pp. 483–486.

[5] D. Fober, S. Letz, Y. Orlarey, and F. Bevilacqua, "Programming interactive music scores with inscore," in *Sound and Music Computing*, 2013, pp. 185–190.

[6] J. Freeman, "Bringing instrumental musicians into interactive music systems through notation," *Leonardo Music Journal*, vol. 21, pp. 15–16, 2011.

[7] H. H. Hoos, K. A. Hamel, K. Renz, and J. Kilian, "The guido notation format – a novel approach for adequately representing score-level music," 1998.

[8] E. Kelly, "Gemnotes: a realtime music notation system for pure data," in *Proceedings of IV International Conference of Pure data–Weimar*, 2011.

[9] M. Laurson, M. Kuuskankare, and V. Norilo, "An overview of pwgl, a visual programming environment for music," *Computer Music Journal*, vol. 33, no. 1, pp. 19–31, 2009.

[10] H.-W. Nienhuys and J. Nieuwenhuizen, "Lilypond, a system for automated music engraving," in *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, vol. 1. Citeseer, 2003.

[11] D. Psenicka, "Fomus, a music notation package for computer music composers," in *Proceedings of the International Computer Music Conference*, 2011.

[12] M. Puckette *et al.*, "Pure data: another integrated computer music environment," *Proceedings of the Second Intercollege Computer Music Concerts*, pp. 37–41, 1996.

[13] J. R. Treviño, "Compositional and analytic applications of automated music notation via object-oriented programming," *Dissertation:*, 2013.