HACKING MUSIC NOTATION WITH BACH AND CAGE

Associate Professor David Hirst
Melbourne Conservatorium of Music
University of Melbourne

ABSTRACT

The Composer's Little Helper (CLH) is a Max patcher that makes use of the "Bach" and "Cage" libraries to manipulate and mix musical notation that has been saved in a "shelf" as separate segments. CLH implements musical operations such as transposition, inversion, retrograde, plus a number of other "treatments" that use the high-level "Cage" library for real-time computer-aided composition.

1. CONTEXT

1.1. Introduction

This paper describes a computer software system, called the "Composer's Little Helper", which can manipulate musical notation that has been stored in sections. Written in the Max/MSP visual programming language, the Composer's Little Helper (CLH) makes use of the "Bach" library for musical notation and computer-aided composition by Agostini & Ghisi (2015). CLH implements musical operations such as transposition, inversion, retrograde, plus a number of other "treatments" that use the high-level "Cage" library for real-time computer-aided composition by Agostini et al (2014).

1.2. Background

The Composer's Little Helper[1] sits within the realm of "interactive composition". In their introductory article to a feature edition of the Journal of New Music Research on interactive composition, Bresson and Chadabe (2017) conclude that:

> By highlighting a number of new concepts, ideas, and challenges in working with interactivity, among them new time paradigms, sequencing, compositional utilities, graphical representation and authoring of interactive processes, we hope to encourage artistic experimentation and foster new ideas in the art of music. (Bresson and Chadabe, 2017: 2)

CLH aims to provide a practical illustration of the use of compositional utilities, graphical representations, and interactive composition, along with artistic experimentation.

CLH utilises the bach library for Max. Developed by Agostini and Ghisi (2015), bach is a Max library for musical notation and computer-aided composition. In their article on the library's development, the authors divide computer music into the two fields of digital signal processing and the treatment of symbolic musical data. They assert that while there have been widespread advances in DSP algorithms and systems, the field of treatment of symbolic musical data "seems confined to a small number of specialist software systems" (Agostini and Ghisi, 2015: 11). Amongst the examples they list are the following: OpenMusic (Agon 1998), PWGL (Laurson and Kuuskankare 2002), and Common Music (Taube 1991).

Miller Puckette has also pointed out this divide: "while we have good paradigms for describing processes … and while much work has been done on representations of musical data … we lack a fluid mechanism for the two worlds to interoperate" (Puckette, 2004: 1).

To some extent, the International Conference on Technologies for Music Notation and Representation[2] (TENOR), which has been held annually since 2015, is a step in bridging this divide. It is against this overall backdrop that the bach library for Max was developed, and the CLH is an attempt at implementing a practical system that might demonstrate one example of "bridging the divide between signal processing and musical representation processes" and implementing "interactive composition".

1.3. About Bach

The bach[3] library comprises about 110 Max externals and 120 abstractions. The main goal of the bach library is the manipulation of musical scores, represented either as common Western notation score form (bach.score) or piano-roll type form (bach.roll). Scores are rather complex data structures and quite often composition software has made use of the language Lisp to represent them. The authors of bach for Max wanted "to allow easy exchange of data with the major Lisp-based systems, such as … OpenMusic and PWGL. Hence we chose to implement a tree structure inspired by the Lisp list, called llll"

[1] CLH software is available from https://davidhirst.me/software/

[2] http://tenor-conference.org

[3] http://www.bachproject.net

(Agostini, A. and D. Ghisi, 2015: 13). This Lisp-like linked list can contain all the standard Max data types, to an arbitrary level of depth, and the detail of the relationship between bach data structures and Max data types is set out in their comprehensive CMJ article (Agostini, A. and D. Ghisi, 2015).

Here is a brief summary of the bach library families:

- Score editors and sequencers (bach.score, bach.roll)
- Other User Interface Objects for Musical Representation (bach.circle, bach.tonnetz)
- Generic Data-Processing Objects (rotation, reversal, retrieval of general information, insertion, substitution, operations on sets, iteration, collection of elements, etc)
- Specialized Data-Processing Objects, designed to operate on bach llls (bach.expr, bach.n2mc)
- Rhythmic Quantization (bach.quantize)
- Constraint Programming, or stating "musical formalization in terms of the qualities of the result" (bach.constraints)

We will see how objects from these families are used in the context of the CLH system below.

1.4. About Cage

The cage library[4] is a library of abstractions built upon the bach library. Developed with the support of the Haute École de Musique in Geneva, cage modules "in general perform higher-level tasks, with a compositional rather than strictly technical connotation" (Agostini, Daubresse, and Ghisi 2014: 308).

Cage is first and foremost a library of ready to use modules to assist with computer aided composition. Cage also has a pedagogical connotation in that it is entirely built as high-level abstractions and can therefore be viewed, analysed and modified in Max. Thirdly, cage is highly influenced by a real-time approach to computer aided composition: "creating and editing symbolic musical data is not necessarily an out-of-time activity, but it follows the temporal flow of the compositional process, and adapts to it." (Agostini, Daubresse, and Ghisi 2014: 308).

Here is a brief summary of the cage library families, with some examples of each:

- Pitch generation (cage.scale, cage.notewalk)
- Generation and treatment of melodic profiles (cage.profile.gen, cage.profile.stretch)
- Processes inspired by electro-acoustic practices (cage.pitchshift, cage.fm, cage.delay, cage.granulate)
- Harmonic and rhythmic interpolation, formalization of agogics (cage.chordinterp, cage.rhythminterp, cage.timewarp)
- Automata, L-systems, etc (cage.chain, cage.life, cage.lombricus)
- Musical set theory tools (cage.chroma2pcset, cage.chroma2centroid)
- Scores – a set of modules for "editing" or "mixing" scores (cage.rollinterp, cage.scissors, cage.glue)
- SDIF files support – modules for reading and writing SDIF files and manipulating or using the data (eg. fundamental tracking to pitch conversion)
- Audio rendering of bach scores: cage.ezaddsynth~ (a basic additive synthesis engine) and cage.ezseq~ (a basic sound file sampler)

1.5. About Hans Tutschku

In 2015, German composer Hans Tutschku (2015) produced a series of four videos that described a Max-based system he created to produce his composition *Shadow of Bells*.[5] The videos outlined his compositional method in using the bach library for Max, but without providing all the details or the Max patcher code. These videos provided the impetus to reproduce and extend this working method using higher level cage Max abstractions.

2. RESEARCH QUESTIONS

The primary research question in this present study was: Can we reproduce the system for computer assisted composition outlined by Tutschku, but with the enhancement of using the higher level abstractions provided for in the cage for Max library? The processing of music data which Tutschku described in his videos relied more on his own idiosyncratic, text-based, lisp-like commands, which he was able to type in. In our system, one aim was to replace this textual way of working with User Interface-driven methods that made best use of the cage library.

A secondary aim was to begin to investigate whether some extensions could be made to the system utilising the dada library that explores graphical user interface objects, and which are more interactive[6]. These dada objects are still in their experimental phase.

3. CLH SYSTEM DETAILS

3.1. CLH overview

The the author's system makes higher level use of the "Cage" library to manipulate common practice western

[4] http://www.bachproject.net/cage/

[5] http://www.tutschku.com/shadow-of-bells/

[6] http://www.bachproject.net/dada/

musical notation in order to create a musical work. The CLH process begins with reading a pre-existing MIDI file and creating segments. The segments are saved, and then various treatments can be performed on selected segments (transposition, retrograde, inversion, rotation, time stretching, pitch stretching, repetition, interpolation, etc).

The treated versions are also saved into the composer's "shelf". The shelf is written to disc. The next step involves mixing the desired segments with each other. The mix can be saved as a MIDI file or converted to a score and saved as a Music XML file which can be read by a music notation program to undergo further refinement and become the final music score. CLH includes an audio playback system to hear each score fragment as the composition proceeds.

Figure 1. The Composer's Little Helper (CLH) Max patcher.

Figure 1 shows the Composer's Little Helper Max patcher with the overview of its structure, on the left-hand side, consisting of the sub-patchers: **CLH-segmentation, CLH-treatment, CLH-mix,** and **CLH-roll2score.** The right-hand side shows the controls for a simple **Global Player.** The user either employs the built in Apple MIDI player or a VSTi instrument loaded as a plugin.

Each of these sub-patchers will now be discussed in detail.

3.2. CLH-segmentation

Segmentation involves reading a MIDI file of a whole piece, or even just a single motive, and segmenting it so that the individual segments can be stored in order to be manipulated in some way at a later stage.

In Figure 2, **CLH-segmentation** displays the input to the segmentation process as a bach.roll object which is a proportional notation representation of pitch, shown using music staff lines, versus time, with duration depicted as horizontal line traces. In this example, we have the first few bars of the Debussy Prélude Book 1 Number 6 *Des pas sur la neige* – purely for illustrative purposes here. When first read, this file already has a number of markers for bar lines and tempi. So any such markers must be cleared using the "clearmarkers" button so that the composer can insert her/his own. Further functionality to clean up the input includes "merge 200" (to synchronise attacks from a MIDI performance) and "legato extend" (to create a more legato performance). There is also a number field to zoom horizontally to focus on detail when adding markers for segmentation (Entered as a percentage).

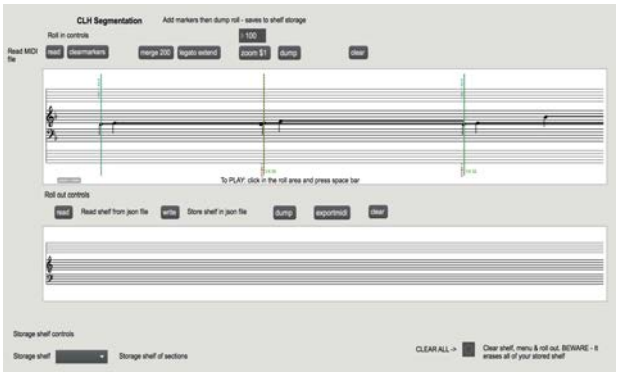

Figure 2. CLH-segmentation, showing the input music "roll" at the top – as yet un-segmented by the composer.

Once all initial markers have been cleared, the composer works methodically through the input music to add markers on the "roll" window itself (using right click) – in order to mark the desired segmentation points. The markers can be moved around, but once happy with their positions, the composer the clicks "dump" and the marked segments are dumped into a temporary storage "shelf". This populates a drop-down menu underneath the "Roll out" window. The composer can then select a section to view in the bottom bach.roll area. At this stage, the shelf of segments can be stored in a json file using the "write" button in the Roll out controls area. Figure 3 shows a section of the Debussy piece with markers 2, 3 and 4 placed on it. Below this is another bach.roll displaying "sectn5" only, which has been selected in the drop-down menu below it.

Figure 3. CLH-segmentation, showing markers placed in the top bach.roll and a single segment in the bach.roll underneath.

Sometimes, the individual note start times, or their length, may need to be edited in the upper roll in order to facilitate segmentation. For example: placing a marker in the middle of a sounding note will lead to unpredictable results. Better to shorten the note or move the marker. Moving notes around and editing them is eminently possible in a bach.roll window. Playback any bach.roll through the sound system by clicking in its window and pressing the space bar, so that you can hear your work as it progresses. Press again to stop playback. At any stage, the composer can save a segment as a separate MIDI file (exportmidi) – then import and segment that. Close CLH-segmentation when the segmentation is complete.

3.3. CLH-treatment

Treatment involves selecting a segment from the shelf menu[7], applying one or more treatments to it, giving the new section a name, storing it to the shelf, writing the augmented shelf to a json file, and repeat this way of working until you have a new repertoire of segments stored in the shelf - to be selected from and mixed at a later date.

To begin, open the CLH-treatment subpatch, clear Roll 1 and Roll 2, read the shelf from a json file, select a segment from the shelf drop-down menu at the top of the CLH Treatment window (Figure 4 shows a cleared-out treatment window before a segment has been selected from the shelf).

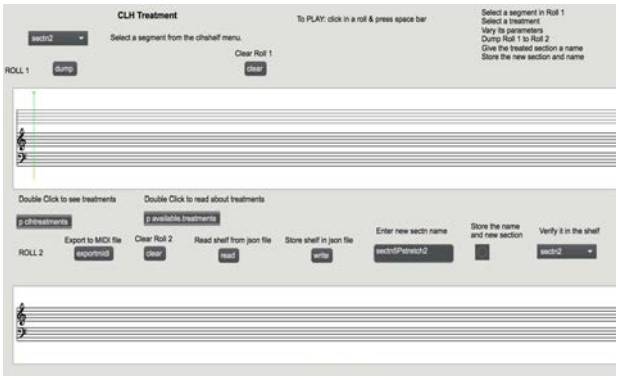

Figure 4. CLH-treatment window with Roll 1 and Roll 2 cleared ready to select a segment from the shelf and apply a treatment to it.

Once a segment has been selected and loaded into "ROLL 1", a treatment can be selected by opening the "clhtreatments" sub-patch, which lies between ROLL 1 and ROLL 2. Figure 5 depicts the available treatments such as: Inversion, Transposition, Time Warp, Pitch Stretch, Trim Silence, Stretch Time Uniformly, Time Shift, Rotate Roll, rit/accel, Repeat, Playpen, down to Retrograde. Treatments are in a sequence from top to bottom, so they can be chained together, but only in the set order. A single treatment can be enabled via a check box.

[7] If you are returning from a break and re-starting Max, you will need to read the json file back in to re-populate the shelf of segments.

Some treatments have a single parameter that can be changed next to it, for example "Transpose" has the transposition value input as a number of cents (+/- = up/down). Other treatments, such as "Inversion", require the composer to open a separate GUI window to manipulate the treatment parameters.

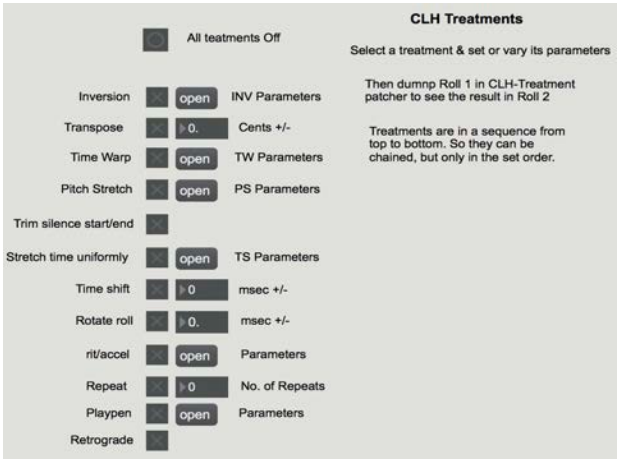

Figure 5. clhtreatments sub-patch with treatments listed from top to bottom.

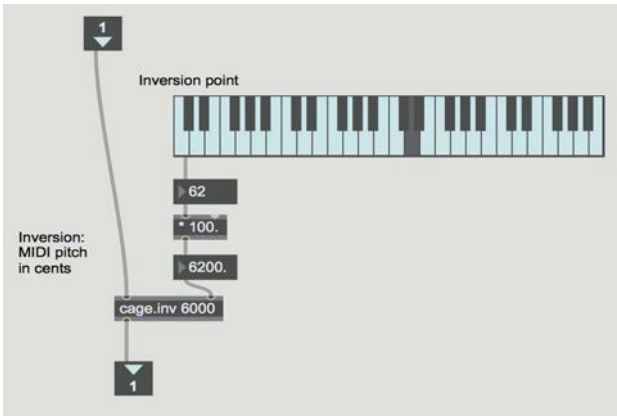

Figure 6. Inversion GUI with the inversion point specified on a MIDI keyboard graphic.

Figure 6 shows the Inversion treatment GUI window, where the inversion is performed around a pitch specified on a keyboard graphic. The resulting inversion is shown on the CLH-treatment window in Figure 7.

Some of the more complex treatments, such as Time Warp and Pitch Stretch, have correspondingly complex parameter GUIs. Pitch Stretch, which is a modified "Through the Looking Glass Bach Tutorial", requires the composer to dump Roll 1 into the Pitch Stretch GUI, vary the paraments to stretch the intervals between pitches (keeping the rhythm the same), and then dump the result out to Roll 2 in the CLH-treatments window (Figure 8).

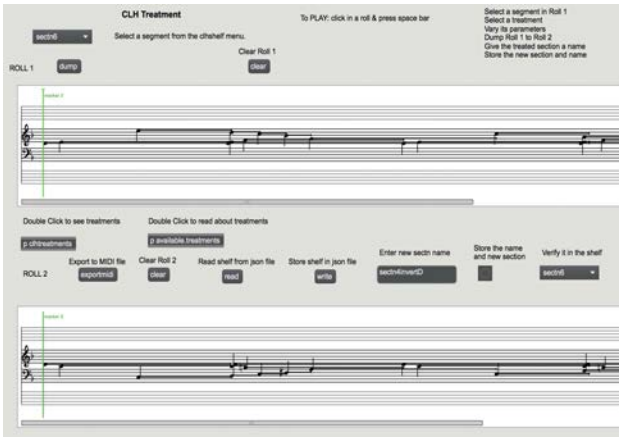

Figure 7. The inversion around D is shown in the CLH-treatment window.

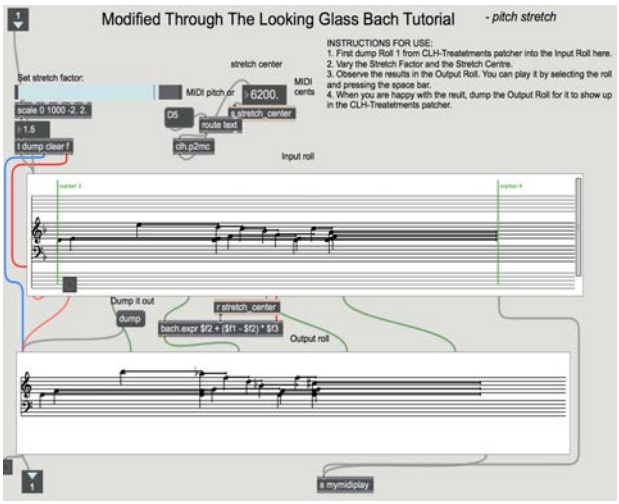

Figure 8. Pitch Stretch GUI involves dumping ROLL 1 into it, varying parameters, and dumping the result back to ROLL 2.

As the composer builds up a repertoire of treated sections in the shelf menu, treated sections can be used as the basis for further treatments. When enough treated sections have been created, the composer writes the shelf to a file using the "write" button on the CLH Treatment window and closes this window to return to the overview patcher to move on to the next phase of the process, which is mixing.

3.4. CLH-mix

From the overview window, the composer opens the subpatch "CLH-mix" to get access to score mixing functionality. The idea of CLH-mix is to load two sections of bach.roll type scores, from the respective shelf menus, into "Roll 1" and "Roll 2"and be able to mix them in one of four ways: Append; Mix; Interpolate; and Insert. Clicking on "dump" results in the mixture appearing in the "Mix" bach.roll pane. The composer can then enter a new section name for the mix and store the mix under that name in the shelf. Clicking "write" stores the shelf to disc.

Figure 9 shows the layout for the CLH-mix sub-patch. It helps to clear out everything before starting to mix, and the various "clear" and "clearmarkers" buttons facilitate starting with a blank canvas. It is also wise to "read" the shelf from the most recently stored json file before commencing.

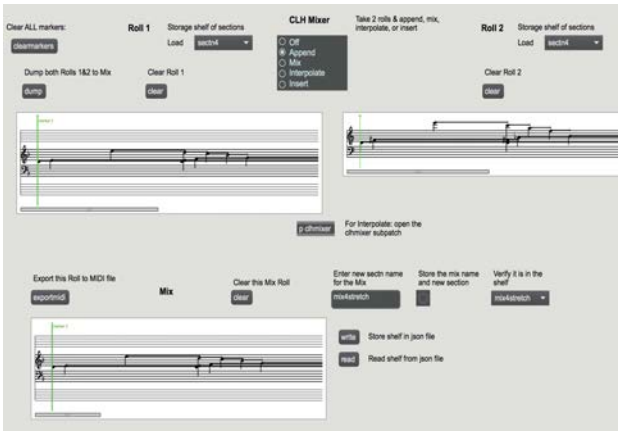

Figure 9. CLH-mix mixes Roll 1 and Roll 2 into the Mix bach.roll via a mix method selected using radio buttons.

Note that a sub-patch called "clhmixer" contains the GUI interface for the more exotic interpolate and insert procedures. Figure 10 shows the opened clhmixer sub-patch and, whilst the details of the patch may not be very clear, the reader may be able to note that a function graph is used to perform an interpolation between Roll 1 and Roll 2. Time is the horizontal axis, using the roll with the longest time. Interpolating coefficient (0 to 1) is on the vertical axis and it is used to determine the amount of each contributing score at each time point.

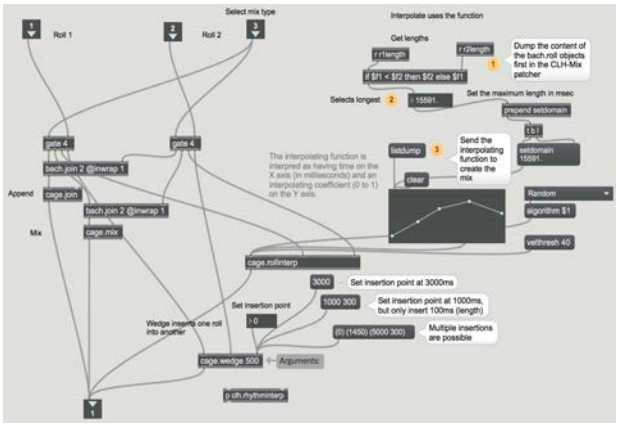

Figure 10. clhmixer performs an interpolation between two bach.roll scores according to a specified function.

Figure 11 shows the result of an interpolation between two sections of music represented as Roll 1 and Roll 2 (in the Mix pane).

Liberal use of the "Append" function can result in an entirely new piece consisting of the assembly of various desired sections. That new assembly can, in turn, be stored under its own name in the shelf menu, which is then saved to a new json file using "write". Such actions prepare the way for the final part of the process which is turning a complete bach.roll into a musical score.

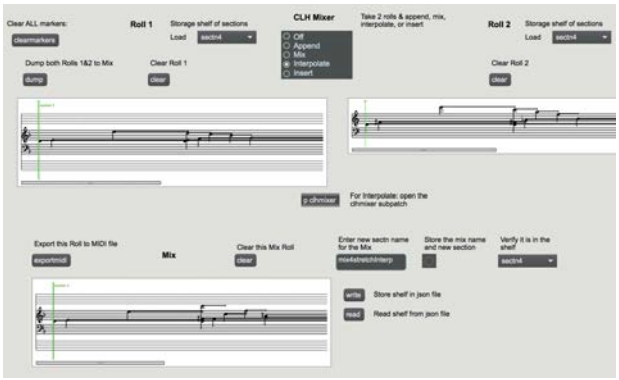

Figure 11. The result of an interpolation between two bach.rolls.

3.5. CLH-roll2score

The final sub-patch, CLH-roll2score, takes a bach.roll representation of the composer's final music mix and renders it into a bach.score standard Western music notation score, which can then be exported to a music XML format file for further tweeking in a specialised music notation program.

Figure 12 is a screen shot of CLH-roll2score. The composer starts by clearing Roll 1 and Score 1, then reading the shelf of segments from a json file. The final "mix" is selected from the shelf menu, the score's time signature and a tempo can be set by editing the values in the time signature button in the Score 1 area. Then Roll 1 is sent to Score 1 by clicking the "quantize" button in the Roll 1 area. The score can be exported to a music XML file and also to a MIDI file too.

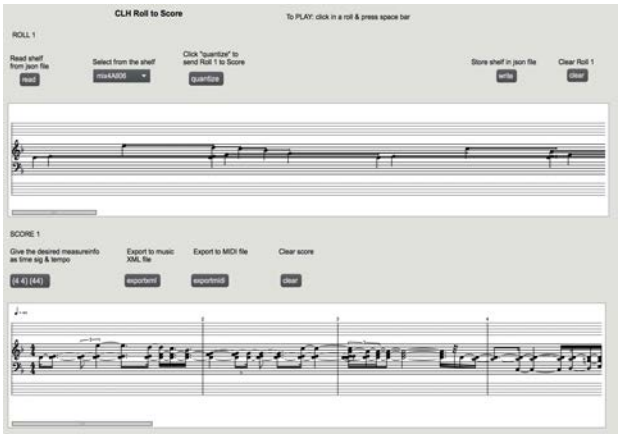

Figure 12. Screen shot of CLH-roll2score.

CLH-roll2score is the least developed of all the sub-patches, and it needs further development. Hans Tutschku has a method of editing the bach.score using text to represent bar number, meter and tempo. He can then reference the bar number to set the meter and tempo in order to render the score into a more readable form. A similar scheme could be developed and tested for the Composer's Little Helper.

4. FURTHER DEVELOPMENTS AND CONCLUSION

At the time of writing, further developments to the Composer's Little Helper would include the addition of Music XML file reading and tidying up the CLH-roll2score score creation to make a score that improves readability. Further extensions of the system include making more use of the "Dada" library (Ghisi & Agon, 2016) for graphic scores or generative systems. Figure 13 shows an example of the current clhtreatments sub-patch replaced by the dada.machines object to create a network of treatment machines like the random one shown in this example.

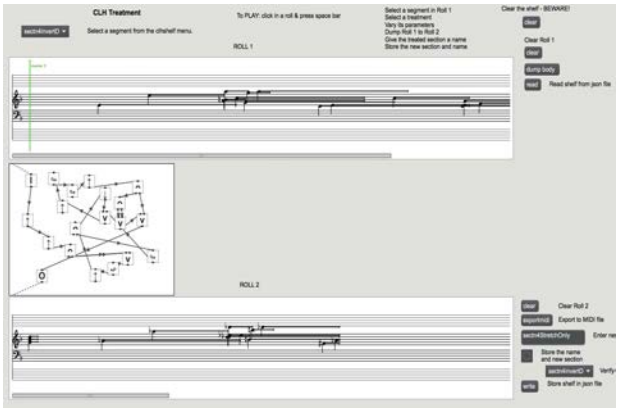

Figure 13. Treatments via a dada.machine network.

5. REFERENCES

Agon, C. 1998. "OpenMusic: Un langage visuel pour la composition musicale assiste´e par ordinateur." PhD dissertation, Universite´ Paris VI.

Agostini, A. and D. Ghisi, 2015. "A Max Library for Musical Notation and Computer-Aided Composition", *Computer Music Journal*, Volume 39, No. 2, p. 11–27, 2015. (Bach)

Agostini, A. Daubresse, E. and D. Ghisi, 2014. "*cage*: a high-level library for real-time computer-aided composition", *Proceedings of the International Computer Music Conference* (ICMC), Athens.

Bresson, J. and J. Chadabe. 2017. Interactive Composition: New Steps in Computer Music Research, *Journal of New Music Research*, 46:1, 1-2.

Ghisi, D. and C. Agon, 2016. "Real-Time Corpus-Based Concatenative Synthesis for Symbolic Notation", *Proceedings of the TENOR conference*, Cambridge, UK. (Dada)

Laurson, M., and M. Kuuskankare. 2002. "PWGL: A Novel Visual Language Based on Common Lisp, CLOS and OpenGL." In *Proceedings of the International Computer Music Conference*, pp. 142–145.

Puckette, M. 2004. "A Divide Between 'Compositional' and 'Performative' Aspects of Pd." In *Proceedings of the First International Pd Convention*. Available online at puredata.info/community/conventions/convention04 /lectures/tk-puckette/puckette-pd04.pdf/at download /file. Accessed July 2018.

Taube, H. 1991. “Common Music: A Music Composition Language in Common Lisp and CLOS.” *Computer Music Journal* 15(2):21–32.

Tutschku, H. 2015. “using max.bach and max.cage” Web page with 4 videos: http://www.tutschku.com/using-max-bach-and-max-cage/ [Accessed 17 July 2018]