ICLI
PORTO
2018

liveinterfaces.org

Trans-Disciplinary Tools for Collaborative, Choreographed, and Embodied Audio-Visual Live Coding

Renick Bell[1]
renick@gmail.com

Joana Chicau[2]
joanabcq@gmail.com

[1]Tokyo, Japan
[2]Rotterdam, The Netherlands

Abstract

Seeking balanced and mutual interaction, the authors designed and implemented tools to connect a live coding system for audio built in Haskell with Javascript tools for live coding browser-based visuals to enable a collaborative audio-visual performance. Each system generates and emits OSC messages through functions developed by the authors and triggered by preexisting functions in those systems. The systems also gained subsystems for receiving incoming messages and modifying system state according to those messages. Means for displaying transmitted data were also implemented, allowing audiences greater insight into performer interactions. The system was designed to enhance the possibility of equal dialogue between the performers and avoid disastrous changes to a partner's system state. It was developed following an ongoing research and recollection of musical and choreographic scores that reference principles of non-linear composition, non-hegemonic time and space constructs, and techno-feminist perspectives.

Keywords
Live coding
Audio-visual performance
Collaboration
Choreography
Web environments
Techno-feminism
Improvisation

Introduction

This research focused on techniques for achieving a philosophically-grounded collaborative improvisation between a live-coded visual system and a live-coded audio system and the corresponding new software tools required to support that improvisation. This paper first explains the philosophical basis for the collaboration which underlies the techniques and tools, then describes the technical goals which the authors targeted. It proceeds to examine existing works which informed this research. The two systems involved are described briefly, followed by a more technical discussion of how the two systems were made to interact and the tooling necessary for that interaction. The paper concludes with a brief analysis of the results of this research.

1.Philosophical Intention

The authors began this research with the intention of collaborating in a way that reflects developments in theory on feminism, interfaces, and live coding to achieve the maximum balance in collaboration possible while still achieving particular technical and aesthetic goals.

Live coding performances, like other forms of improvisation, are a constant negotiation between different forms of agency and computers. Schroeder writes that “Live coding practitioners ask the audience to share the risk and the fascination of live making. By emphasising the risk of such making, these practices deliberately expose the body in flux, the body in constant negotiation with the environment and the instrument, itself in flux.” (2009) While acknowledging and accepting this risk, certain technical choices can be made to reduce some of those risks, correspondingly granting additional freedom in different areas.

It is also important to consider the role of the environment, as described by Rodaway: “... The concept of ecological optics (and ecological formulations of other sensuous information) emphasises the role of the environment itself in structuring optical (auditory, tactile, etc.) stimulation. Potential sources of stimulation pass through the environment and are encoded with the structure of that environment as they are modified in their passage. It is this structured stimulus which the sense organ ‘read’. Therefore, the environment becomes a source of information, not merely raw data.” (Rodaway, 2002) In a collaborative performance, the other performer can also be seen as part of a performer’s environment.

The authors sought to reflect and practice alternative uses of technologies and the pursuit of new resolutions. Every decontextualized materiality may be immediately re-contextualized inside another already existing paradigm or interface. In *The Interface Effect*, Galloway writes that the interface is “not a thing, an interface is always an effect. It is always a process or translation” (2013) Users may completely depend on their conditioning every time they deal with data, so the possibility of escaping the normative or habitual interpretation of interfaces was of interest. The authors intended to include the activity of the other in a deeper way than simply reacting to how that performance is perceived through the five senses. In this way, the data flow from the other as part of the total environment becomes a central part of the structure of each performer’s output.

The authors aimed to expose the process, making digital literacy and experimental tools part of their strategies. TOPLAP has long called for live coders to “show us your screens” (McLean, 2010). The authors intended to take this still-radical concept of the openness of the performer further by showing how the data of the other is actively affecting each performer’s activity.

This approach of each system modifying the other is built on feminist pursuit of “decentered, multiple, participatory practice(s) in which many lines of flight coexist.” (Galloway, 2013) The platforms and tools chosen to integrate this performance, for example the OSC feature later explained in detail, allow for constant and immediate interactions, intending to remove hierarchy, which necessarily means eliminating patriarchy, and reflecting the fact that influencing interactions necessarily involve being influenced in turn. Rather than an enforced equalising of roles so that there is a one-to-one matching of influence, the authors sought to achieve a balance of control appropriate to each situation. Through the data-level connection between the systems, a dialogue can be carried out, and an additional dimension of performativity is opened.

Through the appropriation of interfaces not originally intended for performance, as well as the creation of new vocabularies that form plural and therefore more inclusive views, feminist practices were in part brought into the project. Allowing possibly the world's most common interface, the web browser, to communicate with a very specific interface in the form of the custom Haskell interface, shows that inclusivity. Using feminist perspectives in the interface design means redefining what efficiency and functionality mean.

2.Technical Interaction Goals

There were several key concerns in developing the interaction and tools to make it possible. The principal concern was to enable a balanced interaction between performers; exactly what "balanced" means depends on the demands of particular situations. In accordance with the goal of allowing the other's data to become information for each performer's system, methods were required to pass that data and then make it meaningful.

Some additional concerns related to the interaction included how the performers could switch roles. Different types of activity levels were targeted: being active, being passive, being active simultaneously, having multiple agents active while the performers themselves are not, and so on.

The authors also examined how to avoid demolishing a co-performer's work when domain knowledge was insufficient, which in part reflects the de-emphasis on skill, removing some risks in order to allow more emphasizing mutually supporting communication. This required the authors to consider strategies and means for mapping data flows and then ways to quickly recover when unwanted state modifications take place.

The authors have not always relied on the same set of rules and interactions between the two systems. For example, wait times in one system might be remapped to spatial parameters in another, or an array of strings might be remapped to a graph of parameter values in another. The authors sought as much freedom of mapping as possible within the encompassing technical constraints of the systems involved. Messages could be urgent and acted upon immediately, or they could be deferred and acted upon when the context became appropriate.

The authors were also keen to avoid a mechanical correspondence between the two systems. For example, it was not our intention to make the visual system pulse perfectly in sync with the rhythms presented by the audio system. Deep ways to map the data were sought, yet some mappings that would still be obvious to the audience were also sought so that the audience would not just be aware of the interaction between the two performers but also might be able to follow it to at least a limited degree.

The goal of openness described above led to exploration of ways to reveal to the audience the nature of the interaction as it unfolds through a performance. This required display of those flows and their effects for not just the performers but also to the audience. Because the interfaces of the two systems were different, different means for displaying those flows were required.

Finally, all of the above goals had to be reached while still making sure that the tools function adequately for real-time performances, meaning avoiding unacceptable jitter, delays, glitches or other unwanted system malperformance.

3.Literature review

Collaborative live coding is not new. A number of performers have developed group practices and systems to enable their audio performances. Those include groups such as:

- The Hub (Gresham-Lancaster, 1998)
- OFFAL, "a non-hierarchical collective [aiming] to connect an international group of women engaged in electronic music by developing technological systems and organisational structures that facilitate collaboration." (2018)
- BiLE (Birmingham Laptop Ensemble) (2018)
- Benoit and the Mandelbrots, who use their own BenoitLib and MandelHub (Borgeat, 2010)
- Various groups using David Ogborn's collaborative editor Estuary (Ogborn, 2017)
- Live coding group Glitchlich, which used SuperCollider and their own bespoke tools written in C++

Some systems allow collaboration between audio and visual live coders, such as Charlie Roberts's *Gibber* (2012). There have been some live coders whose practices involve choreography, such as the work of Kate Sicchio (2018) and some pieces by Marije Baalman (2018). However, the authors are unaware of data sharing via OSC between two different live coding environments for the purpose of executing a collaborative audio/visual performance with choreography.

4.Audio System

The live coding system in Haskell (Jones, 2002) uses a text editor and ghci with a SuperCollider audio back-end (the SuperDirt sampler, which is a port of Alex McLean's Dirt sampler to Super-Collider done by Julian Rohrhuber (McLean, 2018)), to which the system communicates through OSC (Wright, 2005). OSC is handled by the hosc package written by Rohan Drape (2010). The audio system uses more than 10 autonomous processes which, in addition to triggering audio synthesis events, also change data used in pattern generation, synthesis, and the state of the other autonomous processes. The processes refer to a set of shared data stores containing tables of rhythms, density patterns, sample patterns, parameter patterns, and so on. Each process runs in a loop, executing a side-effect-producing function and then waiting according a timing function each refers to, as long as it is active. Whether it is active or not is determined either by the operator or another autonomous process which has been assigned a function to start and pause other autonomous processes.

5.Visual System

The visual system involves a choreographic score written in web programming languages (HTML, CSS, JavaScript). The performer uses the browser (Firefox) console to write functions that draw on choreographic concepts and use both local files and already existing interfaces, such as Google search, to explore different functionalities of online interfaces. The performer live codes in Javascript in the web browser, embedding new canvas elements and manipulating various visual elements text, images, modifying in real time sourced web pages, and reading from JSON data stored on the local disk. OSC is handled by Node.js and the osc.js library (Clark, 2014).

6.Inter-system Communication and Interaction System

To achieve the collaboration goals described above, some tools for dealing with OSC messages were developed. OSC messages were designed according the approach described above. The messages are structured to show where they come from, in the following manner:

```
/(audio or visual)/processA/messageType [
…(an optional array of floats or strings
 to be received by a system)...]
```

"Audio" is replaced with "visual" in the case of messages coming from the graphical system to the audio system. With such a message structure, it is possible for each system to respond to the data in an appropriate manner as determined by the coder/performer. The message types reflect where the data was taken from.

Messages belong to one of three types: trigger messages, arrays of numbers, or arrays of strings. The effect is that one performer has passed to the other a critical piece of its inner activity, which the other is free to react to in any way. For example, the audio system might

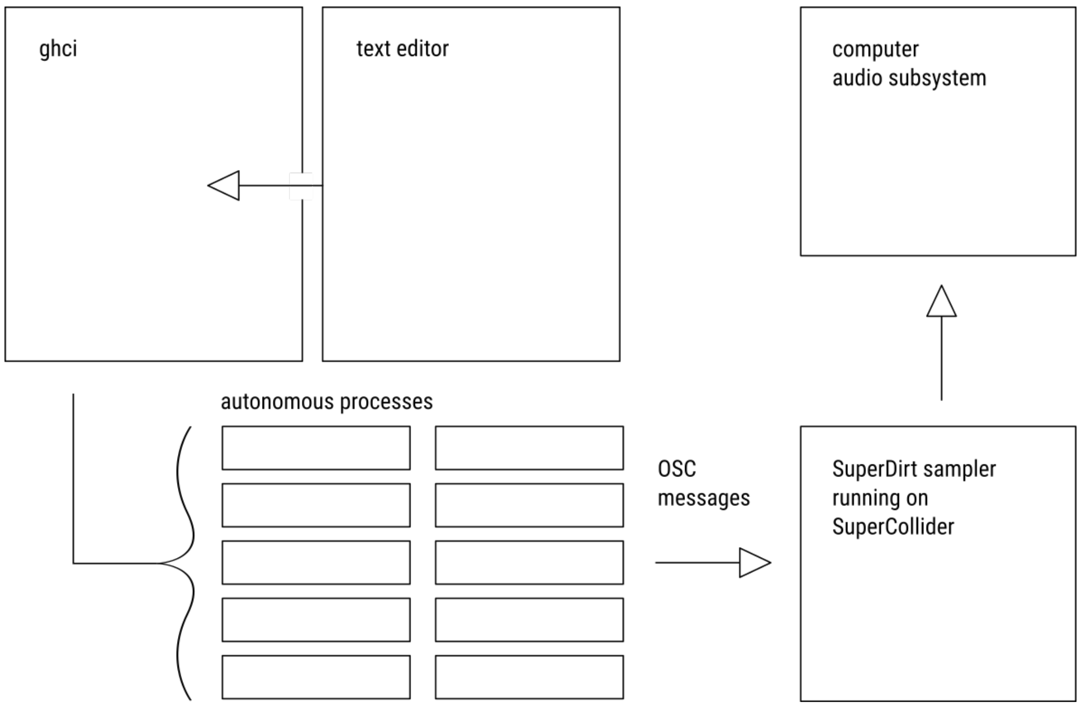

Figure 1. Audio system

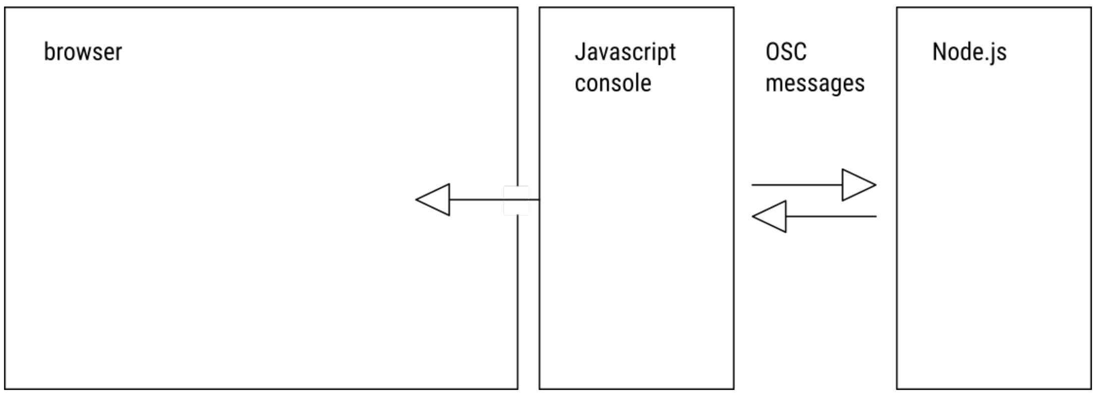

Figure 2. Visual system

send a “rhythmTableRow” message, which is followed by the delta values contained in the selected row of a selected rhythm table. The coder of the visual system is then free to use that data in any manner deemed useful to the performance, such as using these delta times to determine refresh rates to a line of text which changes periodically.

Some examples messages include:

```
/audio/playerKick/trigger

/audio/density [ 0,1,16,1.4,20,1.2,
28,1.7,31.9,1.9] <- a linked listed
displayed as a flattened array

/visual/browserWindow1/trigger

/visual/browserWindow1/waitTimes
[1,1,2,1,1,3]

/visual/searchStrings [“The center
of”, “Spheres”, “Equidistant”]
```

Mapping is flexible and can be decided at performance time by the performer so that it becomes an element of the improvisation. However it is also possible to pre-map incoming data before the performance, and the preparation of various mapping functions before performing makes use of the data in performance safer; in rehearsals safe mappings can be decided that would allow those messages to be passed and executed without disastrous changes in system state or otherwise negatively influencing the performance. Users are able to map the messages so that critical data is protected and that the effects produced are within a safe range.

Some functions were developed in order to reshape data for different uses, such as normalizing values to usable ranges or converting string data to numeric data and vice versa. Received messages can also be handled in two different styles: immediate dispatch or dispatch according to sequence. The meaning of the former should be clear; in the case of the latter, messages are queued and dispatched in order of arrival according to the timing of a sequence determined by the operator of the audio system.

An OSC listening/sending subroutine for the audio system was implemented using hosc. Received messages are interpreted by the listener and displayed in the interpreter, which is visible to the audience in one terminal. Those received messages then trigger corresponding functions which in turn modify or replace the various stateful data of the system. For example, a trigger message can be used to force a change in rhythms, or wait times can be interpreted as a graph for density to be used by some or all of the autonomous processes. An OSC listener for the visual system was implemented in osc.js. It involves a Node.js-based listener which receives messages from the network and passes them to the visualization system running in the browser.

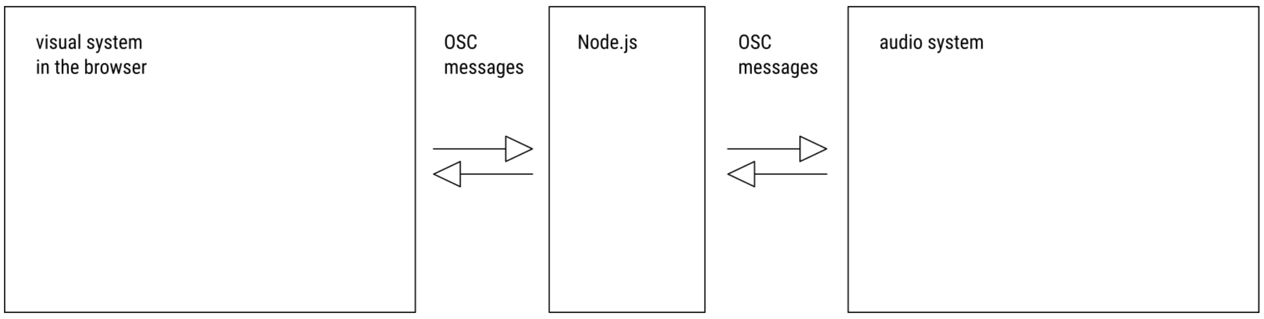

Figure 3. Inter-system communication

The visual system might map incoming messages to things like delta times in page animation, spacing of graphical items, or angles used to skew graphical items.

At the same time, the audio system sends OSC messages to the visual system. Autonomous processes following the same rhythms and densities as the processes triggering sample playback send messages via OSC to the visual system, where they are received by a corresponding listening server. The functions for doing so are designed so that different processes can send different types of messages and at different timing. The visual system has OSC-message-passing functions built into standard functions used for modifying the browser environment so that through normal operation, OSC messages are passed to a node-based server which then sends the messages via OSC to the audio system.

Revealing the interaction required using meaningful naming conventions and crafting messages to be shown in always-visible post windows. Those messages also required particular highlighting so that they would not be lost amidst the data present in those post windows.

The visual system displays the received messages in the DOM as alerts or as text messages in the browser to highlight the communication, while special text formatting was used for received messages in the Haskell interpreter so that those messages would be more visually emphasized than other messages that appear in that window.

7.Evaluation

Basic technical goals were achieved. While there are some advantages to including the functions to send OSC messages to the partner system in embedded functions that are triggered during standard system operation, it was decided finally not to do so in order to allow for a more flexible and therefore timing-appropriate usage of the message-passing functions. It is still challenging to adapt to unfamiliar data from the other system in real-time. It is worth investigating whether this is a matter of practice or if technical solutions can reduce the difficulty of doing so. The authors settled on a simplified message system in which the origin and a string argument are passed. This was done for two reasons. The first of which was that the specification of the more complex message in the performance was too demanding and error-prone; becoming familiar with a more complex form would take more rehearsal time than was available for the initial peformance using the system. In addition, the authors decided on an explicit shared vocabulary for conceptual and aesthetic reasons. This shared vocabulary then was given a behaviour in each system. For example, "spacing" in the visual system increased the spacing between letters in text, while in the audio system it increased the distance in time between events. These were defined in advance so that they could be used more immediately in the performance according to timing chosen by the users. Most in-performance mapping was dropped for the performance in order to keep the pace of the performance fast enough to meet aesthetic goals and avoid errors. If mapping is to be done dynamically in a performance, faster methods will be required. Queueing of messages was also not used in order to maintain the transparency of one performer passing a message to the other and then immediately causing a change in the other system.

Conclusion

The authors intend to use this system for a number of performances, testing it further to determine whether it achieves the goals outlined above, how those goals might be revised, and how the system can then be adapted further to meet those goals. The implementation also remains very specific to the two systems involved; future work includes generalizing the system and documenting it so that it could be more easily used by others for their performances.

Baalman, Marije. “Code LiveCode Live.” n.d. Accessed February 28, 2018. https://marijebaalman.eu/projects/code-livecode-live.html.

Borgeat, Patrick. (2010) 2017. *BenoitLib: SuperCollider Extensions Used by Benoît and the Mandelbrots.* SuperCollider. https://github.com/cappelnord/BenoitLib.

Clark, Colin. (2014) 2018. *Osc.Js: An Open Sound Control (OSC) Library for JavaScript That Works in Both the Browser and Node.Js*. JavaScript. https://github.com/colinbdclark/osc.js.

Drape, Rohan. 2010. *Hosc 0.8.*

Galloway, Alexander R. 2012. *The Interface Effect.* Polity.

Gresham-Lancaster, Scot. 1998. “The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music.” *Leonardo Music Journal* 8: 39. https://doi.org/10.2307/1513398.

Jones, Simon P., ed. 2002. *Haskell 98 Language and Libraries: The Revised Report.* http://haskell.org/. http://haskell.org/definition/haskell98-report.pdf.

“Manifesto | BiLE.” n.d. Accessed February 28, 2018. http://www.bilensemble.co.uk/manifesto/.

McLean, Alex and Rohrhuber, Julian. “Musikinformatik/SuperDirt.” n.d. Accessed February 28, 2018. https://github.com/musikinformatik/SuperDirt.

McLean, Alex. 2010. “TOPLAP Website.” TOPLAP. September 2010. http://www.toplap.org/index.php/Main_Page.

“OFFAL by Offal.” n.d. Accessed February 28, 2018. https://offal.github.io/.

Ogborn, David, Jamie Beverley, Luis Navarro del Angel, Eldad Tsabary, and Alex McLean. n.d. 2017. “Estuary: Browser-Based Collaborative Projectional Live Coding of Musical Patterns.”

Roberts, Charlie, and JoAnn Kuchera-Morin. 2012. “Gibber: Live Coding Audio in the Browser.” In ICMC.

Rodaway, Paul. 2002. *Sensuous Geographies: Body, Sense and Place.* Routledge.

Schroeder, Franziska, and Pedro Rebelo. 2009. “The Pontydian Performance: The Performative Layer.” *Organised Sound* 14 (2): 134–141

Sicchio, Kate. “Sound Choreographer Body Code – Kate Sicchio.” n.d. Accessed February 28, 2018. http://blog.sicchio.com/works/sound-choreographer-body-code/.

Wright, Matthew. 2005. “Open Sound Control: An Enabling Technology for Musical Networking.” *Organised Sound* 10 (3): 193–200.