Proceedings of the 17th Linux Audio Conference (LAC-19), CCRMA, Stanford University, USA, March 23–26, 2019

SC-HACKS: A LIVE CODING FRAMEWORK FOR GESTURAL PERFORMANCE AND ELECTRONIC MUSIC

Iannis Zannos

Department of Audiovisual Arts
Ionian University, Corfu, Greece
zannos@gmail.com

ABSTRACT

This paper presents a library for SuperCollider that enables live coding adapted to two domains of performance: telematic dance with wireless sensors and electroacoustic music performance. The library solves some fundamental issues of usability in SuperCollider which have been also addressed by the established live-coding framework JITLib, such as modifying synth and pattern processes while they are working, linking control and audio i/o between synths, and generation of GUIs. It offers new implementations, which are more compact and easy to use while emphasizing transparency and scalability of code. It introduces binary operators which when coupled to polymorphism facilitate live coding. Several foundation classes are introduced whose purpose is to support programming patterns or commonly used practices such as the observer pattern, function callbacks and system-wide object messaging between language, server processes and GUI.

The use of the library is demonstrated in two contexts: a telematic dance project with custom low-cost movement sensors, and digital implementations of early electroacoustic music scores by J. Harvey and K. Stockhausen. The latter involves coding of a complex score and generation of a GUI representation with time tracking and live control.

1. BACKGROUND

1.1. Bridging Live Coding and Gestural Interaction

The performance practice known as *live coding* emerged from the ability of software to modify state and behavior through the interactive evaluation of code fragments and to synthesize audio at runtime. As a result, several programming environments and technologies supporting live coding have been developed in the past 20 years, such as *SuperCollider*[1], *Impromptu*[2], *ChucK*[3] , *Extempore*[4], *Gibber*[5], and others. It has been noted, however, that such environments and practices suffer from a lack of immediacy and those visible gestural elements that are traditionally associated with live performance [6]. Recent research projects attempt to re-introduce gestural aspects or to otherwise support social and interactive elements in musical performance using technologies associated with live coding ([7], [8], [9], [10]). Amongst various types of gestural interaction, dance is arguably the one least related to textual coding. Few recent studies exist which prepare the field for bridging dance with coding ([11]). The challenges in this domain can be summarized as the problem of bridging the symbolic domains of dance and music notation and the subsymbolic numerical domain of control data streams input from sensors. This also implies translating between continuous streams of data and individual timed events, possibly tagged with symbolic values. This is a technologically higly demanding task which is subject of research in various gestural interface applications. The work related in the present paper represents an indirect and bottom-up approach to the topic, based on DIY and open source components and emphasizing transparency and self-sufficiency at each step. It does not address the task of gesture recognition, but rather it aims at supporting live coding in conjunction with dancers and instrumental performers. Ongoing experiments together with such performers, are helping to identify low-level tasks and features which are essential for practical work. This type of work is purely empirical, and tries to identify useability criteria purely from practice, rather than to develop features that are inferred from known interaction paradigms in other related domains. At this stage of the project it is still too early to formulate conclusions from these experiments. Instead, this paper concentrates on the fundamentals of the implemenation framework on which this work is based. These are readily identifiable and their potential impact on further development work as well as experiments are visible. This paper therefore describes the basic principles and design strategy of the *sc-hacks* library, and discusses its perceived impact on performances. Finally, it outlines some future perspectives for work involving data analysis and machine learning.

1.2. Live Coding Frameworks in SuperCollider

1.2.1. Types of Live Coding Frameworks

Live Coding libraries can be divided into two main categories depending on the level of generality of their implementation and their application scope. First, there are libraries which extend *SuperCollider* usage in order to simplify the coding of very behaviors or features which are very common in performance, but are otherwise inconvenient to code in *SuperCollider*. To this category belongs the JITLib framework. JITLib (Just-In-Time programming Library) has been around since at least August 2006, with an early version since ca 2000 [1] and is very widely used in the community, being the de-facto go-to tool for live coding in *SuperCollider*. The second category consists of libraries that concentrate on specialized usage scenarios and attempt to create domain-specific mini-languages for those scenarios on top of *SuperCollider*. Such are: *IXI-Lang* (a sequencer / sample playing mini-language by Thor Magnusson [12]), *SuperSampler* (a polyphonic concatenative sampler with automatic arrangement of sounds on a 2-dimensional plane, by Shu-Cheng Allen Wu [13]), and *Colliding* (An "environment for synthesis-oientd live coding", simplifying the coding of Unit Generator graphs, by Gerard Roma [14]). Finally, *TidalCycles* by Alex McLean [15] should be mentioned, which develops its own live coding language based on Haskell and focussing on the coding of com-

[1] See https://swiki.hfbk-hamburg.de/MusicTechnology/566 (accessed 20-December-2018)

plex layers of synchronized beat cycles with sample playback and synthesis, and uses the SuperCollider synthesis server as audio engine.

1.2.2. sc-hacks *Objectives and Approach*

sc-hacks belongs to the first category of frameworks, and its initial motivation was partly to implement some of the solutions of JITLib in more robust, simple, and general ways. In parallel, inspiration from *ChucK*'s => operator led to the development of a minimal extension of the language based on 4 binary operators (`+>`, `<+`, `*>` `*<`), which, coupled with polymorphism, permit simplified and compact coding of several common sound-structure coding patterns. Furthermore, the implementation of some basic programming patterns [2] opened new possibilities for the creation of GUI elements which update their state. This led to a proliferation of GUI building and management facilities and resulted in several interfaces for live coding tasks, such as a code browser based on the concept of code snippets, a browser for editing and controlling the behavior of named players holding synth or pattern items, and shortcuts for building GUI widgets displaying values of parameters controlled by OSC, MIDI or algorithmic processes. Finally, ongoing experiments with dancers and instrumentalists are giving rise to new interface and notation ideas. The current focus is on building tools for recording, visualising and playback of data received from wireless sensors via OSC, in order to experiment with the data in performance, and to apply machine-learning algorithms on them.

2. APPROACH

2.1. Players and Player Environments

JITLib addresses four fundamental problems in coding for concurrent sound processes: (a) Use of named placeholders for sound generating processes, (b) managing the control parameters of processes in separate namespaces, (c) modifying event-generating algorithmic processes (known in SuperCollider as *Patterns*) on the fly and (d) interconnecting audio signals between inputs and outputs of synth processes. Sc-lib offers alternative solutions to these problems which present advantages, described in the following sections:

2.1.1. Named placeholders: -def classes vs. Player class

To use a name as placeholder for a synth process in order to start, stop or modify the process on the fly, JITLib introduces the *[X-]def* convention, i.e. it defines a number of classes which act as named containers for different types of processes (Synths: `Ndef`, Tasks: `Tdef`, Patterns `Pdef`, etc.). Sc-hacks uses a single `Player` object class instead. A Player instance can play a Synth or a Pattern depending on the type of source which it is asked to play, i.e. synth definition, synth function, or event-stream generating instance (see for example code below 3). This provides greater flexibility and simplicity in the coding of synth processes over JITLib.

2.1.2. Separate parameter namespaces: ProxySpace vs. Nevent

A significant innovation introduced by JITLib consisted in the concept of a ProxySpace, that is, a namespace that can function as the current environment. ProxySpace is based on EnvironmentRedirect, a Class which holds a Dictionary and ensures that a predefined custom function is executed each time that a value is stored in one of the keys of the Dictionary. Sc-hacks defines a subclass of EnvironmentRedirect similar to ProxySpace, but defines a custom function that provides extra flexibility in setting values which is useful during performance in accessing control parameters. This enables keeping track of which parameter refers to which process, storing parameter values between subsequent starts of a process belonging to a player, and updating GUI elements to display values as these change. Additionally, sc-hacks makes the environment of the player current after certain operations, in order to make the current context the one normally expected by the performer. This however is not always a secure solution. For this reason, the target environment can be provided as adjective argument in binary operators involving players, which ensures that code will work as expected even when changing the order of execution of code in irregular manner.

2.1.3. Modifying event generating processes on the fly

Event generating algorihm processes are implemented in SuperCollider through class Pbind. Pbind takes an array of keys and associated streams as argument and creates a Routine that calculates parameters and event types for each set of keys and values obtained from their associated streams, and schedules them according to the duration obtained from the stream stored under the key `dur`. The implementation of Pbind allows no access to the values of each event, i.e. it is not possible to read or to modify the value of a key at any moment. Furthermore, it is not possible to modify the structure of the dictionary of keys and streams while its event-generating process is playing. This means that Pbind processes cannot be modified interactively while they are playing. In order to circumvent this limitation, a number of techniques have been devised which require to add code for any key that one wishes to read or to modify. JITLib uses such techniques and also provides a way to substitute a Pbind process while it is running with a new one, thereby indirectly allowing modification of that process. Sc-hacks provides a new approach for playing event-generating processes, which uses the same Event-playing mechanism as Pbind, but grants both read and write access to the data which generate the event stream, and thus permits modification of the generating key-stream collection on the fly. This radically simplifies the task of modifying event generating processes while they are playing. For example, adding or substituting key-value stream pairs to a process while it is playing can be achieved simply by sending the corresponding key-stream pairs as events to the same player, as shown in the following code 1.

```
(dur: 0.1) +> \mystream;
// Substitute duration stream:
(dur: [0.1, 0.2].prand) +> \mystream;
// Add degree stream:
(degree: (-10..10).prand) +> \mystream;
```

Figure 1: *Adding and substituting key streams to event generators.*

2.1.4. Interconnecting audio signals

The task of connecting the output of one audio process with the input of another audio process is complicated in SuperCollider by the requirements (a) to specify the bus which will carry the signal to be shared and (b) to ensure that the synth reading from the signal will be placed *after* the bus which is writing to the signal in the execution

[2] See for example the Observer pattern: `https://en.wikipedia.org/wiki/Observer_pattern` (accessed 20-December-2018)

order of the synth engine (scsynth). The implementation of the solution in JITLib involves several classes with several instance variables and hundreds of lines of code and defies description within the scope of the present paper. Additionally, coding the configuration of one-to-many or many-to-one interconnections of audio i/o between synth processes can be both verbose and complex, as witnessed for example in exchanges on the SuperCollider mailing list such as this one: `https://sc-users.bham.ac.narkive.com/PAapaSaM/many-to-one-audio-routing-in-jitlib` (accessed 20-December-2018). Sc-hacks introduces a new solution which permits simpler coding and guarantees persistence of established configurations even when the server is rebooted during a work session. The implementation is based on mechanisms for hierarchical namespaces and function callback implemented in sc-hacks through two new classes discussed below: *Registry* and *Notification*. The coding of one-to-many and many-to-one connections is exemplified through the following code 2:

```
// many - to - one interconnection
\source1 *> \fx1;
\source2 *> \fx1;
// one - to - many interconnection
\source3 *< \fx2;
\source3 *< \fx3;
```

Figure 2: *Interconnecting audio signals.*

Note that no additional coding is required if using the default input and output parameter names `\in` and `\out` and number of channels (1). PersistentBusProxy is used to specify custom parameter names and channel numbers. The operator `@` can optionally be used as shortcut to create PersistentBusProxy instances.

2.2. Binary operators

The primary coding strategy of *sc-hacks* for sound processes is built around a small number of binary operators. Each operator encapsulates a group of actions on sound objects such as synthesis parameters, player objects holding single synths or synth processes, busses, buffers, midi or osc control instances. The operators are:

left operand	operator	right operand
source	+>	player
source	*>	player
parameter	<+	value
parameter	*<	value

2.2.1. *+> : Play source in player*

The +> plays the *source* in the *player*. The source can be the name of a synthesis definition as symbol, a synthesis function, or an event. For example the code in 3 can be evaluated line-by-line to play in the player named `'example'` in sequence a synth using SynthDef named `'default'`, a Unit Generator Synth Graph containing a Sine Oscillator, an empty event with default parameters (degree: 0, dur: 1), an event with duration 0.1, and an event with degree a pattern using a brownian stream with values between -10 and 10 and maximum step 2. Sending different types of sources (synthdef names, synth functions, events) to the same player will replace the previous source with the newest one. Sending `nil` stops the player.

```
\default +> \example; // play synthdef
{ SinOsc.ar(440, 0, 0.1) } +> \example;
() +> \example; // play event
(dur: 0.1) +> \example; // modify event
(degree: [-10, 10, 2].pbrown) +> \example;
nil +> \example // stop player;
```

Figure 3: *Player operator +>.*

Additionally, sc-hacks permits one to browse the code executed for each player on a dedicated GUI (similar to operations on *Shreds* in the miniAudicle GUI of *ChucK*), to edit existing code and resend it to the player, and to start or stop a player by clicking on its name in the list of existing players, as shown in Figure 4. The list of evaluated code strings is permanently saved on file for each session.

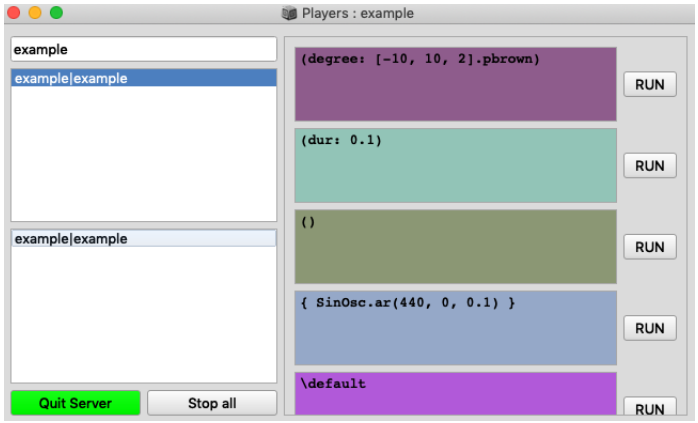

Figure 4: *Player GUI.*

2.2.2. **> : Advanced operations on player argument*

The *> operator takes different meanings depending on the type of the right operand, as follows:

type of left operand	action
Event	set parameter values without starting events
Function	Play function as routine in environment
Symbol	Add receiver as audio source to argument
PersistentBusProxy	Add source with custom i/o mapping

2.2.3. *<+ : Set or map parameter*

The <+ operator acts on the parameter named by the receiver (left operand) depending on the type of the argument (right operand), as follows:

type of right operand	action
Integer or Float	Set parameter value
Symbol	Map parameter to named control bus
Envelope	Map parameter to envelope signal
Function	Map parameter to Synth Function output
MIDI	Bind parameter to MIDI input
OSC	Bind parameter to OSC input

The parameter named by the left operand belongs by default to the current environment. In order to specify a different environment, one can name the environment as an adverb to the binary operator using standard SuperCollider syntax, e.g.: `\freq <+.myenvir 660`.

2.2.4. ⋆<+ : One-to-many audio i/o interconnections

The ⋆< operator, in analogy to ⋆>, is used to create one-to-many i/o interconnections, that is, to connect the audio output from one Player to the inputs of several different Players.

2.3. Fundamental Classes

To implement the above features, sc-hacks introduces classes which implement pattern-language-like features that enable functionality across a wide variety of tasks such as storing and retrieving single instances in tree data structures (Registry Class), updating state of concerned items in response to changes (Notification Class), and enforcing sequential order of execution in asynchronous calls to the server when booting, loading synthdefs and loading or initializing audio buffers (ActionSequence Class). These classes formed the backbone for rapid creation of custom extensions to the library to meet needs of performance requirements described in the next section. These results are encouraging indications that the library will serve as framework to develop more ambitious applications in the next stages of this work.

3. APPLICATIONS

3.1. Telematic Dance

Sc-hacks was first used in a telematic dance project whose goal is to enable dancers to perform together concurrently in different cities by sharing data from motion sensors sent via OSC over the internet [16]. Sensors were constructed using LSM9D0 motion sensor modules and Feather Huzzah ESP8266 wifi modules from Adafruit, and connected to SuperCollider via micro-osc package on micropython. Several sessions with dancers in Tokyo, Athens and Corfu served to experiment with different sound synthesis algorithms and to test the usabiity of the interface and algorithms for dance improvisation. The results were generally more encouraging than expected, except in Corfu where the dancers showed a more cerebral approach emphasizing control over the sound result rather than free exploration of the sonic landscape through movement.

A significant new turn in the development of the library was prompted during the initial tests for remote collaboration performed during a workshop organized at the University of Manchester by Prof. Ricardo Climent for the EASTN-DC EU-Culture program. This showed the need for distributing versions of the library to different remote partners, using different custom settings for each partner. Opening files in the SuperCollider IDE in order to select and execute appropriate code segments was soon proven to be impractical under the pressed time circumstances of preparing the test within a large scale workshop and awkward time-zone difference between the partners involved. Thus, a plug-and-play solution had to be devised, or at least one that relied on selecting options from menus or lists and clicking on buttons rather than opening files and executing code. This gave rise to a new interface as a GUI for selecting and evaluating snippets of code contained within files within subfolders of a global "Snippets" folder 5. The scheme has since served for the archival of experiments and performances, facilitating easy overview and reuse of past code. It is furthermore integrated for use with EMACS as primary IDE for SuperCollider, with automatic updates of code between EMACS and the SuperCollider based GUI.

Two further features were necessary for the experiments with dancers. First, a GUI that displays OSC data as they are received, and second a mechanism that scales and assigns incoming OSC data to the desired parameters. The following code shows how to generate a gui that displays data changes for a set of named parameters. Updates are displayed whenever a parameter is changed, independently of the source of the change (i.e. automated algorithm, evaluation of code, MIDI or OSC input).

```
\lsm1.v(
        \dur.slider([0.1, 12], \lsm1),
        \pos.slider([0.0, 1.0], \lsm1),
        \rate.slider([0.2, 15], \lsm1),
        \gps.slider([0.5, 20.0], \lsm1),
        \pan.slider([-1, 1.0], \lsm1),
        \amp.slider(\amp, \lsm1)
);
```

The GUI in figure 6 was generated by the code above.

Following example shows how to scale data input from OSC messages and to assign them to named parameters in a specified environment `'lsm1'`.

```
\dur <+.lsm1
   '/gyroscope1'.osc(0, [-40, 40], [0.01, 12.5]);
\pos <+.lsm1
   '/gyroscope1'.osc(1, [-20, 40], [0.0, 1.0]);
\rate <+.lsm1
   '/gyroscope1'.osc(2, [-20, 40], [0.1, 15]);
\gps <+.lsm1
   '/magnetometer1'.osc(0, [-1.0, 0.5],
                           [0.2, 15]);
\pan <+.lsm1
   '/magnetometer1'.osc(1, [-0.25, 0.25],
                           [-1, 1]);
\amp <+.lsm1
   '/magnetometer1'.osc(2, [-0.05, 0.25],
                           \amp);
```

The above features are only the beginning. As experiments with dancers have shown, other GUIs and coding schemes are needed to facilitate adjustment of the responsiveness of the sensors and adaptation of their sound control aspects during performance. In this respect a considerable amount of work is still required.

3.2. Coding Electroacoustic Music Performances

A second test scenario was provided through the collaboration with Dan Weinstein, a concert cellist specializing in contemporary music performance with good knowledge of contemporary audio tools in Linux. Mr. Weinstein selected two pieces from the early repertory of electroacoustic music scored for tape recorder: Jonathan Harvey's "Ricercare una melodia" and Karlheinz Stockhausen's *Solo 19*. Both pieces had to be coded in SuperCollider and rehearsed within one week during a residency of Mr. Weinstein in Corfu, leading to a public performance of the pieces. The time constraints were critical because the pieces were both complex and demanding in terms of score interpretation, following and coordination. The Stockhausen

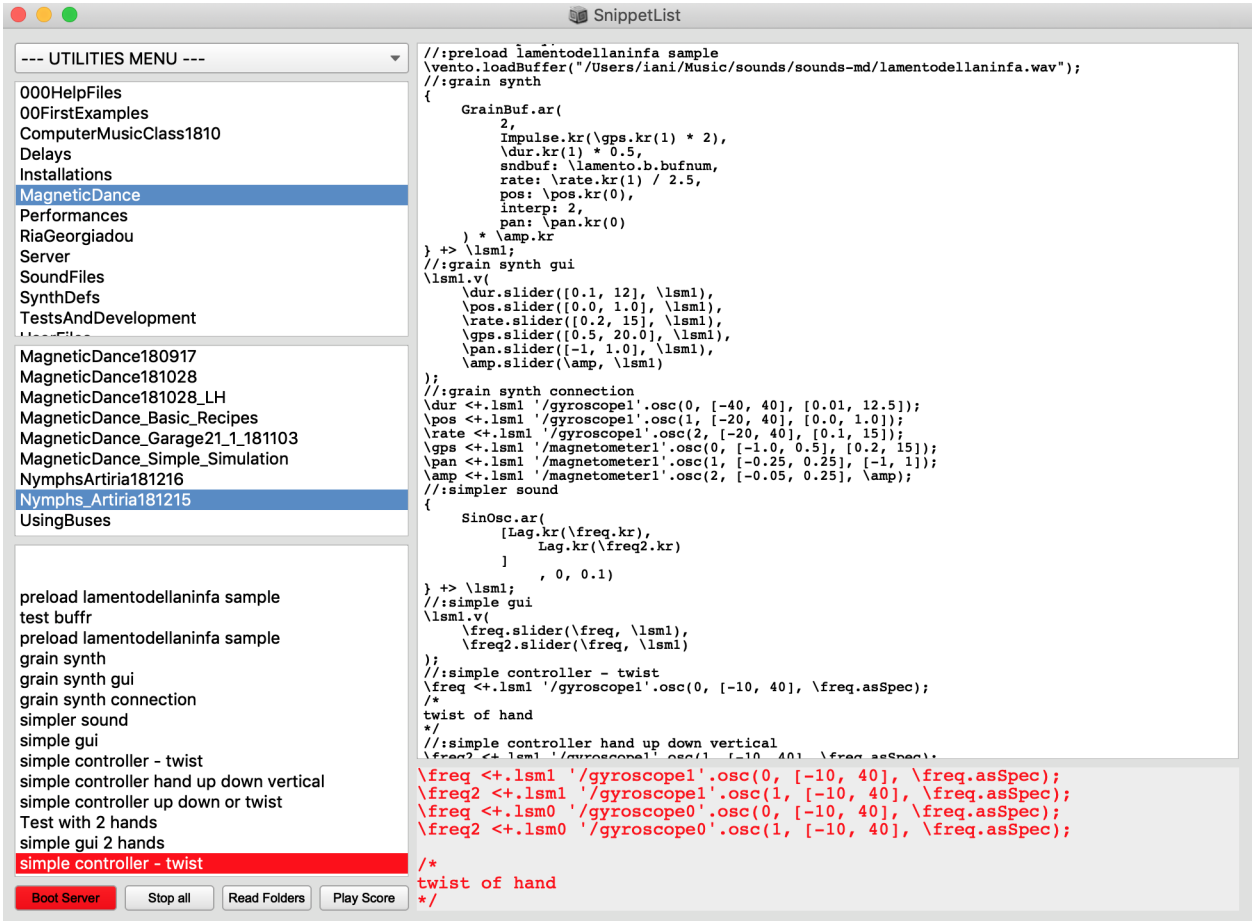

Figure 5: *Snippet List GUI.*

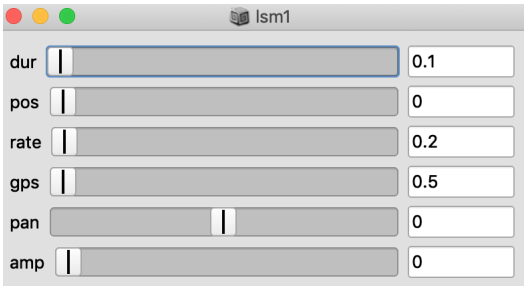

Figure 6: *Grain Control GUI.*

piece proved to be especially difficult as it is initially scored for 4 assistants in the electronic part, where each assistant is assigned control of the recording, playback and feedback levels of two tape recording channels with varying loop durations between sections, using two potentiometers. To execute this with a single performer on the computer, the slider actions as well as the loop duration changes had to be automated according to the indications in the score. Even under these circumstances, an ideal faithful performance was impossible, because each of the 6 levels demanded constant adjustment according to the actual level of the instrumental performer, and each transition had to be timed manually to prevent abrupt noticeable changes. Still, this proved to be a fruitful exercise in creating a user interface and coding the entire score, consisting of 6 different realization versions. It resulted in a compact coding scheme for durations of prescribed length (see 7 for the notation of the first version - Form-schema I, and 8 for its translation into GUI and automated performance). This notation mechanism can in the future be repurposed as a type of beat sequencing notation similar to this found in ixilang or TidalCycles (although the Cycle scheme of Tidal has other features which go beyond the scope of the present discussion).

4. CONCLUSIONS AND FUTURE WORK

Sc-hacks is a general purpose extension to SuperCollider, and the intense use of several binary operators may raise doubts about its legibility or the general validity of its design priorities. However, stress-testing sc-hacks through collaborations with dancers and instrumentalists has shown its strong potential to solve diverse and demanding problems under time pressure, and furthermore has provided indications of its scalability in terms of coding various features. This indicates that it is a suitable platform for further work, and it is hoped that it will serve as a tool for addressing questions of machine listening in live performance as well as other advanced topics.

```
    *formschemaI {
        ^StockhausenSoloFormschema(
            [  // numperiods, duration per period
                [11, 6],
                [8, 14.2],
                [7, 19],
                [6, 25.3],
                [9, 10.6],
                [10, 8]
            ], thisMethod.name.asString
        )
        .loadPeriodStates(
"
X__X__XXX__|XXXXXXX_|X__X___|X__X__|_X_X_X___|X_X__X_X__
_XX_XX__XXX|XXXXXXX_|XXX_XXX|_XX___|XXXXX_XX_|_X__X_X_X_
_XXXX__XXXX|_XXXXXXX|_X__XXX|_X__XX|__XX__XX_|_XXXXX__X_
__X__XX__XX|_XXXXXX_|_XX__XX|__X___|_XXXXXXX|__X____XX_
_XX_XX__XX_|_XXX_XXX|XXX_XX_|_XX_XX|X_XXX_XXX|_X_XX_X_XX
__XX__XX__X|_X_XXX_X|___X_XX|__XX__|_X___X__X|X_XX_X_XXX
"
        );
    }
```

Figure 7: *Code for Formschema I of Stockhausen Solo 19.*

Recording data received from sensors is a first priority in the project. A first prototype has been implemented using the built-in archival facilities of SuperCollider. A second implementation is currently under development, which will record data into multichannel audio signal buffers, and employ an extra channel to record the time interval between receipt of successive OSC messages. Based on this, and using the existing graphic visualization facilities of SuperCollider for audio signals, a functionality similar to the MuBu tools from IRCAM [3] is envisaged. In collaboration with PhD students working on Machine Learning, it is planned to use this for further research.
In parallel, work is being done to connect data sent over the internet in remote performances, and in developing a performance repertory with instrumental soloists interested in improvisation with live electronics. In both these cases, the most serious challenge consists in making the software stable and easy to use enough to be able to release it to non-specialist performers for work in real-world creative events without the need of specialized technical assistance to run it. This remains a major driving factor and design guideline in developing this software. At the same time it is expected that these requirements will help create best practice solutions that constitute the wider contribution of this project. In this sense, the present project is placed within the scope of efforts for developing contemporary languages of notation for performance practice that have lasting impact on the community and its aesthetics.

5. REFERENCES

[1] S. Wilson, D. Cottle, and N. Collins, Eds., *The SuperCollider Book*, MIT Press, 2011.

[2] A. Sorensen, "Impromptu: An interactive programming environment for composition and performance," *Proceedings of the Australasian Computer Music Conference*, 01 2005.

[3] A. Kapur, P. Cook, S. Salazar, and G. Wang, *Creating music with ChucK*, Manning, 2015.

[4] A. Sorensen, *Extempore: The design, implementation and application of a cyber-physical programming language*, Ph.D. thesis, The Australian National University, 2018.

[5] C. Roberts, M. Wright, and J. Kuchera-Morin, "Music programming in gibber," in *Proceedings of the 2015 International Computer Music Conference*, pp. 50–57. 01 2015.

[6] D. Stowell and A. McLean, "Live music-making: A rich open task requires a rich open interface," in *Music and Human-Computer Interaction*, pp. 139–152. Springer, 2013.

[7] S. Salazar, "Searching for gesture and embodiment in live coding," in *Proceedings of the International Conference on Live Coding*. 2017.

[8] G. Wang, G. Essl, J. Smith, S. Salazar, P. Cook, R. Hamilton, and R. Fiebrink, "Smule= sonic media: An intersection of the mobile, musical, and social," in *Proceedings of the International Computer Music Conference*, p. 16–21. 2009.

[9] S. Salazar and J. Armitage, *Re-engaging the Body and Gesture in Musical Live Coding*, 2018, [Online; accessed 20-December-2018], `https://embodiedlivecoding.github.io/nime2018-workshop/workshop-paper.html`.

[10] J. Armitage and A. McPherson, "The stenophone: live coding on a chorded keyboard with continuous control," in *Proceedings of the International Conference on Live Coding*. 2017.

[11] K. Sicchio, "Hacking choreography: Dance and live coding," *Computer Music Journal*, vol. 38, no. 1, pp. 31–39, 2014.

[12] T. Magnusson, "The ixi lang: A supercollider parasite for live coding," in *Proceedings of the International Computer Music Conference*, pp. 503–506. 2011.

[13] A. W. Shu-Cheng, "Supersampler: A new polyphonic concatenative sampler synthesizeer in supercollider for sound motive creating, live coding, and improvisation," in *Proceedings of the International Computer Music Conference*. 2017.

[14] G. Roma, "Colliding: a supercollider environment for synthesis-oriented live coding," in *Proceedings of the 2016 International Conference on Live Interfaces*. 2016.

[15] A. McLean and G. Wiggins, "Tidal – pattern language for the live coding of music," in *Proceedings of the 7th Sound and Music Computing Conference*. 2010.

[16] I. Zannos and M. Carle, "Metric interweaving in networked dance and music performance," in *Proceedings of the 15th Sound and Music Computing Conference*, pp. 524–529. 2018.

[3] `http://forumnet.ircam.fr/product/mubu-en/` (accessed 20-December-2018)

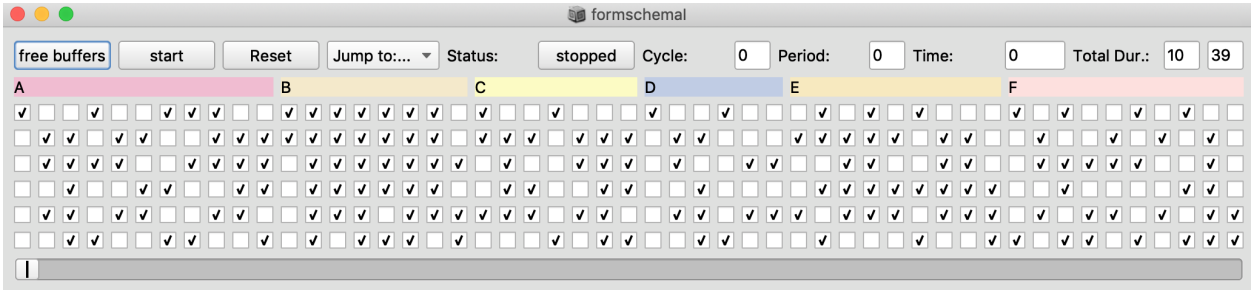

Figure 8: *GUI for Formschema I of Stockhausen Solo 19.*