Cognitive Dimensions of Modular Noise Improvisation

James Noble
Engineering & Computer Science
Victoria University of Wellington
kjx@ecs.vuw.ac.nz

Abstract

The availability, affordability, and portability of Eurorack format modular synthesizers has lead to an increase in their use in live, improvised performance. Decreasing prices and physical size, coupled with increasing reliability has meant modulars are finally leaving the studio and appearing on stage. While modular synthesizers are typically prepatched (configured at leisure) ahead of time, contemporary synthesists are adopting livepatching — wiring up a modular — as an integral part of their performance practice. This paper uses the cognitive dimensions framework to analyse the programmatic content of modular livepatching, in the context of the author's experience with modular synthesizers for performing improvised noise.

1. Introduction

Modular synthesizers are finally leaving the esoteric world of academic electronic music studios and moving into concert halls, rock venues, and dodgy experimental performance spaces (Paradiso, 2017; Auricchio & Borg, 2016). The key innovation has been the gradual adoption of the Eurorack format for modular synthesizers, developed by Doepfer in the late 1990s, which are physically smaller than classical modulars of the 1960s (Moog and Roland modules were 5U high and used large "1/4 inch" (6.35mm) plugs and sockets; Eurorack modules are 3U high and use 2.5mm plugs and sockets). Relying on electronics of the 1990s and beyond (rather than the 1960s) means Eurorack modulars are cheaper and lighter than traditional modulars (as well as smaller), greatly easing their practicality for musical and sound-art performance (Rossmy & Wiethoff, 2019).

This paper interprets modular synthesizer patching — particularly "livepatching" where modules are (re-)connected as part of a public noise-art performance[1] — as a live physically-embodied domain specific programming language (see Figure 1. Modular synthesizers are essentially domain specific analogue computers; configuring analogue computers by patching modules together and adjusting parameters is domain specific programming, distinct from "playing" the synthesizer by triggering notes from a keyboard. "Livepatching" — adding and changing connections between modules and adjusting the parameters of those modules — is the modular analogue of music performance by live programming — a.k.a. "livecoding" (McLean, Rohrhuber, & Collins, 2014; Hutchins, 2015).

Figure 1 A Quite Noise: Winter Livepatching

The next section briefly overviews modular synthesis and gives the context of my experience livepatching improvised modular noise. Section 3 then

[1] *A Quiet Noise: Winter*, rDc, Dunedin, 14 June 2019. Further illustrations and video available from `https://dunedinsound.com/gigs/a-quiet-noise-winter/` by fraser@dunedinsound.com under CC-BY-SA 4.0.

considers livepatching using the cognitive dimensions framework, section 4 discusses some related work, and section 5 concludes.

2. Livepatching Eurorack

For the last five years I've been livecoding, improvising, and livepatching as half of "Selective Yellow", an experimental improvisation duo of indeterminate orthography drawing on New Zealand's heritage of experimental music practice (Russell, 2012; McKinnon, 2011) that seeks to recreate (electronically) all the worst excesses of free jazz with all the enthusiasm of antisocial teenagers meeting their first MOS6851 (Wilson & Noble, 2014). Selective Yellow performances typically employ a number of different synthesizers or sound generators as well as modular synthesizers, ranging from digital toys (Kaosscilators, Buddhamachines) to semi-modular analogue and MIDI digital synthesizers, played with a variety of controllers (wind controllers, monome grids, knob boxes etc) — while eschewing a primary role for digital audio workstation software and computer-based virtual instruments. Since Selective Yellow rehearses and performs only a few times each year, we're probably only Grade 4 (Nilson, 2007).

Figure 1 shows the configuration of typical modular synthesizer — a smaller version of the modular I used in a recent Selective Yellow performance. The modular rack is in front of the performer, who is interacting with a separate control surface (Beatstep Pro sequencer). A couple of smaller standalone synthesizers are to either side of the main modular. There is a tangle of patch cables on the modular, some from the sequencer to the modular proper, the majority interconnecting modules in the rack.

Figure 2 shows a simple modular patch. On the left a MIDI module is providing a regular clock signal; this signal is being used to control the sequencer module next to it. The outputs from the sequencer (a gate signal and a pitch control voltage) are then used to control the Edges oscillator bank module. The "MIX" audio output of Edges is linked into the audio input of the Wasp filter; the cutoff frequency of the filter is modulated by two separate LFOs from the Quadruple LFO module. Finally the filter's output is patched into the "Outs" module, which can then be connected to a terrifyingly large amplifier.

Figure 2 Example modular patch. Screenshot of modularGrid.com. From left to right the modules are MIDI (as clock source); micro-sequencer; Edges oscillator bank; Wasp filter; Quadruple LFO; Outputs.

The dynamics of textual livecoding — performance programming to produce music — using relatively traditional textual programming languages and editors or development environments has been well examined (McLean et al., 2014; A. Blackwell, McLean, Noble, & Rohrhuber, 2014; Magnusson, 2014). One of the goals of Selective Yellow has been to transfer that aesthetic from the disembodied, virtual, digital world of programming languages to the embodied, tangible, analogue world of modular synthe-

sis. Our approach is similar to Hutchins (2015) in following classical livecoding practice (Nilson, 2007): as much as possible starting with an empty, unpatched synthesizer; performing the patching either where it can be seen directly by the audience (Figure 1 was taken from the audience less than two metres from the performers) or, in larger venues, indirectly projecting an image of the modular to the audience; and explicitly focusing on the programmatic aspects of patching — connecting modules via patch cords.

Modular livecoding is centred on two main activities "patching" and "tweaking" (Bjørn & Meyer, 2018). Patching is constructing circuits by connecting modules together, and tweaking (also "twiddling") involves adjusting the settings of the patched modules by e.g. turning the knobs or operating other controls on their faceplates, or adjusting settings on control surfaces. Neither patching nor tweaking are really direct analogues of playing a traditional musical instrument: tweaking comes closest but is often more tentative or exploratory.

Modular synthesizers can also include traditional performance-oriented controllers such as joysticks, light sensors, monome grids, or theremins, and via MIDI or OSC, can have access to essentially any contemporary digital performance controller. This gives rise to a third potential activity — that of an instrumental soloist where much of the expression of the generated sound is under the performer's immediate control. (Even without such controls, there is always the option of performing a solo by "playing" the pitch control knob of an oscillator and filter cutoff or mixer attenuation in real time. I find this soloing activity qualitatively different to adjusting module parameters to configure their place in a larger patch: tweaking an oscillator pitch control while gurning like a 70s guitar hero is direct and immediate, much more like playing a conventional musical instrument, while configuring a module by turning a knob that controls the amount of LFO modulation to be applied to the pitch is much more indirect and delayed, and much more like programming.

3. Cognitive Dimensions

The *Cognitive Dimensions* framework (Green, 1989; Green & Petre, 1996) analyses notations of any kind, including but by no means limited to data or code visualisations and programming languages. For example, the framework has been used to compare and contrast Ableton Live and the ChucK programming language (A. F. Blackwell & Collins, 2005) and to evaluate the design of a tangible programming language (A. F. Blackwell, 2003); earlier I used this framework to interrogate programming with the LittleBits SynthKit (Noble, 2014).

The Cognitive Dimensions framework is a collection various "dimensions" that can be used qualitatively to evaluate a design. The framework is flexible in that there is no canonical set of dimensions, rather existing dimensions can be adopted and new dimensions proposed to suit the task at hand. Following A. F. Blackwell (2003), in this section I evaluate livepatching under twelve commonly-used dimensions, writing the name of the dimension in **boldface** and the key words of the analysis in *italics*.

Medium The medium of expression is primarily the patch cables that link modules; the action of livepatching is ultimately the manipulation of patch cables. The medium of expression also involves the configurations, programs, or knob settings of individual modules. Module parameter knobs are also manipulated in livepatching — more time may be spent tweaking knobs to adjust fine details of a patch rather than the higher-level (and more visible) module patching itself.

Inasmuch as livepatchers (like other Eurorack synthesists) typically select and combine modules to configure their own unique modular synthesizer, this is also part of the medium of expression: the choice of modules, and their layout within rack cases. Unlike patching and knob twiddling, actually building modular synthesizers is not typically part of a livepatched modular performance.

Unlike some other tangible interfaces, patching locations are necessarily *constrained* to the various inputs and outputs on each module, and knob settings are likewise constrained to the options offered by each module. Both livepatching and twiddling expression is *transient*, as cables can be patched or modules twiddled at any time. The transience of the patching is important: as the name implies, it is the

livepatching, rather than the knob twiddling, that distinguishes a livepatched modular performance from a more typical "prepatched" modular performance — where modules parameters will be adjusted, and sounds triggered e.g. using sequences or other gestural controllers, but where the patching is primarily fixed before performance.

Both twiddling and patching are based on the *absolute position* of modules within a rack, and of inputs and outputs within a module.

Activities The key livepatching activity is *incremental construction* and then *modification* of patches, by connecting modules with patch cables, and also *modification* of module parameters by tweaking the parameter knobs. This is essentially *exploratory design* — how exploratory depends on both the courage and ignorance of the synthesist: courage to try something when they're not sure of the result, and ignorance to increase the likelihood of not being sure of the result.

While modular synthesists may use *transcription* to record patches and module settings, I have not transcribed settings as part of Selective Yellow performances.

Visibility To livepatch a modular synthesizer, the physical synthesizer and patch cables must be to hand: in principle, then, the patch is always *visible*. Unfortunately this principle is not realised in practice: where "spaghetti code" is a metaphor, "spaghetti patching" (or "rat's nests") are the rule rather than the exception (see figure 1). While a judicious use of different colours of patch cables and careful module placement within a rack can help mitigate this, it is in practice difficult to "read" a complex modular patch without tracing each individual patch cable — under stage lighting, tracing may mean physically following the path of a cable by touch rather than sight (a novel sense of tangible interface).

Most modules are designed so that the settings of their knobs and switches are visible, at least given sufficient lighting of the performance space (either ambient light, or e.g. small LEDs illuminating the modular). Reasonable visibility of the modular itself is required to successfully patch outputs to inputs. Some modules backlight input and output sockets, mostly to look cool, although this does assist livepatching on stage.

Modules may have other visual outputs — from individual LEDs and 7-segment displays right up to embedded OLED screens (see Figure 3) — to make at least some of their internal state visible. Even individual LEDs can be surprisingly useful: particularly for low frequency oscillators (LFOs) which are primarily used to control other modules. Often the amount of LFO modulation taken into a module will governed by a separate knob: being able to see where the LFO is in its cycle (maximum, minimum, or somewhere in between) can help setting the right levels to create a particular audio effect.

Figure 3
Oscilloscope.

Diffuseness Any given modular will have a fixed physical size. Most modular cases are designed to fit into standard 19" wide racks, each module occupying 3U of rack space. Individual modules are *moderately sized* ranging from about 1cm wide to as much as 30cm or more. Thus modular patches are relatively *compact* because they have to fit within the available hardware. Even though modulars are larger than most laptops, they are physically smaller than e.g. a pair of large LCD panels used by a professional programmer — and of course, while a program in most languages can be much larger than the available screen real estate, that is not possible with actual hardware. Conversely, the entirety of a modular patch is immediately accessible to the synthesist, whereas editing most programs requires first mapping some parts of the program into the available screen space.

The density of interaction items (input and output sockets, knobs, LEDs etc) varies somewhat across modules, and is limited by the physical size of 2.5mm jacks, LEDs, and knobs. The highest practical density is around one interaction item per couple of square centimetres, with some larger modules being significantly more diffuse — in Figure 1, the left-most "MIDI" module (19 controls in 40cm^2) is significantly more dense than the central "Edges" module (26 controls in 130cm^2). Individual interaction items also vary in size from jack sockets and thin knobs e.g. up to MIDI sockets, LED screens, or knobs that are bigger than usual — either to offer finer control than a smaller knob, for aesthetic reasons, or because twiddling a big knob is somehow more fun than twiddling a smaller one (see Figure 4).

Figure 4
Verbtronic Module.

Viscosity Patch cables can be unpatched and repatched between modules, resulting in relatively *low viscosity*. Adjusting module parameters by adjusting knobs is pretty much as direct as possible — the limit being less in actually making changes (viscosity) than in determining which change to make (visibility, above).

Many kinds of changes can involve more than one patch cable, i.e. more than two modules: these kinds of changes encounter rather higher viscosity. Consider an LFO patched to one aspect of a sound (filter cutoff, say) that the performer would like to connect to another aspect of the same sound (amplitude) so that they will be modulated together. This requires the cable linking the LFO and filter to be unpatched, that cable to be repatched into a "multiple" module (which takes a single input and connects it to multiple outputs), then separate cables run from the multiple outputs to the filter and VCA modules. Another common case is inserting one module in the middle of an existing signal path: a drum module could be connected directly to a mixer channel on the way to an output, but the performer would like to process the drum through a low-pass filter modulated by an LFO. This single logical interaction requires three distinct actions (unplugging the output, plugging that into the filter, and then plugging the filter output back into the mixer channel), four actions if fetching another patch cable is counted! Here is one case where tangibility of interaction looses out to the greater flexibility e.g. offered by virtual modulars such as the Nord (Clavia DMI AB, 1999) which supports multiple connections from outputs and interlinking modules as single interactions.

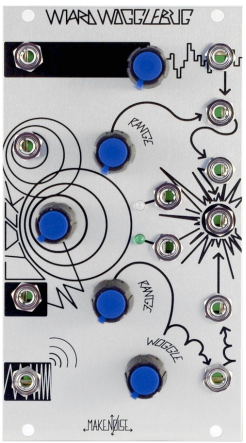

Figure 5
Wogglebug Module.

Secondary Notation Eurorack modules come from a wide range of manufactures other than Doepfer, so the physical and graphical design varies immensely. Even Doepfer, who began with a very utilitarian grey-on-grey design, have more recently branched out into "special editions" and "vintage series" with different panel and knob colours. The design of the modules themselves is decoupled from their function and serves as a secondary notation to distinguish modules from one another. Depending on their design, modules may be *labelled* with their name, and the functions of the various inputs, outputs, knobs, or other interaction devices — either explicitly with text or simple graphical icons, or implicitly by incorporating controls and sockets as integral parts of a more complex graphical design (see Figure 5). Knob size also serves as secondary notation, making it easier to distinguish between otherwise similar controls. Eurorack modules often follow implicit secondary notation conventions. A very common arrangement is that a parameter of a module is governed by three controls: a knob to set the base value of that parameter (b), a CV input which modulates the value of the parameter (c), and a control knob that attenuates the modulation (m) — the overall value being $b + mc$. The convention here is that the base knob (b) will be larger than than modulation attenuator (m); that ideally all three controls will be placed close to each other (gestalt), or at least the modulation CV input and

attenuator will be placed close together (see the large "OUTPUT MIX" knob, and the smaller "MIX CV IN" knob and socket on the Verbtronic in Figure 4). Where that is not done, graphical elements will link the three controls together (usually from the CV socket *c* to the attenuator *a* then to the base control *b*). Graphically, the base control is a metonomy for the underlying function it performs.

Finally, it is in theory possible to use qualia of the patch cables themselves as secondary notation — different brands of patch cables are different colours, with different finishes (plastic vs fabric). In practice I make little use of this — perhaps because the main supply of patch cables I have use colour coding for length, but more likely because while carefully selecting a cable may be possible in a studio setting, in a livepatching live performance it is difficult to give attention to such details.

The one exception is for initial cabling from the external Beatstep Pro sequencer and control surface: here the patch outputs are on the back of the device, that is, facing away from the performer. I therefore take care to pre-patch cables into the outputs before a performance starts and choose different cable designs for different functions (melodic sequencers vs drum triggers). Even so, more than once I have ended up peering into the back panel of the Beatstep trying to understand why a particular CV signal isn't behaving the way I expected[2].

Hidden Dependencies Because every intermodule connection is *explicit*, embodied by a patch cable, there are no structural (or syntactic) hidden dependencies between modules. This true at least in the modular configuration I most commonly use: some particular sets of modules (often the same brand) may support extra connection busses that are not part of the Eurorack specification — cables connecting the rear sides of modules, resulting in dependencies not visible on their faceplates.

Some modules have hidden dependences within their design: so called "normalled" connections that take a signal from an input or output socket in a module into a second input when there is no patch cable in the socket for the second input. For example, the "Edges" module (Figure 2) contains four separate oscillators, each with a separate Gate and Pitch (V/Oct) input socket. The gate and pitch inputs are normalled, so that a single pitch and gate input can control all four oscillators in parallel: additional patching can then take control of each oscillator individually.

The most critical hidden dependencies in modular patching are *semantic*, rather than syntactic. While patch cables establish explicit connections between modules, the interactions between those modules can be much more subtle that a simple connection might imply. A common experience patching modulars (and indeed with much synthesizer programming) is that the patch produces no sound. This can be for as simple a reason as no trigger patched into a gate input, so that oscillators are never switched on, or rather more complex, e.g. if a lowpass filter's cutoff frequency is below the audio frequency of the oscillator patched into its input, then all that audio will be filtered out. These semantic dependencies are not just hidden, but also silent.

Role Expressiveness Livepatching can be conceived within two distinct roles. First, concretely (or extensionally, or with the performer's perspective centred), livepatching is primarily what its name suggests: live patching of synthesizers. The domain model is modular synthesis itself: actions within that domain model include "adding in another oscillator" or "adjusting a filter cutoff parameter" or "modulating oscillator pitch by a random stream". In this role, the *notation models the domain directly* — performers directly interact with the modular synthesizer qua modular synthesizer.

Second, abstractly (or intensionally, or centering the listener) livepatching is manipulating *sound objects* rather than synthesizer modules. The domain is sound, rather than hardware, and activities include "make the sound more plaintive" or "emphasize the bassline" or "descend into a howling vortex of

[2]There's an example at about 3min of the video of the *Quiet Noise: Winter* performance, where the pitch of the sequenced oscillator bank didn't change: it turned out the patch cable going into the oscillator pitch input was plugged into the wrong Beatstep output).

feedback". In general this role is *not supported* — rather performers must reconceptualise activities in this domain back into the modular domain. Sometimes this mapping can be direct — "emphasize the bassline" may be as simple as turning up the bass in the output mixer; while other times the mapping can be very indirect — "make the sound more plaintive" could require many different manipulations of module parameters, and unpatching and repatching of many modules.

In Selective Yellow, I have found myself *switching* between these roles. Part of this is due to the complexity of constructing the mapping between sound object and synthesizer patching required for the second role; part of this is often choosing to focus on the first role. As with other livecoding practices, demonstrating the underlying construction, emphasizing the "modular-ness" of the sound production, is a key part of the performance: making expansive gestures while patching in a new making module aims to draw attention to the predominance of the first role.

Premature Commitment Repatching a modular synthesizer is essentially the same as patching from scratch, and the generally low viscosity ensures that performers are *not prematurely committed* to particular structures or module settings. Patches can be built (or rebuilt) in any order — subject again to the viscosity issues discussed above: inserting a third module into a signal flow between two modules requires first disconnecting the two modules and then repatching. The physical embodiment of patch cables, and the low cost of repatching, support a *fluid* experience.

The fixed hardware resources of a modular may seem to give rise to premature commitment: if you are using a module for one thing (say an LFO modulating oscillator pitch) you cannot use that same module for something else (say modulating a delay line's feedback). My typical Selective Yellow modular configuration has eight LFOs: once they are all used, there aren't any more. This is a problem of *commitment* rather than *prematurity* — an LFO can be unpatched from its current task, and then repatched to perform some other function: this can be one in any order at any time, with only local changes.

Progressive Evaluation Modular patching is by its nature incremental. Sound will be produced so long some audio-rate signal eventually flows to an output module connected to a suitable PA system. Changes to module parameters are typically *evaluated immediately* and become immediately audible — again, provided a suitable signal path from that module to the output.

Accurately patching modules does take time however, so there can be the appearance of a shorter or longer *delay in evaluation*, depending on the performer's virtuosity in the physical actions of patching. In the *Quiet Noise Winter* performance, for example, the drums are introduced gradually, as individual percussion modules are patched in. Explicit delay modules also induce *delay* (or repetition) in evaluation — presumably a delay intended by the performer (see Figure 6).

Figure 6
Chronoblob.

Provisionality Cables can be patched and parameter knobs adjusted without being connected to an output, *supporting provisional arrangements* but of course without audible feedback (no progressive evaluation). As mentioned above, for practical reasons I tend to pre-patch connections from sequencer control surfaces simply because making such connections quickly and correctly in the middle of a performance is impractical.

As with many livecoding disciplines I do not use a cue mix to monitor provisional configurations. While separate headphone output modules, or integrated mixer modules with cue busses are available, I choose to work so that "the performer hears only what the audience hears" (Nilson, 2007). If I need to test a particular audio signal, I typically patch that signal into a mixer channel going to the main outputs: the audience can thus hear additional oscillators being brought into the mix, being

tuned manually to match the existing sound material, their timbre being adjusted. This way of working is much more feasible for performances in the noise subgenre rather than algorave bangers: in a noise performance an illtuned oscillator can be a welcome feature of the performance, whereas it will at best be distracting (and at worst incompetent) on the dancefloor.

Figure 7 Mult.

Abstraction The tangibility of hardware modular synthesizers is fundamentally *abstraction hating*: the kinds of abstraction mechanisms found in programming languages (e.g. a sub-circuit, packaging that behind an interface which encapsulates the implementation, exporting as sub-set of the input and output connections and parameter controls) cannot be encompassed within a fixed hardware system. There is also little benefit: all the parameters and connections of the implementing modules are readily to hand. It is not possible to replicate multiple versions of an abstract module in hardware the way a procedure can be invoked multiple times, or a Rack in Ableton Live instantiated in multiple channels.

Modular livepatching does support some abstraction techniques, although they are obviously more tangible ("concrete abstractions") relying on the use of standard modules. Two kinds of modules are useful here. First "multiples" (aka "mults" or splitters, a single input socket connected to several output sockets, see Figure 7) can allow a single control voltage (or audio source) to be used in several difference places in the synthesizer. A melodic pattern from a sequencer could thus control several different oscillators playing (somewhat) in tune, promoting the sequence into an abstraction in the sense that altering the sequence can alter the performance of a number of different modules. Second, submixers can act in a complementary way, combining a number of audio sources (say all the drum modules) or even a collection of control voltages into a single output. That output can then be processed further "downstream" — perhaps running a drum submix into a single channel of the primary output mixer; or into a flanger or a filter. The downstream controls or processes will now act over the combination of all the audio signals: the knob controlling the main mixer channel that is taking the drum submix now offers a control over all the drums as a single abstractions.

Finally, Eurorack systems (and thus livepatching) can incorporate higher-level abstractions, at the level of rack configurations (module choice) rather than individual patches. Traditional analogue modules provide one function (a single oscillator, filter, or envelope, built from discrete components) however many recent Eurorack modules combine more than one function. A drum module could combine an oscillator, filter, and envelope, yet offer far fewer control parameters, knobs, and input or output sockets than a similar circuit built from individual modules (Figure 8). The reduction in degrees of freedom ("expressibility") increases ease of use ("musicality"): it is much easier to produce a drum sound from a drum module than it is to patch and configure a whole collection of other modules to produce the same sound, not to mention cheaper. The smaller size of contemporary electronic componentry means that integrated modules can be physically smaller collections of more basic modules, which is important when the synthesizers will be carried into performance venues, rather that being bolted to walls of specialised electronic music studios.

Figure 8 Drum.

Many contemporary modules digitise incoming audio and control voltages, feed them into a realtime audio algorithm, and then render the output back to analogue signals — evolving towards embedded musical computers rather than single function analogue circuits (Scott, 2016). Interface modules can be used to route audio signals generated within the rack to external signal processors or guitar pedals and then route the resulting audio back into the modular (the hardware equivalent of a foreign function interface). Modules like the AppiOsc (Lawson, Smith, & Appio, 2016) are the logical end-point of this evolution, bi-directionally integrating a modular with a textual live coding environment.

4. Discussion and Related Work

In "Live Coding For Free", McLean (2008) describes how "We can think of coding live in the sense of working on electricity live, re-routing the flows of control around a program with the real danger that a faulty loop will get activated, causing the program to crash in sparks of logic". Eurorack livepatching works with live electricity — control voltages and audio signals — and loops or misconnections can cause electrical sparks and shorts (although given the low voltages involved, without any serious consequences). McLean also gives three criteria for livecoding. First, *Rules must be explicit* — "written down and modified". In modular livepatching rules are not written, rather they are embodied in arrangements of patch cables and the settings of sequencers or contents of delay lines: modifying those rules is precisely the point of livepatching. Second, *Higher order functions must be defined and manipulated*: one control signal modulating signal is a higher-order composition of those functions. Third, *An audience is not required.* While I would agree with this philosophically, I have found there is a large difference between rehearsing in private, and performing in public: especially performing to a paying audience (most of who are waiting for the death metal band on next) rather that the privilege of presenting an academic demonstration or educational workshop.

Hutchins (2015) has also investigated livepatching in the wider context of live programming — this paper contributes an analysis based on cognitive dimensions to that investigation. The cognitive dimensions analysis here is similar to an earlier study of livepatching the littleBits SynthKit (Noble, 2014). The key difference is that the Synth Kit is "patched" by physically attaching modules to each other: cabling modules at fixed positions in a rack has quite a different feel. Where manipulating littleBits seemed more object-oriented, my experience of modular patching is more functional, threading streams of values (time-varying control voltages) into functions that produce further value streams. This is in spite of the fact that Eurorack modules often have more internal state than Synth Kit components.

The tension between real and virtual is explicit in comparison with tangible live music programming language systems such as the reacTable (Jordà, Geiger, Alonso, & Kaltenbrunner, 2007), which uses physical objects on a multi-touch table interface to produce music. Animations in the touch table make the physical objects appear live, giving feedback along with the generated sound. In contrast, modular patches are in fact live, tangible, and generate sound directly. Moodler (Piponi, 2015) goes one step further and provides a "Mock Modular" with patch sockets and knobs mounted on a whiteboard so "modules" can be sketched, in order to provide tangible control to a virtual modular in Haskell. Mosaic (Mazza & de Pisón, 2019) moves in the other direction, providing a digital "virtual modular" designed to support livepatching on a laptop computer.

There are still many dimensions of modular livepatching (or indeed modular synthesizer performance) that we have not yet explored. Selective Yellow is a duo, but generally each performer acts independently (other than listening to what the other is doing). We hope to explore the possibilities of multiple performers using a single modular synthesizer, or equivalently, several modular and semi-modular synthesizers being patched together. This seems similar to multiple textual livecoders using computer networks to exchange code fragments during performances, although if anything with more direct interaction. Patching modulars together means one performer can directly affect the sound produced by another, as in Cage's *Imaginary Landscape No.4* where one performer controls a radio's tuning, and a second independently controls volume and tone, and where the result is silence as often as sound.

5. Conclusion

In this paper I have reflected on my performance practice livepatching modular synthesizers, through the lens of the cognitive dimensions framework. Treating modular synthesizer as an embodied domain specific programming language has helped identify points that make this practice unique, or at least, positioned at an intersection of two better known practices: livecoding and instrumental improvisation. The tangibility of the modular is the key linkage point, allowing rapid movement between indirect, programmatic patching and configuring (tweaking); and occasional excursions into more direct soloing. The complexity of modular synthesizers, the potential interactions, feedback loops, and higher-order

constructions complements the tangibility: LFOs or random function generators can modulate volume or timbre or any other parameter (including, of course, other modulation paths) and the resulting patches' complexity is limited only by the available hardware. In a Selective Yellow improvisation, this often results in silence (as Cage would have it) but equally often at least some kind of noise.

6. References

Auricchio, N., & Borg, P. (2016). New modular synthesizers and performance practice. In *SAE Melbourne Symposium.*

Bjørn, K., & Meyer, C. (2018). *Patch and tweak.* BJOOKS.

Blackwell, A., McLean, A., Noble, J., & Rohrhuber, J. (2014). Collaboration and learning through live coding (Dagstuhl Seminar 13382). *Dagstuhl Reports*, *3*(9), 130–168.

Blackwell, A. F. (2003). Cognitive dimensions of tangible programming languages. In *First Joint Conference of EASE and PPIG* (pp. 391–405).

Blackwell, A. F., & Collins, N. (2005). The programming language as a musical instrument. In *Psychology of Programming Interest Group (PPIG)* (pp. 120–130).

Green, T. (1989). Cognitive dimensions of notations. In A. Sutcliffe & L. Macaulay (Eds.), *People and Computers V* (pp. 443–460). Cambridge.

Green, T., & Petre, M. (1996). Usability analysis of visual programming environments: A 'Cognitive Dimensions' framework. *Journal of Visual Languages and Computing*, *7*, 131–174.

Hutchins, C. (2015). Live patch / live code. In *International Conference on Live Coding (ICLC).*

Jordà, S., Geiger, G., Alonso, M., & Kaltenbrunner, M. (2007). The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. In *Tangible and Embedded Interaction (TEI).*

Lawson, S., Smith, R. R., & Appio, F. (2016). Closing the circuit: Live coding the modular synthesizer. In *International Conference on Live Coding (ICLC).*

Magnusson, T. (2014). Herding cats: Observing live coding in the wild. *Computer Music Journal*, *38*(1), 8–16.

Mazza, E., & de Pisón, M. J. M. (2019). Mosaic, an openFrameworks based visual patching creative-coding platform. In *International Conference on Live Coding (ICLC).*

McKinnon, D. (2011). Centripetal, centrifugal: electroacoustic music. In G. Keam & T. Mitchell (Eds.), *HOME, LAND and SEA: Situating music in Aotearoa New Zealand* (p. 234-244). Pearson.

McLean, A. (2008). Live coding for free. In A. Mansoux & M. de Valk (Eds.), *FLOSS + Art.* GOTO10.

McLean, A., Rohrhuber, J., & Collins, N. (2014). Special issue on live coding. *Computer Music Journal*, *38*(1).

Nilson, C. (2007). Live coding practice. In *New Interfaces for Musical Expression (NIME).*

Noble, J. (2014). Livecoding the SynthKit: Little Bits as an embodied programming language. In *Software Visualization (VISSOFT).*

Nord Modular Manual (V3.0 ed.) [Computer software manual]. (1999). Sweden.

Paradiso, J. A. (2017). The modular explosion - deja vu or something new? In *Voltage Connect.*

Piponi, D. (2015). Moodler: A digital modular synthesiser with an analogue user interface. In *Functional Art, Music, Modelling and Design (FARM).*

Rossmy, B., & Wiethoff, A. (2019). The modular backward evolution – why to use outdated technologies. In *New Interfaces for Musical Expression (NIME).*

Russell, B. (Ed.). (2012). *Erewhon calling: Experimental Sound in New Zealand.* The Audio Foundation and CMR.

Scott, R. (2016). Back to the future: On misunderstandiung modular synthesizers. *eContact!*(17.4).

Wilson, C., & Noble, J. (2014). *Selective Yellow.* https://selectiveyellow.bandcamp.com.

Acknowledgments

Thanks to Chris Wilson, the better half of Selective Yellow, without whom which.