2014 Second IEEE Working Conference on Software Visualization

Livecoding the SynthKit: littleBits as an embodied programming language

James Noble
School of Engineering and Computer Science
Victoria University of Wellington
New Zealand
Email: {kjx}@ecs.vuw.ac.nz

***Abstract*—littleBits (littleBits.cc) is an open-source hardware library of pre-assembled analogue components that can be easily assembled into circuits, disassembled, reassembled, and re-used. In this paper, we consider littleBits — and the littleBits SynthKit in particular — as a physically-embodied domain specific programming language. We describe the littleBits system, explain how littleBits "programs" are constructed as configurations of physical modules in the real world, and describe how they are typically used to control physical artefacts or constructions. We then argue that littleBits constructions essentially "visualise themselves". We describe how littleBits' liveness, embodiment, and plasticity assists both learning and debugging, and then evaluate littleBits configurations according to the cognitive dimensions of notations.**

I. Introduction

littleBits (littleBits.cc) is an open-source hardware library of pre-assembled analogue components that can be easily assembled into circuits, disassembled, reassembled, and re-used [1]. Designed to inspire and teach basic electrics and electronics to school-aged children (and adults without a technical background) littlebits modules clip directly onto each other. littleBits users can build a wide range circuits and devices with *"no programming, no wiring, no soldering"* [2] — even extending to a "Cloud Module" offering a connection to the internet, under the slogan "*yup. no programming here either* [sic]" [3].

The littleBits system comes packaged as a number of kits: "Base", "Premium" and "Deluxe" kits with 10, 14, and 18 modules respectively; and a series of booster kits containing lights, triggers, touch sensors, and wireless transceivers. littleBits have recently introduced special purpose kits in conjunction with third party organisations, notably a "Space Kit" designed in conjunction with NASA, and a "Synth Kit" designed in conjunction with KORG that contains the key components of an analogue modular music synthesizer.

In spite of littleBits' marketing slogans, in this paper, we analyse littleBits —- and the littleBits Synth Kit in particular[4] — as a live physically-embodied domain specific programming language. If building littleBits circuits is programming, then performing music with the littleBits Synth Kit (configuring modules to construct an analogue music synthesizer, and then producing sounds with that synthesizer) can be considered as a music performance by live programming — a.k.a. "livecoding" [5] — especially as the circuit construction typically occurs simultaneously with sound production.

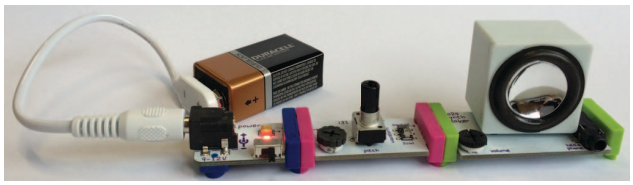

Fig. 1. A simple littleBits Synth Kit circuit. From left to right, the three modules are a power source, an oscillator, and a speaker

The next section introduces the littleBits system generally, and the littlebits Synth Kit (the focus of our practice) in particular. Section III then describes our experience live programming with littleBits, and section IV evaluates littleBits using the cognitive dimensions framework. Section V discusses some related work and section VI concludes.

II. The littleBits SynthKit

Figure 1 shows the one of the simplest circuits in the littleBits synth kit — indeed, the simplest littleBits circuit that can actually make any sound. This circuit is composed of three simple modules — a power module on the left, an oscillator module in the centre, and a speaker module on the right. The power module accepts power from a nine volt battery (or a 9V guitar pedal mains adapter) and provides that power to "downstream" modules — as seen in the figure, littleBits circuits flow in a particular direction, and all modules are oriented so that this flow is left to right.

Figure 2 gives an overview of all the modules in the Synth Kit. A 2 input, 1 output mixer allows two audio streams to be mixed, while a 1 input to 2 output splitter creates a fork in the signal flow. Each kit has two oscillators that produce pitches, and random (white noise) generator. Sound from the oscillators (or random noise) can be processed by an analogue delay line, modulated by an amplitude envelope or audio filter, and keyboard and sequencer modules can be used to control oscillator pitch or filter cutoff. Finally a power module supplies power to a circuit, and a speaker module plays the resulting sound (or provides a feed to a PA system or headphones). A module's connectors are coloured to represent

978-0-7695-5305-4/14 $31.00 © 2014 IEEE
DOI 10.1109/VISSOFT.2014.16

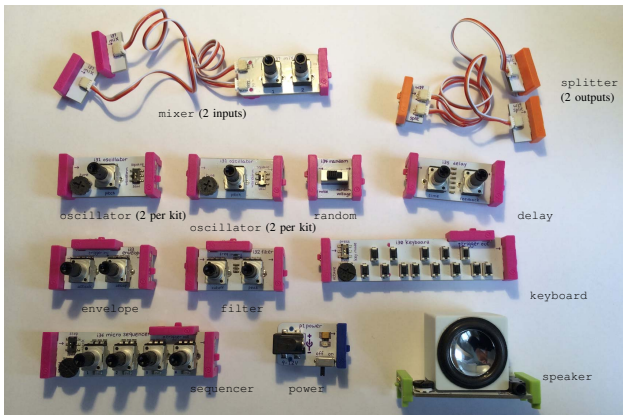

Fig. 2. All the Synth Kit modules

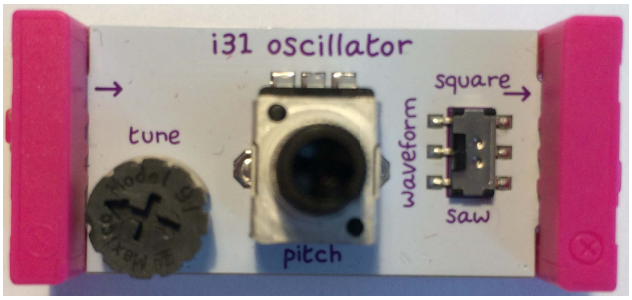

Fig. 3. A single littleBits oscillator module (larger than actual size).

the module type: power modules, blue; input modules, pink; output modules, green; and wiring modules, yellow.

Figure 3 shows the key sound producing module in the Synth Kit, the oscillator. An oscillator must be connected (directly or indirectly) to a power source on its left (input) connector, and produces an audio signal on its right (output) connector. The type of the audio signal is set by the small "waveform" switch to either a square wave (only odd integer harmonics) or a sawtooth wave (all integer harmonics). The pitch of the output tone is set by the relatively large shaft (coarse "pitch") and the relatively small thumbwheel (fine "tine"). If a varying signal is passed in to the oscillator, that signal will modulate the oscillator's pitch.

Note that almost every module (including the speaker, see figure 2) has at least one input and one output connector: the only exception is the power module, which has an output but no input. Some modules have more than two connectors: the mixer and splitter have two inputs and two outputs respective; the filter and envelope modules have an additional input that modulates the filter frequency (see figure 4) or triggers the envelope; the keyboard and sequencer modules have an additional output that signals when a new note is started (and is often wired e.g. to the filter's modulating input). An important point here is that the design of the modules is very much centred around the audio signal flow, that goes left to right

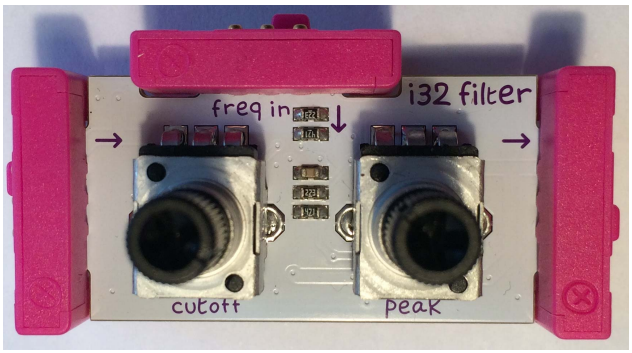

Fig. 4. A single littleBits filter module (larger than actual size).

across almost module, and modules will not function without a primary input to supply power (except the power module of course). This, in the filter module (figure 4) unfiltered input audio arrives at the left, is modulated by the filter (based the filter cutoff frequency and peak width (Q), as set by the two shafts), and departs on the right, while the additional cutoff modulation control signal comes in on the secondary connector at the top of the module.

III. Livecoding with littleBits

If building and configuring circuits with littleBits can be considered as a form of embodied, tangible, programming, then performing music "live" with littleBits can be considered as a form of livecoding — performance programming to produce music [5], [6], [7] — or in this case both *building* and *playing* synthesizers as a live performance. In this section we describe our practice livecoding littleBits, and compare and contrast with typically textual livecoding (inasmuch as typical livecoding practice can be supposed to exist).

This section in particular draws on the first author's experience livecoding/performing littleBits with "Selective Yellow", an experimental improvisation duo of indeterminate orthography drawing on New Zealand's heritage of experimental music practice [8], [9] Selective Yellow performances typically employ a number of different synthesizers or sound generators as well as littleBits, 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. Selective Yellow is still a relatively young project, probably only Grade 2 as evaluated by Nilson [10].

Livecoding with littleBits involves two main activities that are tightly interleaved in a performance, first building the circuits by clipping modules together, and second "playing" the resulting synthesizer by turning the shafts, thumbwheels, switches, the "keys" on the keyboard module to actually generate sound. Generally a performance — or rather the portion of the performance improvised upon littleBits — starts with the smallest possible sound-generating circuit, typically the

single unmodulated oscillator in figure 1. Once the littleBits are assembled (and the speaker module's output patched into the sound system) we can manipulate the oscillator's pitch and output waveform. Depending on the context of the improvisation, the possibilities of such a straightforward sound generator will be more or less quickly exhausted, at which point the performer will disassemble the circuit, insert one or more additional modules (a second oscillator, a filter, or perhaps a keyboard or sequencer module) and then continue playing the resulting circuit. In this overall pattern, littleBits livecoding is similar to some textual livecoding, where performers typically start with a single texture and then build a more complex improvisation over time.

While the circuit building and playing are conceptually separate activities, an advantage of the physical nature (and careful design) of the littleBits components is that the two activities can be very tightly interleaved. Indeed, with more complex circuits (or more than one Synth Kit) its is quite possible to keep part of a circuit playing and producing sound (such as a sequencer driving an oscillator) while building/editing another branch of the same circuit — adding in a second oscillator controlled by the keyboard module with an independent output route, perhaps, or adding in a modulation path to a filter that is already making sound in the main circuit. Again, this overall dynamic is also found in some textual livecoding performances (see e.g. the SuperCollider jitlib [11]). Of course, because of the underlying simplicity of the analogue synthesizer embodied within the littleBits modules, the sounds produced by littleBits Synth Kit are much less complex than the sounds that can be produced by a general-purpose laptop running a range of digital synthesis or sampling (virtual) instruments, although, being purely analogue, they have a tone all of their own.

In the same way that programmatic live coders generally display the code on their laptop screens to the audience of the performance [11], Selective Yellow projects an image of the desk or floor space where the little bits circuits are being assembled. The projected image not only seeks to dispel "dangerous obscurantism" [12] but also to illustrate how the source is being generated - especially as some modules include LEDs as feedback to the performer. The sequencer module, for example, lights an LED to indicate the current sequencer step, and other littleBits modules can also be used to provide more visual feedback on circuits where that is necessary.

This projected display seems particularly useful for audiences when the performer is "debugging" their circuit (live). Here again the physicality of the littleBits modules comes to the fore, so there is something for the audience to see: the easiest way to debug a littleBits circuit is just to pull it apart, and insert a speaker module after each module in the circuit in turn, listening to the sound (if any) being output by each module. Often this lets the performer to understand (and the audience to notice) that there is no sound output from a littleBits circuit, allowing the performer to either readjust the module parameters, or/and re-assemble the circuit in a different design, if not producing the desired sound, at least producing something.

IV. Cognitive Dimensions

In this section we use the *Cognitive Dimensions of Notations* [13], [14] framework to conduct a qualitative evaluation of the LittleBits SynthKit as a coding notation. The Cognitive Dimensions framework analyses of notations of any kind, including but by no means limited to data or code visualisations and programming languages. Blackwell and Collins [15] use the cognitive dimensions of notations framework to compare and contrast Ableton Live and the ChucK programming language; Blackwell [16] has also used the cognitive dimensions framework to evaluate the design of a tangible programming language.

As its name implies, the Cognitive Dimensions framework is a collection "dimensions" that are used qualitatively to evaluate a design. In this section, following Blackwell [16] we evaluate littleBits under twelve commonly-used dimensions, writing the name of the dimension in **boldface** and the key words of the analysis in *italics*.

1) Medium: The medium of expression is primarily the littleBits circuit modules themselves. This expression is *transient* as circuit modules can be moved at any time. Unlike some other tangible interfaces, positioning is *constrained* — modules must be attached to other modules — and based on *relative position* rather than absolute location.

2) Activities: The key activity of live coding with littleBits is *exploratory design*: building circuit configurations and then experimenting with modules' settings in that configuration. Thus *modification* and *incremental construction* of circuits is very common. As part of Selective Yellow performances, we do not use *transcription* of example circuits e.g. from documentation, although presumably other users of littleBits may transcribe circuits from documentation or e.g. users' web forums.

3) Visibility: The large-scale structure of a littleBits circuit is physically present and thus always *visible*, however, the settings of individual modules (e.g. the position of the shafts controlling a filter's cutoff frequency and Q, see fig. 4) are generally *invisible*.

4) Diffuseness: littleBits modules are generally small and *compact*, and so circuits generally fit within the space of a single A4 page. Modules' relatively *small size* makes manipulating module settings difficult — especially those set with thumbwheels or grub screws rather than shafts.

5) Viscosity: It is easy to change circuit configurations by detaching and reattaching modules, resulting in *low viscosity*.

6) Secondary Notation: Each module is *labelled* with its name and the functions of its control affordances, although in a live performance setting, we have learned to recognize the SynthKit modules (and module's affordances) by their shape. As fig. 2 shows, module connectors provide syntax colouring. Modules' spatial positions could be used as secondary notation, but modules are most often coupled directly to the other modules with which they communicate. Splitters (see fig. 2)

or connecting wires (included in other littleBits kits, but not the synth kit) could be used to decouple modules physical and logical positions, but we have generally not done this in our practice.

7) Hidden Dependencies: Because every intermodule connection is tangibly present, and because entire circuits tend to be compact, to a first approximation, structural intermodule dependencies are *explicit* rather than hidden. Dependencies between the settings of modules' control parameters are *hidden* however — for example, if a filter's cutoff frequency is below its preceding oscillator's pitch, then no sound will be heard as the oscillator's output cannot pass the filter.

8) Role Expressiveness: Inasmuch as performers manipulating littleBits SynthKit modules are interested in the circuit-level details of analogue synthesis, the *notation models the domain directly*. For performers in other roles (perhaps a musician attempting to play a folk tune, or a technician trying to calibrate an oscilloscope) the roles are *not supported* by the notation.

9) Premature Commitment: Making changes to circuits is the same as building circuits from scratch, and general low viscosity, means that a circuit can be built in any order. Programmers are *not prematurely committed* to circuit structures or module settings.

10) Progressive Evaluation: Sounds can be produced so long as a power source and speaker are sensibly connected to a circuit. In practice when building or modifying circuits there will be a shorter or longer *delay in evaluation*, depending on the performer's manipulation of the circuit modules. Changes to module settings are *evaluated immediately* and immediately audible, again so long as the circuit is connected.

11) Provisionality: Modules can be moved and adjusted without being connected to an output, *supporting provisional arrangements* but of course without audible feedback (no progressive evaluation). Provisional arrangements could be monitored via a second speaker module connected to headphones: Selective Yellow eschews such private monitoring.

12) Abstraction: littleBits offer no abstraction facilities, nor is it possible to cut and paste (parts of) circuits. Tangibility here is *abstraction hating*.

V. Related Work

In "The Programming Language as a Musical Instrument", Blackwell and Collins [15] argue that livecoding transforms a programming language into a musical instrument, with the very difficulty of "performing" upon a programming system as a conscious æsthetic choice. In this paper, we make a complementary argument: that a musical instrument, the littleBits Synth Kit, can be considered as a programming language. As a programming language there are a range of other analyses that could be performed on littleBits, such as (for example) an analysis of the semiotic structure of littleBits programs [17], the use of metaphor in the design and naming of the circuit elements [18], or a usability analysis of the littleBits synthKit as a whole [19].

One of the first tangible live music programming language systems is the reacTable [20], using physical objects on a multi-touch table interface to produce music. (GaussBits [21] offers another similar features, based on magnets rather than cameras for sensing fiduicary markers.) Animations in the touch table make the physical objects appear live, giving feedback along with the generated sound. In contrast, littleBits objects are in fact live, not only providing performers with an affordance for tangible interaction but actually embodying the electronics producing the sound. Rather than relying on an external source to provide animated feedback, littleBits modules can give feedback directly — lighting indicator LEDs like the sequencer module, or with additional modules outside the synth kit, trigger servos, vibrators, buzzers, or motors — as well as produce sounds.

Livecoding has recently become the focus of more academic research, including a special issue of the Computer Music Journal [5], a Dagstuhl workshop [6], and a formal research network in the UK, livecodenetwork.org. Considered as a live programming environment, littleBits reaches level 4 on [22]'s scale [22], while many textual live coding environments are at level 3 [23].

VI. Discussion and Conclusion

In this paper we have described the littleBits KORG Synth Kit, described how it is played (or programmed). We have argued the Synth Kit can be considered an embodied programming language, have conceptualised our practice performing with the Synth Kit as livecoding in that programming language, and have analysed littleBits according to the Cognitive Dimensions framework.

There are still many dimensions of livecoding / performance with littleBits that we have not explored. Just as multiple livecoders can use computer networks to exchange code exchange code fragments during performances, LittleBits afford an opportunity for multiple performers (presumably using multiple Synth Kits) to physically exchange circuit components — perhaps individual modules, a sequencer, say, configured in a particular way, or collections of modules clipped together.

As Selective Yellow performs with a wide range of electronic music equipment, we plan to compare working with littleBits with other aspect of electronic music performance which also have some relationship to programming. If littleBits configuration can be considered as programming, how about physically patching a modular or semi-modular synthesizer, virtually patching a representation of a modular synthesizer within a computer, editing the actual patch on a one-knob-per-function analog synthesizer? Strange as it may seem, something about they way you physically move littleBits modules around and clip them together, changing they physical shape of the "program" seems closer to programing than patching together control-voltage inputs and outputs on a semi-modular or racked modular synthesizer — even though a relatively basic (semi-)modular design (say an MS-20 mini, or even a Microbrute) will offer more flexibility and power than the limited selection of modules in the Synth Kit.

The experience of littleBits livecoding is intensely mediated by the physical embodiment of the modules, and the ease of combining them into circuits. Both of these are consequences of the pedagogic goal of the littleBits designs [1]. LittleBits circuits can be plugged and unplugged without risk of damaging the component modules, resulting in an important freedom to explore that is as valuable for an improviser performing in public as it is for a student learning electronics. In this sense, littleBits once again resembles the pedagogy of textual programming, in particular the famous statement at the end of the first chapter of the ZX81 BASIC PROGRAMMING manual: *"Regardless of what you type in, you cannot damage the computer"* [24].

Quoting John Cage, Christoph Cox has argued that a crucial difference between European art music and sound art is that art music is typically closed: an event at a particular time, with a beginning, middle, and end; whereas sound art is an ongoing process that is "fundamentally open, without origin, end, or purpose." [25]. Selective Yellow's performance practice is much closer to a Cagean open sound process than a determinate musical event. Inasmuch as referentially transparent functional programs can easily describe ongoing processes there is some affinity between the paradigm of the programming language and the paradigm of the sound event it will produce.

On the other hand, many textual livecoding languages (ChucK, SuperCollider, Gibber, etc.) are imperative or object-oriented, rather than functional. Even more than in textual livecoding, playing or editing a littleBits program, whether by restructuring its circuits or "tweaking" the controls that parameterize each module, is a sequence of actions that happen in the real world, in a particular place and time, and that change both the program and the world. The physicality of littleBits modules, their very presence as manipulable objects, encourages an object-oriented view as much as a functional view — as the Scandinavians put it: *"A program execution is regarded as a physical model, simulating the behaviour of either a real or an imaginary part of the world."* [26]. Programming in littleBits rather than in BETA (or Simula, Smalltalk, or Haskell) we have an embodied, tangible physical model as our program, the execution of the program is the electrical behaviour of the circuits within the littleBits modules, and the sound that is produced is the behaviour of a part of the world — imaginary in the sense of invented, or imagined, not existing until brought into being by the live act of programming.

Finally, this research programme may offer new perspectives on two sides of an age-old question. On one side, why do some people (such as the littleBits developers) hate programming so much that "no programming" is promoted so highly as a feature of the littleBits systems? On the other side, why do others (such as textual livecoders) fetishize the accidents of 1980s-era programming as a means for producing music?

ACKNOWLEDGEMENTS

Thanks to Chris Wilson, for being the other half of Selective Yellow. Thanks to the anonymous reviewers for their comments. This work is partially supported by the Royal Society of New Zealand (Marsden Fund and James Cook Fellowship).

REFERENCES

[1] A. Bdeir, "Electronics as material: littleBits," in *Proc. Tangible and Embedded Interaction (TEI)*, 2009, pp. 397–400.

[2] ——, "littleBits, big ambitions!" `http://littlebits.cc/-littlebits-big-ambitions`, Apr. 2013.

[3] littleBits, "Sneak peek: The cloud block," `http://littlebits-.cc/cloud`, May 2014.

[4] J. Noble and T. Jones, "[demo abstract] littlebits synth kit as a physically-embodied, domain specific functional programming language," in *FARM*, 2014.

[5] A. McLean, J. Rohrhuber, and N. Collins, "Special issue on live coding," *Computer Music Journal*, vol. 38, no. 1, 2014.

[6] A. Blackwell, A. McLean, J. Noble, and J. Rohrhuber, "Collaboration and learning through live coding (Dagstuhl Seminar 13382)," *Dagstuhl Reports*, vol. 3, no. 9, pp. 130–168, 2014.

[7] T. Magnusson, "Herding cats: Observing live coding in the wild," *Computer Music Journal*, vol. 38, no. 1, pp. 8–16, 2014.

[8] B. Russell, Ed., *Erewhon Calling: Experimental Sound in New Zealand*. The Audio Foundation and CMR, 2012.

[9] D. McKinnon, "Centripetal, centrifugal: electroacoustic music," in *HOME, LAND and SEA: Situating music in Aotearoa New Zealand*, G. Keam and T. Mitchell, Eds. Pearson, 2011, pp. 234–244.

[10] C. Nilson, "Live coding practice," in *New Interfaces for Musical Expression (NIME)*, 2007.

[11] N. Collins, A. McLean, J. Rohrhuber, and A. Ward, "Live coding in laptop performance," *Organised Sound*, vol. 8, no. 3, pp. 321–330, Dec. 2003.

[12] A. Ward, J. Rohrhuber, F. Olofsson, A. McLean, D. Griffiths, N. Collins, and A. Alexander, "Live algorithm programming and a temporary organisation for its promotion," in `read\_me --- Software Art and Cultures`, O. Goriunova and A. Shulgin, Eds., 2004.

[13] T. Green, "Cognitive dimensions of notations," in *People and Computers V*, A. Sutcliffe and L. Macaulay, Eds. Cambridge, 1989, pp. 443–460.

[14] T. Green and M. Petre, "Usability analysis of visual programming environments: A 'cognitive dimensions' framework," *Journal of Visual Languages and Computing*, vol. 7, pp. 131–174, 1996.

[15] A. F. Blackwell and N. Collins, "The programming language as a musical instrument," in *Psychology of Programming Interest Group*, 2005, pp. 120–130.

[16] A. F. Blackwell, "Cognitive dimensions of tangible programming languages," in *First Joint Conference of EASE and PPIG*, 2003, pp. 391–405.

[17] J. Noble and R. Biddle, "Patterns as signs," in *European Conference on Object-Oriented Programming*, 2002.

[18] M. Duignan, J. Noble, P. Barr, and R. Biddle, "Metaphors for electronic music production in Reason and Live," in *APCHI*, 2004, pp. 111–120.

[19] K. Arnold, "Programmers are people, too," *ACM Queue*, vol. 3, no. 5, pp. 54–59, 2005.

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

[21] R.-H. Liang et al., "GaussBits: Magnetic tangible bits for portable and occlusion-free near-surface interactions," in *CHI*, 2013.

[22] S. Tanimoto, "VIVA: a visual language for image processing," *Journal of Visual Languages and Computing*, vol. 1, no. 2, pp. 127–139, 1990.

[23] L. Church, C. Nash, and A. F. Blackwell, "Liveness in notation use: From music to programming," in *Proc. PPIG*, 2010, pp. 2–11.

[24] S. Vickers, *ZX81 BASIC PROGRAMMING*, 2nd ed. Sinclair Research Limited, 1981.

[25] C. Cox, "From music to sound: Being as time in the sonic arts (excerpt from "Von Musik zum Klang: Sein als Zeit in der Klangkunst")"," in *Sound*, C. Kelley, Ed. MIT Press, 2011, pp. 80–87.

[26] O. Lehrmann Madsen, B. Møller-Pedersen, and K. Nygaard, *Object-Oriented Programming in the BETA Programming Language*. Addison-Wesley, 1993.