

Live music-making: a rich open task requires a rich open interface

Dan Stowell
Centre for Digital Music
Queen Mary University of London
dan.stowell@eecs.qmul.ac.uk

Alex McLean
The OAK Group
Department of Computer Science
University of Sheffield
alex@slab.org

In this position paper we present some themes of our research, strands of which reflect our title's assertion in various ways. Our focus here is on live music-making, in particular improvised or part-improvised performances which incorporate digital technologies.

Is music-making rich and open? Rich, yes, as evident from the many varieties of emotional and social content that a listener can draw out of music, meaningful from many (although not all) perspectives (Cross and Tolbert 2008). Even unusually constrained music styles such as minimalism often convey rich signification, their sonic simplicity having a social meaning within wider musical culture. Music is particularly rich in its inner relationships, with musical themes passed between musicians both consciously and subconsciously, weaving a complex tapestry of influence. And open, yes: the generative/composable nature of musical units means a practically unbounded range of possible music performances. While musical genres place constraints on music-making, such constraints are often seen as points of departure, with artists celebrated for pushing boundaries and drawing a wide range of cultural and meta-musical references into their work. As a result, new musical genres spring up all the time.

1. RICH INTERFACES

Against the rich and open background of music as a whole, we examine the use of computers in live music making. Famously, computers do nothing unless we tell them to. We can think of them as lumps of silicon, passively waiting for discontinuities amongst the electric and magnetic signals, which provide the on/off states of digital

representation. Computers operate somewhat like a mechanical music box, where pins on a cylinder are read by tuned teeth of a steel comb. In particular, computers are controlled by performative language, where describing something causes it to happen. However the magic of computation comes when such sequences of events describe operations upon themselves, in other words perform higher order, abstract operations. We can describe computers then as providing an active system of formal language for humans to explore. Musicians may choose to work directly in this system of discrete language, on the musical level of notes and other discrete events such as percussive strikes. Alternatively, they may use computer language to describe analogue systems such as traditional musical instruments, and expressive movements thereof. As such, computers allow us to engage with music either on the level of digital events or analogue movements, but we contend that the greatest potential for a musically rich experience lies in engaging with both levels, simultaneously.

1.1. Interfaces and metaphors

Many computer music interfaces are based on pre-existing music technology: the mixing desk, the step-sequencer grid, the modular synth patchbay. These (sometimes called design metaphors) have an advantage of providing a good leg-up for those users who are familiar with the original technologies, but can run into problems; such users may get unpleasant surprises when the analogy is incomplete, while unfamiliar users may face what seems like inexplicably-motivated design decisions. In particular, these interfaces are full of skeuomorphic design elements, originating in physical constraints which no longer apply. The user

may be left feeling as though they are dealing with a nonsensical metaphor, which induces unneeded limitations (such as running out of display space), and embeds now-irrelevant design decisions (e.g. based on wiring considerations).

Rigorous attempts to base computer interface design around coherent metaphors have consistently met with failure (Blackwell 2006). From this it would seem that structuring software design around fixed metaphors does not hold cognitive advantage, beyond helping users adjust to software interfaces in the short term. It seems likely that this is because such metaphors typically reflect neither the “problem space” (the target music domain) nor the breadth of possibilities provided by the computer. The target music domain is in any case hard to specify and may vary from genre to genre or track to track. If we assume that everyone has their own systems of metaphor (Cognitive Semantics; Lakoff and Johnson 1980), then we should instead develop interfaces that let people apply their *own* metaphors. This is an important part of our definition of an open interface.

1.2. Mapping performers’ gestures: experience from beatbox research

A specific musical example comes from one of our (DS’s) research into interfaces based on extended vocal techniques – in particular, beatboxing, which is a genre of vocal percussion (Stowell 2010, Section 2.2). Vocal expression is potentially a very rich source of information that could be used by musical interactive systems – pitch, loudness, timbre, linguistic content (if any) – and traditions such as beatboxing demonstrate a wide variety of timbral gestures, rich in signification.

One way to connect such a vocal signal to a computer system is by detecting vocal “events” in real time and classifying them (Stowell and Plumbley 2010, and citations within). However, vocal events can be robustly classified into only a small number of classes (especially in real time, where only the start of the event’s sound is known), so the end result is a system which can learn to trigger one of a small number of options – like playing a drum machine with only three or four buttons. Much of the richness of the audio input is discarded. This can create a system which is accessible for immediate use by amateurs, but doesn’t lead to long-term engagement as a tool for musical expression, certainly not for experienced beatboxers. A more expressive way to make use of the vocal source is to treat the timbral input data as a continuous space, and to try to recreate some of the nuance of performers’ continuous timbral gestures, for example by controlling a synthesiser such that it performs analogous gestures (Stowell 2010, Chapter 5). Any “classification” is

deferred to the perception of the listener, who can understand nuances and significations in a way that is beyond at least current technology. In a user study with beatboxers (discussed further below), this approach was found useful. Further, we informally observed the interface’s openness in the fact that the participating beatboxers found new sounds they could get out of the system (e.g. by whistling, trilling) that had not been designed in to it!

This research is one example in which designing for an open musical interaction can allow for richer musical possibilities. Similar lessons might be drawn about physical gesture interfaces, for example – should one discretise the gestures or map them continuously? The continuous-mapping approach has been used by some performers to achieve detailed expressive performance even when the mappings are simple.¹

2. PROGRAMMING LANGUAGES

How would one design a computer music interface that can allow for a rich, structured yet open-ended musical expression? One answer is to design for an interaction pattern that makes use of human abilities to represent ideas in a structured but unbounded way, to abstract, to make meta-references – all well-represented in the linguistic faculty. A grammatical interface is a consistent and so learnable interface. Many different levels of musical expression find a representation in formalised language, from music-theoretic abstractions, through pattern manipulations such as modulations and L-systems, down to sample-by-sample structures for audio synthesis and effects. Hence the grammatical interface represented by programming/scripting languages is used in some of the more open sound-and-music systems (SuperCollider, CMusic, Chuck, SAOL).

2.1. Live coding

Live coding is an emerging practice of creative public programming, used to make music and other art forms in a live context. It renders digital performance in some sense more transparent to the audience, allowing them to share in the creative process.

Improvised performance with a tightly-constrained system (such as a simple drum machine) can be expressive; but improvised performance with an open system (such as an audio-oriented programming environment) allows for an interaction that coherently gives access to many of the different levels of music-making in the digital system, from high-level structure to phrasing to sound design and effects.

¹ <http://ataut.net/site/Adam-Atau-4-Hands-iPhone>

Most current livecoders are using performance systems that have not been around for long enough to reach the well-publicised figure of 10,000 hours of practice for mastery; the virtuoso livecoder might not yet have been encountered. However many people spend many hours typing and/or programming in their daily lives, and one advantage of a programming interface over a skeuomorphic interface might be that it can recruit skills developed in neighbouring arenas.

A livecoder, like any improvising performer, is under pressure to do something interesting in the moment. Livecoders can use abstraction and scheduling so the notion of “in the moment” may be a little different to that for more traditional instrumental improvisers. It can lead to a lack of immediacy in how the performer’s actions relate to the music, which can sometimes deny the more raw physiological expressionism that some people seek in music. Hence it may be useful to combine the symbolic interaction of livecoding with open gesture-based expression; one of us (AM) has been doing this in collaboration with vocalists, thrash guitarists, drummers and banjo players. The other (DS) has taken both roles in solo live performance by combining livecoding with beatboxing. The cognitive load for one performer carrying out both roles is high, but the combination of continuous organic expression with symbolic abstraction helps to provide immediate multimodal access to the multiple levels of musical ideas. As we will see in the following section, it is possible to cast programming on its own in similar terms.

2.2. Visual programming notation

Programming language source code is generally considered as discrete, one dimensional text constrained by syntactical rules. Myers (1990, p. 2) contrasts *visual* programming languages as “any system that allows the user to specify a program in a two (or more) dimensional fashion.” This definition is highly problematic, for a number of reasons. First, several text based languages, such as Haskell and Python have two dimensional syntax, where vertical alignment is significant. Second, amongst those systems known as visual programming languages, it is rare that 2D arrangement has any real syntactical significance, including Patcher languages such as Puredata and Max/MSP (Puckette 1988). Indeed lines and boxes in visual programming languages allow non-visual, high dimensional syntax graphs of hypercubes and up.

The significance of visual notation then is generally as *secondary notation* (Blackwell and Green 2002), in that it is not syntactical but still of key importance to human readability. We can relate this to Dual Coding

theory (Paivio 1990), in treating visuospatial and linguistic representations as not being in opposition, but rather supporting one another, with humans able to attend to both simultaneously, experiencing an integrated whole. Visual layout therefore not only supports readability, but supplements code with meaningful expression that is in general ignored by the software interpreter.

Some computer music interfaces, such as the ReacTable by Jordà et al. (2007), Nodal by McIlwain et al. (2005) and Al-Jazari by Dave Griffiths (McLean et al. 2010) use visual layout in primary syntax. In the case of the ReacTable, Euclidean proximity is used to connect functions in a dataflow graph, and proximity and relative orientation are used as function parameters. In the case of Nodal and Al-Jazari, city block distance maps to time, in terms of the movements of agents across a grid. Inspired by the ReacTable in particular, one of us (AM) is developing a visual, pure functional programming notation based on Haskell, designed for live coding of musical pattern. This is a work in progress, but feedback from preliminary workshops with non-programmers has already been highly encouraging (McLean and Wiggins 2011). All four of the aforementioned visual programming languages allow, and with the exception of Nodal are designed primarily for live coding. Whereas research from the live coding field has mainly been concerned with time, we can think of this research as extending computer music notation into space.

We assert that integration between linguistic and spatial representations is what makes a musical experience rich. We can relate this to beat boxing, which is experienced both in terms of discrete instrumental events via categorical perception, and continuous expression within spatial experience. This is much like the relationship between the perception of words and prosody, respectively discrete and continuous, but both symbolic and integrated into a whole experience. By considering a programming notation as necessarily having both linguistic and visuospatial significance, we look to find ways of including both forms of representation in the human-computer interaction.

3. RICH AND OPEN EVALUATION

Much development of new musical interfaces happens without an explicit connection to HCI research, and without systematic evaluation. Of course this can be a good thing, but it can often lead to systems being built which have a rhetoric of generality yet are used for only one performer or one situation. With a systematic approach to HCI-type issues one can learn from previous experience

and move towards designs that incorporate digital technologies with broader application – e.g. enabling people who are not themselves digital tool designers.

Wanderley and Orio (Wanderley and Orio 2002) made a useful contribution to the field by applying experimental HCI techniques to music-related tasks. While useful, their approach was derived from the “second wave” task-oriented approach to HCI, using simplified tasks to evaluate musical interfaces, using analogies to Fitts’ Law to support evaluation through simple quantifiable tests. This approach leads to some achievements, but has notable limitations. In particular, the experimental setups are so highly reduced as to be unmusical, leading to concerns about the validity of the test. Further, such approaches do not provide for creative interactions between human and machine.

For live music-making, what is needed is more of a “third wave” approach which finds ways to study human-computer interaction in more musical contexts in which real-time creative interactions can occur. And live music-making can feed back into HCI more generally, developing HCI for expressive and ludic settings and for open interactions.

In my (DS) work I developed a structured qualitative evaluation method using discourse analysis (DA) (Stowell et al. 2009). DA originates in linguistics and sociology, and means different things to different people: at its core, it is a detailed analysis of texts (here, transcribed participant interviews) to elucidate the structured worlds represented in those texts. In the context of a user of a new interface, it can be used to explore how they integrate that interface into their conceptual world(s), which gives a detailed impression of affordances relatively uncontaminated by the investigator’s perspective.

This approach is useful and could benefit from further exploration, perhaps in different contexts of interface use. The approach bears an interesting comparison against that of Wilkie et al. (2010) who analyses musicians’ language using an embodied cognition approach, which differs in that it decomposes text using a set of simple metaphors claimed to be generally used in abstract thought. As in any exploratory domain, the approach which attempts to infer structure “directly” from data and the approach which applies a priori structural units each have their advantages.

Such rich and open evaluation approaches sit well with the nature of creative musical situations. Alternative approaches may be worthwhile in some cases, such as controlled experimental comparisons, but often risk compromising the musical situation. As one example from a slightly

different domain, Dubnov et al. (2006) conducts a numerical analysis of an audience’s self-evaluated response to a composed piece of music over time. The study went to great lengths to numerically explore audience response in an authentic musical context – commissioning a composed piece whose structure can take two configurations, attracting a concert audience willing to be wired up to continuous rating system, etc. Their results were fairly inconclusive, demonstrating mainly that such a scientific approach is at least possible if logistically difficult. (Simpler approaches in the same mould exist in the computer-games literature, where the audience can often be only one person (Mandryk and Atkins 2007).) In evaluating systems for music-makers we have the added complication that gathering concurrent data is generally not possible: self-reports would distract from the music-making process, while physiological measures (including brain activity sensors) are generally disrupted by the muscle movement impulses (and sweating) that occur in most music-making. Thus we see little prospect in the immediate future for concurrent protocols, hence the use of retrospective protocols in e.g. Stowell et al. (2009).

Music-making HCI evaluation is still very much an unfinished business: there is plenty of scope for development of methodologies and methods. Evaluation of music-making, like that of computer games, fits well with the developments in the HCI field that are called “third paradigm” (non-task-focused, experiential, ludic). But further: music-making is a key area in which the development of rich and open, yet structured, HCI approaches are crucial to the development of the field.

4. REFERENCES

- Blackwell, A. and Green, T. (2002). *Notational Systems – the Cognitive Dimensions of Notations framework*, pages 103–134. Morgan Kaufmann.
- Blackwell, A. F. (2006). The reification of metaphor as a design tool. *ACM Trans. Comput.-Hum. Interact.*, 13(4):490–530.
- Cross, I. and Tolbert, E. (2008). *Music and Meaning*, chapter Music and Meaning. Oxford University Press.
- Dubnov, S., McAdams, S., and Reynolds, R. (2006). Structural and affective aspects of music from statistical audio signal analysis: Special topic section on computational analysis of style. *Journal of the American Society for Information Science and Technology*, 57(11):1526–1536.
- Jordà, S., Geiger, G., Alonso, M., and Kaltenbrunner, M. (2007). The reacTable: Exploring the synergy between live music performance and tabletop

- tangible interfaces. In *Proc. Intl. Conf. Tangible and Embedded Interaction (TEI07)*.
- Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press, first edition edition.
- Mandryk, R. L. and Atkins, M. S. (2007). A fuzzy physiological approach for continuously modeling emotion during interaction with play technologies. *International Journal of Human-Computer Studies*, 65(4):329–347.
- McIlwain, P., McCormack, J., Dorin, A., and Lane, A. (2005). Composing with nodal networks. In Opie, T. and Brown, A., editors, *Proceedings of the Australasian Computer Music Conference 2005*, pages 96–101.
- McLean, A., Griffiths, D., Collins, N., and Wiggins, G. (2010). Visualisation of live code. In *Proceedings of Electronic Visualisation and the Arts London 2010*.
- McLean, A. and Wiggins, G. (2011). Texture: Visual notation for the live coding of pattern. In *Proceedings of the International Computer Music Conference 2011*.
- Myers, B. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123.
- Paivio, A. (1990). *Mental Representations: A Dual Coding Approach (Oxford Psychology Series)*. Oxford University Press, USA.
- Puckette, M. (1988). The patcher. In *Proceedings of International Computer Music Conference*.
- Stowell, D. (2010). *Making music through real-time voice timbre analysis: machine learning and timbral control*. PhD thesis, School of Electronic Engineering and Computer Science, Queen Mary University of London.
- Stowell, D. and Plumbley, M. D. (2010). Delayed decision-making in real-time beatbox percussion classification. *Journal of New Music Research*, 39(3):203–213.
- Stowell, D., Robertson, A., Bryan-Kinns, N., and Plumbley, M. D. (2009). Evaluation of live human-computer music-making: quantitative and qualitative approaches. *International Journal of Human-Computer Studies*, 67(11):960–975.
- Wanderley, M. M. and Orio, N. (2002). Evaluation of input devices for musical expression: borrowing tools from HCI. *Computer Music Journal*, 26(3):62–76.
- Wilkie, K., Holland, S., and Mulholland, P. (2010). What can the language of musicians tell us about music interaction design? *Computer Music Journal*, 34(4):34–48.