McPherson, Andrew; Tahiroglu, Koray

Idiomatic Patterns and Aesthetic Influence in Computer Music Languages

Published in:
Organised Sound

DOI:
10.1017/S1355771819000463

Published: 01/04/2020

Document Version
Publisher's PDF, also known as Version of record

Idiomatic Patterns and Aesthetic Influence in Computer Music Languages

ANDREW MCPHERSON and KORAY TAHİROĞLU

Queen Mary University of London, School of Electronic Engineering and Computer Science, UK. Email: a.mcpherson@qmul.ac.uk
Aalto University, School of Arts, Design and Architecture, Finland. Email: koray.tahiroglu@aalto.fi

It is widely accepted that acoustic and digital musical instruments shape the cognitive processes of the performer on both embodied and conceptual levels, ultimately influencing the structure and aesthetics of the resulting performance. In this article we examine the ways in which computer music languages might similarly influence the aesthetic decisions of the digital music practitioner, even when those languages are designed for generality and theoretically capable of implementing any sound-producing process. We examine the basis for querying the non-neutrality of tools with a particular focus on the concept of idiomaticity: patterns of instruments or languages which are particularly easy or natural to execute in comparison to others. We then present correspondence with the developers of several major music programming languages and a survey of digital musical instrument creators examining the relationship between idiomatic patterns of the language and the characteristics of the resulting instruments and pieces. In an open-ended creative domain, asserting causal relationships is difficult and potentially inappropriate, but we find a complex interplay between language, instrument, piece and performance that suggests that the creator of the music programming language should be considered one party to a creative conversation that occurs each time a new instrument is designed.

1. INTRODUCTION

Since the advent of electronic sound production, a strain of idealistic discourse has posited that the new technology could create any sound imaginable (Théberge 1997). Such claims have been made of early recording, analogue synthesis and digital synthesis, and they feature prominently in the marketing of novel digital musical instruments (McPherson, Morreale and Harrison 2019). Indeed, utopian predictions accompany the introduction of many new technologies. Whether the claims about music technology are intended as literal or metaphorical, historical hindsight shows that the range of sounds producible by early analogue electronics was in fact highly limited. Regarding control and interaction possibilities, future retrospectives may show that our current digital instruments are likewise narrowly constrained. Still, even if we were to achieve the utopian vision of a boundless space for musical exploration, we would still be left with the question of what possibilities musicians would choose to explore within it. The design of any tool favours certain types of thinking, certain modes of interaction, certain outcomes over others. It is this non-neutrality in the context of music programming languages which we explore in this article.

In musical interaction, as in many artistic fields, constraints can be a powerful creative motivator. Magnusson (2010: 65) describes the process of learning a digital musical instrument as '"getting a feeling" for the instrument's constraints, rather than engaging with its affordances'. Even the simplest of digital musical instruments (DMIs) can nonetheless give rise to wide variations in style (Gurevich, Marquez-Borbon and Stapleton 2012), and comparative studies have found that adding more degrees of freedom to a DMI might paradoxically reduce the scope of a performer's exploration (Zappi and McPherson 2018). On the one hand, constraints of acoustic instruments were often seen as inspiring while in DMIs they were more often seen as frustrating, perhaps out of the feeling that the instrument should be capable of anything. On the other hand, Magnusson and Hurtado Mendieta (2007) found that imposing constraints in music software was sometimes desirable, but that respondents drew a contrast between musically 'directive' software (ProTools, Cubase, GarageBand) and ostensibly open-ended music programming languages such as Csound, SuperCollider, ChucK, Pure Data (Pd) and Max.

This last point merits scrutiny. Unquestionably, languages such as Pd and SuperCollider that construct sounds from first principles present a wider space of possibilities with fewer limitations compared to digital audio workstations. But are these tools really less directive? To what extent, despite their hypothetical potential to express any imaginable sonic outcome, might these languages be nonetheless inscribed with their own specific aesthetic values and ideologies? How might we discover these ideologies?

In this article we examine these questions through the lens of idiomaticity. We focus on new digital musical instruments: instruments intended for live performance which involve a software component created in a music programming language. Many but not all of these instruments also include a hardware interface, but in this article we primarily examine the idiomatic patterns suggested by the software. We do

Organised Sound **25**(1): 53–63 © Cambridge University Press, 2020. doi:10.1017/S1355771819000463

not directly consider fixed media composition, but many of the ideas in the article may apply equally to software-based electroacoustic music beyond the realm of digital musical instruments.

In essence, we argue that DMI designers' actions will be shaped by what a given language makes most apparent, natural or easy to do, regardless of its (lack of) constraints. Through a review of the literature and correspondence with the creators of commonly used computer music languages, we will seek to understand some of the baseline assumptions and models inherent in each language. Next, through a survey of DMI creators, we will look for influences of the particular programming language on the aesthetic perspective of the instrument. We conclude with some reflections on what aspects of a language might determine its idiomatic patterns and how we might view the relationship between language creator and instrument designer.

2. IDIOMATICITY AND INFLUENCE

Musicians will speak of a passage of music being *idiomatic* to an instrument. The idiomaticity of a passage can be contrasted with its difficulty. Huron and Berec (2009: 115) explain that an idiomatic passage is one that 'of all the ways a given musical goal or effect may be achieved, the method employed by the composer/musician is one of the least difficult'. In the realm of instrument design, an analogy can be drawn to Jordà's *musical instrument efficiency*, defined as the ratio of musical output complexity to input complexity, while correcting for the diversity of control (Jordà 2005: 187).

Idiomaticity appears in many musical contexts. In improvisation, patterns which are natural to the body–instrument relationship tend to form the basis of musical material (Sudnow 1978; de Souza 2017). Composers often study how to write idiomatically for instruments, and many virtuosic showpieces are written idiomatically so that their apparent musical complexity exceeds the actual difficulty of execution on the instrument. Vasquez, Tahiroğlu and Kildal (2017) explore idiomatic composition practice as a way of generating a repertoire for new digital instruments, as well as development of a common framework for performance practices of novel instruments (Tahiroğlu, Vasquez and Kildal 2017) while conversely Gurevich (2017) examines how repertoire can be used to inspire new instruments. Tahiroğlu, Gurevich and Knapp (2018: 129) link idiomaticity to the development of individual style:

> A gesture vocabulary that is idiomatic to the new musical instrument establishes a set of playing techniques for that instrument, which is in turn an essential component of a developing a performance practice ... Only once such a repertoire of gestures and techniques has been established can the individual variations between performances begin to develop the potential for 'expression' or style.

Challenges to idiomaticity are also found in both acoustic and digital performance practice, including jazz guitarist Kurt Rosenwinkel's 'voluntary self-sabotage' or retuning his guitar to challenge his typical improvisation patterns (de Souza 2017) to the potentially engaging effect of incorporate disfluent characteristics into digital instruments (Bin, Bryan-Kinns and McPherson 2018). Idiomaticity in traditional practice is sometimes linked to virtuosity, but virtuosity too is often challenged in digital instrument practice (Gurevich and Treviño 2007; Morreale, McPherson and Wanderley 2018).

Idiomaticity need not be limited to embodied practice. For example, live coding performances involve different kinds of feedback and decision-making than traditional instrumental performance (Goldman 2019), but certain structures will nonetheless be more natural to the chosen language than others. Nash (2015) applies the Cognitive Dimensions of Notation framework (Green and Petre 1996) to two music software environments, examining each on dimensions such as *viscosity* (resistance to change), *visibility* (ease of viewing) and *hidden dependencies*. In considering what is idiomatic in a musical programming language, we should consider, as Nash (2015) does, that these dimensions may not have a single global quantity across a language, but that they might apply differently depending on the particular idea or structure being expressed in the language.

2.1. The latent influence of tools

Sergi Jordà (2005) brings in a discussion that 'music instruments are not only in charge of transmitting human expressiveness like passive channels. They are, with their feedback, responsible for provoking and instigating the performer through their own interfaces' (quoted in Gurevich and Treviño 2007: 108). Magnusson (2018: 79) similarly argues that 'instruments are actors: they teach, adapt, explain, direct, suggest, entice. Instruments are impregnated with knowledge expressed as music theory ... they explain the world'. For example,

> the piano keyboard 'tells us' that microtonality is of little importance (and much Western music theory has wholly subscribed to that script); the drum-sequencer that 4/4 rhythms and semiquavers are more natural than other types; and the digital audio workstation, through its affordances of copying, pasting and looping, assures us that it is perfectly normal to repeat the same short performance over and over in the same track. (Magnusson 2009: 171)

In music as in other areas of technology, an object must typically be understood in the context of the broader social and cultural ecosystem in which it operates (Green 2011). For example, the guitar is not merely a physical object with six strings and a fretted fingerboard; it is the locus of a rich and complex interplay of cultural associations and embodied practices (Harrison, Jack, Morreale and McPherson 2018). In developing new instruments, the problem confronting both the luthier and the organologist is how to account for the role of existing cultural associations in a new technology (Bijsterveld and Peters 2010; Horn 2013; Magnusson 2018). Bates (2012) suggests that even the personal identity of the performer might be changed through association with an instrument.

The encoding of epistemic knowledge within technologies is not unique to musical instruments; it has also been observed about instruments of science (Baird 2004; Tresch and Dolan 2013). Akrich (1992: 207–8) explains that

> sociologists of technology have argued that when technologists define the characteristics of their objects, they necessarily make hypotheses about the entities that make up the world into which the object is to be inserted ... A large part of the work of innovators is that of 'inscribing' this vision of (or prediction about) the world in the technical content of the new object.

The influence of music programming languages on creative practice is not necessarily expressed in terms of formal limitations on what is possible to create, in that a Turing-complete programming language is theoretically capable of anything. Rather, languages will differ in what ideas and structures are the most expeditious or obvious to code (Nash 2015). Analogies have been drawn between programming and craft practice (Lindell 2014; Blackwell 2018), in which code functions as a material, and programming is akin to the reflective dialogue between practitioner and material that has been theorised for other craft disciplines (Ingold 2009; Karana, Barati, Rognoli, Der Laan and Zeeuw 2015).

It is interesting to question through what mechanisms does a script which is embedded idiomatically within a musical technology shape human action. A human–computer interaction perspective might focus on affordances, whether apparent or hidden (Gaver 1991), though exploration of constraints may be equally salient (Magnusson 2010). In the context of embodied interaction, Tuuri, Parviainen and Pirhonen (2017: 503) propose the notion of *experiential control*, looking beyond 'control spaces merely as instrumental spaces delineated by sensors, physical performance and the related input information' towards the 'subjective and intentional viewpoint of the musician playing the instrument'. In this view, instruments can push the performer away from actions that are difficult or have no apparent function, while pulling them towards actions which are convenient or natural. Jack, Stockman and McPherson (2017) observe the rapid emergence of these *push* and *pull effects* when percussionists encounter an unfamiliar instrument, with initial techniques derived from hand drumming giving way within minutes to techniques narrowly directed to the sensing modality of the instrument.

The influence of technology on action is by no means limited to embodied interaction. In the context of digital musical instrument design, Newton and Marshall (2011: 251) created a toolkit with the expectation that designers would start with ideas of gesture and sound, but instead found that designers took a technology-focused approach: 'They started by examining the sensors that were available to them, the parameters these sensors could detect and where on the instrument they could be easily mounted. Only then would they think about the gestures that the sensors could be used for.' In Andersen's 'magic machines' creative ideation workshops with musicians, materials with acoustic properties (e.g., elastic bands) are deliberately excluded to avoid easy or obvious associations (Andersen and Wakkary 2019). Building on this workshop methodology, Lepri and McPherson (2019) found that designing fictional instruments with non-functional materials brought the aesthetic priorities of the individual designer into sharper focus, though even non-functional materials cannot be a neutral medium for ideation.

2.2. Instruments and pieces

In the community of digital instrument creators whose locus is the NIME (New Interfaces for Musical Expression) conference, a continuing debate exists on the applicability of traditional separable musical roles to musical practices with NIMEs (e.g., composer, performer, instrument builder) (Dobrian and Koppelman 2006; Gurevich and Treviño 2007). Gurevich (2017) endorses the view of Johnston (2016) that roles in NIME are 'contingent and dynamic', such that instruments might change identities, composers might produce interactive systems rather than notated scores, and performers might prioritise exploration and discovery over virtuosity. Goddard and Tahiroğlu (2013) suggest particular awareness and understanding of the evolving nature of the performer-instrument-composer relationship within specific contexts and cultures of new instruments which could help us to inform how these relationships may be facilitated in various ways during the design process of new instruments. A survey by Morreale et al. (2018) found that 'composed instruments', which are inseparable from a particular piece, were common among NIME designers.

The question is where the boundary of composed instruments can be found. Many live coding practitioners have developed their own languages or language variants; are these languages composed instruments? One respondent to the survey by Magnusson and Hurtado (2007) says of using the ixilang live coding language, 'Like, the pieces wouldn't be by me, they'd be by you.' Are more generalist music programming languages like SuperCollider or Pd also, to some limited extent, composed instruments? Could we say that *any* instrument, including traditional acoustic instruments, might be 'composed' in the sense that its identity is inherently bound up with the pieces at the time of its creation? Green (2011: 141) questions 'whether the propensity of digital systems to "script" action really is a special property of a given device or class of device when we consider the instrument as an assemblage of material and immaterial resources rather than as a single artefact'.

The purpose of asking these questions is not to suggest that every piece written with SuperCollider (or with a violin) is a variant of some hypothetical meta-composition, but rather to highlight that no instrument, whether acoustic or digital, emerges from a cultural vacuum. Instruments and music languages are influenced both by general theories of music and by particular pieces that were important to the designer and the time of their creation. We might reasonably expect these influences to be reflected in what is idiomatic to produce on the instrument. In turn, we have seen that the idiomatic strongly shapes the patterns of thought and action by subsequent musicians who use the instrument.

3. VALUES OF COMPUTER MUSIC LANGUAGES

The design of any new technology is situated in a social and material context, and the design will respond to values and ideas embedded in the tools (Suchman 1987). Designers of new digital instruments will likely be influenced not only by their own pre-existing aesthetic values (Lepri and McPherson 2019) but by the assumptions and conveniences of the software and hardware they use to create their instruments. Each computer music language will come with its own distinctive outlook. For example, interpretations or transformations of sonic features which occur in real-time audio synthesis plug-ins might differ considerably from those of dataflow programming languages or from those of pattern-based object-oriented languages (Lyon 2002). These differences in audio processing among languages may have more to do with differing basic paradigms for manipulating data rather than different aesthetic priorities among language creators.

The design process of a music programming language constitutes both method and approach in providing a domain, 'an abstract model of computation' (McCartney 2002: 61). In fact, this is the primary distinguishing feature that allows these environments to be programmable, not resulting in ever greater complexity in problem solving, but rather allowing us to think about an intuitive way to work on problem domains, 'without worrying about details that are not relevant to the problem domain of the program' (Zicarelli 2019). The challenge of the low-level music language is to 'provide a set of abstractions that makes expressing compositional and signal processing ideas as easy and direct as possible. The kinds of ideas one wishes to express, however, can be quite different and lead to very different tools' (McCartney 2002: 61). If in practice, digital musical instrument designers are influenced by core operating assumptions of the languages they use, then there is value in highlighting some of these assumptions and how they differ among popularly used tools.

To gain insight into the assumptions behind popular languages, we circulated a short questionnaire to creators and developers of five widely used languages: Max/MSP (Zicarelli 1991; Puckette 2002; Clayton 2019; Zicarelli 2019), Pure Data (Puckette 1997; Grill 2019; Puckette 2019), SuperCollider (McCartney 2002; Baalman 2019), Csound (Lazzarini, Heintz, Brandtsegg and McCurddy 2016; Lazzarini 2019) and ChucK (Wang, Cook and Salazar 2015; Wang 2019). The questions concerned the basic paradigm of manipulating data and parameters in the language, the perceived differences to other languages, and the ways that the language influences music or instruments created within it. Below we describe some emergent themes of these conversations.

3.1. Open-endedness

One theme that emerges across interviews, and in the publications about the languages, is the desire for the languages to be sufficiently open-ended to let the practitioner express their own sonic ideas without bias. Puckette (2002: 12) writes of the Max family of languages (Max/MSP, Pd, jmax) that the design 'goes to great lengths to avoid a stylistic bias on the musician's output.' In response to our questionnaire about Pd, Puckette highlights 'Pd's low-level primitives (phasor, but no band-limited sawtooth generator, for instance) – which is intended to encourage users to develop their own "sound" rather than guide them' (Puckette 2019). Thomas Grill (2019) concurs: 'I guess there is no such "canonical" organization on a perceptual sonic level at all.' Joshua Kit Clayton (2019) writes of Max/MSP: 'Specific sonic features are prioritized less within the environment than the ability to

access information at any level, prototype quickly and interactively, and to build relationships between different types of data streams.'

Marije Baalman (2019) similarly writes of SuperCollider: 'The environment by itself does not prioritise particular sonic features, rather the user creates her own sonic units from the smaller building blocks.' By contrast, Zicarelli (2019) sees Max/MSP as accepting limitations in service of learnability: 'we want to look at a problem space and imagine how someone could design within some subset of the space in such a way that there are only a few basic gestures that you need to learn; then you can combine the basic gestures in mostly arbitrary ways to make what you want. With this approach, we are not trying to cover 100% of what you can do in the domain.' For ChucK, the language's strong focus on explicit timing may lead to an aesthetic focus on 'a more immediate, deterministic mindset for specifying computer music programs' (Wang, Cook and Salazar 2015: 13).

3.2. Abstraction

The language creators also focus on abstraction and encapsulation as values of their languages. Victor Lazzarini (2019) writes that Csound 'is mostly "object-based" rather than class-based, as all the user sees are objects that implement a given unit-generator behaviour' but also notes that the fact that its variables hold signals allows the direct specification of signal-processing operations. Baalman (2019) highlights the progression in SuperCollider from low-level unit generators through to *SynthDefs* ('blueprints for synthesis units') to instantiated synth instances within its audio engine. She sees the most important features as 'the combination of flexibility (the possibility to livecode, change code on-the-fly) and the capacity to make abstractions (in SynthDefs, but also classes) to create building blocks of your own'. According to Zicarelli (2002: 45), an original design principle of Max was to implement only a few primitive objects while allowing the creation of abstractions from combinations of them: 'with encapsulation, users need not be concerned with whether an element was one of the basic ones Miller provided or one they made themselves'. Clayton (2019) emphasises that Max objects can be programmed in a variety of different languages.

3.3. Rapid prototyping

Iteration time is a concern of several language developers. Wang (2019) writes that 'ChucK is meant to support rapid experimentation through a kind of "on-the-fly programming" or "live coding"'. Live coding is also a key concern of SuperCollider, which supports on-demand evaluation of blocks of code. Over the past decade, SuperCollider has been used to host other live coding languages such as ixilang (Magnusson 2011), and its synthesis engine (which is separate from its language) has become a popular back-end for live coding languages such as TidalCycles (McLean 2014), whose syntax is based on Haskell, Overtone and Sonic Pi (Aaron and Blackwell 2013), both based on the Clojure Lisp dialect. SuperCollider's rapid iteration means 'the experimentation and exploration of a new concept can happen in the same time' (Baalman 2019). Baalman sees the text-based nature as supporting rapid experimentation: 'I can type faster than I can move the mouse.' In Max, the ability to 'prototype quickly and interactively' is a priority (Clayton 2019).

3.4. Dynamism (or lack thereof)

A contrast emerges between the data flow languages (Max and Pd) and the others with respect to dynamic instantiation of synthesis processes. Baalman highlights the importance of being able to dynamically define and create *Synth* units in SuperCollider, and Lazzarini (2019) explains that Csound 'uses the notion of an instrument as a class that may be instantiated a number of times (any, in the case of Csound) for a set period (which in Csound may be unlimited) starting at any given time from now into the future.' On the other hand, Zicarelli (2019) draws an analogy between Max and modular synthesis, and writes: 'The thing that Max can't do easily is manage N of something, where N is dynamic . . . If I cared about this kind of dynamic instantiation I would probably use SuperCollider. But a lot of music is covered by static systems.' Puckette (2019) similarly observes: 'The fact that signal connections are static makes it easier to make instrument-like patches and less convenient to make things that spawn variable numbers of voices.'

3.5. Aesthetic influence

In his 2002 paper on Max, Puckette writes: 'The design of Max goes to great lengths to avoid imposing a stylistic bias on the musician's output', pointing to the blank page that greets the new user (something that Zicarelli (2002) also highlights). Still, Puckette (2002: 13) concedes:

> Perhaps we will eventually conclude that no matter how hard one tries to make a software program culturally neutral, one will always be able to find ever more glaringly obvious built-in assumptions. But this is not a reason to stop designing new software; on the contrary, it gives hope that there may be many exciting discoveries still to be made as more and more fundamental assumptions are questioned and somehow peeled away.

The preceding themes already highlight divergent priorities for different languages: explicit specification of time in ChucK, scheduling and simultaneous access to high-level objects and low-level DSP in Csound, dynamism and flexibility of syntax in SuperCollider, an instrument-like paradigm and flexible data connections in Pd and Max. Puckette is likely correct that any software program cannot be culturally neutral, and this certainly aligns with other authors' observations (Haworth 2015; Nash 2015). In the next section we will examine how the influence of these languages might be expressed in the creation of digital musical instruments and compositions.

4. SURVEY OF INSTRUMENT CREATORS

To complement the perspective of the language developers, we conducted an online survey of digital musical instrument creators to seek insight into how their choice of language might influence their aesthetic choices. We expected from the beginning that this would be a difficult question to ask directly: designers might not acknowledge (or even be aware of) the possibly subtle ways that languages shape their instruments, and the survey should not be seen as diminishing their creative agency. We therefore sought to gather information on the languages used, instruments created, and pieces and performances with that instrument, to identify any notable associations.

The survey was approved by the ethics board at Aalto University and run with the university's Webropol service. It was circulated to mailing lists for digital musical instrument communities (NIME, SMC) and user groups for music programming languages (Pd, SuperCollider, Csound). The survey was anonymous in that no names or demographic data were collected, though respondents were invited to share names of their instruments and pieces. The survey contained 27 questions (8 multiple choice, 19 free text) spanning three themes: digital musical instrument, composition and performance. Participants were invited to name one instrument and a piece and performance associated with it.

The completion rate of surveys started was 39 per cent; in total 36 complete responses were received.[1] To create their instrument, participants reported using a variety of languages: Pd (7 respondents), SuperCollider (7), Csound (5), C/C++ (5), Max/MSP (3), others (5) and multiple languages within the same instrument (5). We asked the respondents to describe the sonic or musical language of their instrument. After a clustering into categories we found the following appearing more than once: ambient/drone/noise (5), algorithmic (3), gestural/mapping (2), live coding (2), physical modelling (2). Of course, we acknowledge that genre is a problematic concept (Haworth 2016) and that a diversity of micro-genres exists within the umbrella of electroacoustic music. Three of five algorithmic or live coding instruments were made with SuperCollider (another used Forth, the last used Tidal Cycles). Both physical modelling instruments were implemented in C/C++. Otherwise, no clear associations were evident: of the five ambient/drone/noise instruments, five different languages were used (C/C++, SuperCollider, Max, Pd, Csound).

A thematic analysis was performed on the responses about the relationships between language, instrument, piece and performance (Braun and Clarke 2006). We identified the following themes: one instrument for one piece, limitless possibilities of the languages, flexibility for transformation, interactivity, generative and random features in instruments, complexity of the processed involved in compositions, first the piece then the language, more people in low-level languages, appropriate software, unexpected uncertainty and responsive instruments. Combining these themes in groups resulted in three high-level themes: Constraints and Opportunities, Under the Influence of Language and Instrument as the Piecee.

4.1. Constraints and Opportunities

This theme is associated with how the choice of programming language or audio software environment influenced the design of their digital musical instrument. It is directly linked to the responses regarding the design and problem space of the programming language. Several responses mentioned limitless possibilities of their language:

> There is no user interface, it starts with a tabula rasa. (#33, using Tidal Cycles/Haskell)

> MaxMSP in particular allows easy creation of visual feedback systems for the instrument. I suspect I'd not have bothered with visual feedback if I started out using SuperCollider instead. While set of built-in objects and add-on libraries are seen as opportunity were key to making it. (#30)

> Pd doesn't really have a particularly preformed sense of what's possible within it. (#15)

> Csound has great flexibility to allow the realisation of the conceptual idea without limitations. (#25)

Constraints and Opportunities might be considered as the results of sequences of experimentation with the programming language. Whether the participants speak of opportunities from rapid prototyping capabilities, or constraints from conceptual ideas they

[1]Survey responses available at https://sopi.aalto.fi/dmiip/idiomaticity/survey/data2019 (accessed 20 November 2019).

applied to the design of the instruments, we see sources of influence which guide the development of the instrument.

4.2. Under the Influence of Language

While identifying a simple causal relationship between programming language and instrument aesthetics is a challenge, the Under the Influence of Language theme looks at more general effects on practitioner's design work:

> It [the choice of programming language] does influence for sure. Each environment has its own limitations, particularities and affordances somehow. (#28, Python/Pyo)

> Almost entirely predicated upon the SuperCollider language. Would be highly unlikely to build a similar instrument in a different environment. (#31)

> My software made it possible to build this instrument in a certain way. There will be ways to build it with other software, but it would be 'different'. The tool is the path. (#32, SuperCollider).

> Oh it [the language] didn't [influence the piece] we just needed something for real time mathematical modelling. (#11, Pd)

> It's the other way around. The instrument influenced the choice of programming language and audio environment. (#16, SuperCollider/OpenFrameworks)

We might draw an analogy between the role of the language and Ihde's *background relation* where technology shapes people and their surroundings without calling attention to itself (Ihde 1990). The influences on instrument design may be obvious (where an instrument depends on specific affordances of a language) or they may be subtle and latent. The influence of the language on the composition is subtler still. Responses to this question had several variations of answer: the composition was influenced by the language, the composition was influenced by the instrument and therefore only directly by the language, or the language/instrument was chosen/developed to suit the piece:

> The instrument came from Csound and the piece is almost entirely this instrument so the closest answer in 'totally'. (#19)

> The composition was heavily influenced by the specific features of the instrument. As most of the features of the instrument are connected with the employed languages and environments, I guess there is a certain connection between the composition and the technology. (#5, C++)

> This question should be reversed. My desire to compose algorithmically led me to identify appropriate software with which to do so. (#20, Csound)

4.3. Instrument as the Piece

We asked whether the instrument had been involved in the writing of the piece. A total of 83 per cent of the respondents who answered this question said it had. When asked whether the piece could be played on any other instrument, a plurality (41 per cent) of responses said it could not; 34 per cent said it could, and 25 per cent gave an intermediate answer (e.g., the piece could be played but it would sound different, or that it might be possible but they had not tried). In these numbers and in the text responses, we see a common theme that the instrument and the piece are either one and the same, or at least inextricably linked. This trend has been noted before in the NIME community (Morreale et al. 2018).

> It follows no strict time domain notation however. Every possibility of oscillographic sound/image correspondence in comes from the digital instrument. The work is impossible without it. (#3, Pd)

> I think the piece reflects the composer's first impression of the instrument. (#5, C++)

> The distinction between instrument and piece does not apply here. The piece is a computer program that renders the sound. (#21, Csound)

> It is quite central as all the music is tailored around the instrument. Digital lutherie is the work. (#24, SuperCollider/Arduino)

What emerges is not a straightforward chain of influence from language to instrument to piece, but rather a complex interplay where aesthetic goals will shape the choice of language, the language will in turn have an effect on the instrument, and the piece remains tightly intertwined with the affordances and constraints of the instrument. If these responses do not prove specific types of aesthetic influence from specific languages, at least they demonstrate that digital musical instrument creation is fertile territory for such influences to emerge. This balance is captured by respondent #20, who used seven different languages in their instrument: 'While I am sure the programming languages/environments still affected our work, we did a lot of thinking about the conceptualisation and representation of the instrument in more abstract terms (gesture, form, sound).'

5. DISCUSSION

The proposition that technology is a non-neutral mediator of human cognition has deep roots in philosophy (Ihde 1990; Verbeek 2005), and a practical mechanism for this influence can be seen by the extent to which musical instrument creation is not an abstract goal-directed exercise but rather one that explores and

responds to the opportunities and constraints of the tools (Magnusson 2010; Newton and Marshall 2011; Lepri and McPherson 2019).

Following Vallgårda and Fernaeus (2015: 173), we might consider certain approaches to digital musical instrument design to be a *bricolage* practice, where 'the bricoleur does not plan ahead but develops the project in-situ with concerns of interaction, physical form, and behaviour pattern not being hierarchically ordered a priori'. Specifically, we suggest that these in-situ explorations are channelled by what is idiomatic to the underlying tools, for two reasons. First, as with improvisation, decisions are often made in the moment in response to immediate opportunities and constraints of the environment (Magnusson 2010). This may be one reason that many music programming languages prioritise rapid iteration time: 'experimen tation and exploration of a new concept can happen in the same time' (Baalman 2019). Second, languages impose certain epistemic perspectives (Lindell 2014) which can elevate the perceived importance of certain design decisions while obscuring others.

5.1. Explicit and implicit qualities

The correspondence with language developers reported earlier in this article shows certain contrasts in what each language makes explicit. For example, *time* is explicitly represented in certain languages, including ChucK (described by Ge Wang as a 'first-class feature of the language' which supports a '"hyper-imperative" mindset') and Csound (whose score functionality allows the timing and duration of events to be explicitly controlled). The specification of time in Csound follows a scheduler model, where the timing of *score* events might be specified in seconds or in musical time with adjustable tempo; within synthesis processes (*instruments*), time is mostly implicit, with individual opcodes maintaining a distinction between audio and control sample rates. In ChucK, the time of every computation is specified to sample-level precision through the use of the *now* keyword. On the other end of the spectrum, time is implicit in data flow languages such as Max and Pd, whose mostly static signal graphs abstract away the underlying changes in data that flow through them. On the other hand, data relationships and mappings are easily represented in Max and Pd whose paradigm is partly inspired by modular synthesis (Zicarelli 2019), and in SuperCollider by the connections between unit generators, but perhaps less so within more imperative languages.

We propose that the *explicit* and the *idiomatic* are linked in music programming languages. In general, it will be more convenient to manipulate explicitly represented qualities in a language or parameter space than to create a desired effect by shaping implicit elements. The result will be design processes that prioritise the explicitly represented aspects of the tool.

By way of example, consider the creation of a MIDI controller using sensors and a data flow programming language. The fundamental unit of MIDI is the *message*, especially the *note on* and *note off* message. The former is characterised by two explicit parameters: a note number (organised in discrete semitones) and a velocity. Other musical elements such as rhythm or harmony are implicit, emerging only through interrelationships of multiple messages. As a result, a common approach especially among student creators is to build keyboard-like interactions based on triggers for discrete messages and data mappings from numerical sensor values to either note number or velocity. Where rhythm is considered, for example in a step sequencer environment, it is often based on a regular clock which is easily created from a timer object such as *metro* in Pd or Max.

5.2. Built-ins and presets

The aforementioned step sequencer example also highlights the influence not only of the language structure, but also of the particular objects and presets that are built into it. Zicarelli (2002) describes Max as 'more of an ecosystem than a language', pointing to Miller Puckette's original vision in the late 1980s of a minimal set of core objects which would support a diversity of community-contributed higher-level encapsulations. Today, hundreds of such objects (whether implemented directly in C or as abstractions of other objects) are included in the software.

In our survey, we see several responses highlighting reliance on particular built-in objects (Max/Pd), opcodes (Csound) or UGens (SuperCollider): one respondent (#19) explained the instrument 'grew from experiment with Csound opcodes I did not know'; another (#10) wrote that 'Pd's rich set of built-in objects and add-on libraries were key to making' the instrument. Here we might draw an analogy to the importance of the built-in presets on digital hardware and software synthesisers in defining how it is used in most performance situations: even where a full-featured parameter editor is available, it is likely that many musicians will choose among the presets.

Zicarelli's 2002 description of Max as an ecosystem is aimed at its openness to creating new elements and relationships, a 'universal language of meaningless numbers' (Zicarelli 1991), but the term 'ecosystem' could equally describe the role of Max as the locus of a developer and user community which shares patches, ideas, musical works and technical support. In the current era, a beginning instrument creator working with Max is less likely to engage in isolation with the generic affordances of the data flow environment, but rather

will borrow from, adapt and recombine example patches, seek guidance from online tutorials and forums, and share ideas with other developers. These influences cannot be directly attributed to the structure of the language itself but they are crucial to understanding the ideas that emerge. A corollary to this observation is that apparently similar languages such as Max and Pd may nonetheless have different aesthetic predilections depending on the built-in objects and the priorities of their active community members.

5.3. Limitations

Our survey of instrument designers finds acknowledgement of the influence of tools, but we also find a significant minority of respondents rejecting the premise that the instrument design follows the language (in many cases suggesting that the reverse was true: a language was chosen based on the needs of an instrument or piece). In general, it can be difficult to establish any definitive conclusion in that the influence of the language may be latent and subconscious. Designers following 'scripts [which] are often well hidden and concealed' (Magnusson 2009: 170) may not be fully aware of them, but on the other hand, a lack of self-reported influence should not be taken as evidence that a well-concealed script exists. A considerably larger sample size, or a comprehensive review of published instruments and performances, might yield a richer perspective by supporting statistical associations between the use of particular languages and aesthetic choices, independent of any self-reported influences.

We should also consider the possibilities that designers gravitate to particular languages because they support their musical outlooks, or that for some practitioners the community rather than the language is the primary source of aesthetic influence. These positions are not necessarily opposed to the idea that the language influences the instrument: by way of choosing a language, a designer might choose which scripts they want to be guided by while knowing that the influence of the script might manifest in unforeseen ways later in the design process.

In our correspondence with language developers, a common theme was the ability to create any sonic outcome starting from fundamental building blocks. However, this capability does not contradict the notion that each language might make certain ideas easier or more convenient than others (Nash 2015). If every language was equally capable of making any instrument, and every language promoted the same thought processes, one could justifiably ask why so many languages are needed and why so many of them continue to support vibrant user communities.

6. CONCLUSION

The intention of this article has been to draw together insights from several fields to seek the latent influence of music programming languages on digital musical instrument design, and indirectly on composition and performance. We support this investigation through correspondence with language creators and a community survey of instrument designers. Our purpose in this endeavour is not to create a material-deterministic view of the world where the creativity of the designer is trivialised as following a script written by the language developer, but rather to elucidate the subtle influences of tools which are often promoted for their ostensible open-endedness for any musical application.

We suggest that *idiomaticity* is a useful lens through which to view the influence of both instruments and languages. In every language, certain ideas and design patterns will be easier or more convenient than others, perhaps relating to what a language makes explicit versus what it leaves implicit. In the same way that idiomatic patterns of a traditional instrument affect improvisation and composition, what is idiomatic in a music language affects the in-situ exploration that characterises much of digital musical instrument design. Similar influences likely exist in electroacoustic composition, where the language or other software tool makes certain musical patterns more obvious or easy to execute than others.

We also suggest that the encoding of epistemic knowledge about music is not something unique to fully realised musical instruments or high-level music production software, but is also present in low-level languages regardless of whether they are theoretically capable of implementing anything. In this way, we endorse Magnusson's (2009) notion of the instrument as *epistemic tool* but propose that this designation can extend to *any* language for creating musical structures or interactions. The main difference with low-level languages is not the presence or absence of hidden scripts, but the extent to which they are concealed from the designer on first encounter.

While our findings are not intended as proof of particular aesthetic influences and the creativity of the individual practitioner always remains primary, we propose that digital musical instrument design could be seen as a dialogue across time and distance between the language creator and the instrument designer who ultimately summons the resources (and with them, the hidden script) of that language to suit their creative goals. In this way, language design becomes a kind of meta-creative act that can ultimately have aesthetic ramifications that carry on for decades.

Acknowledgement

Thank you to Marije Baalman, Joshua Kit Clayton, Thomas Grill, Victor Lazzarini, Miller Puckette, Ge Wang and David Zicarelli for sharing their thoughts on the languages they develop. This work was supported by the Academy of Finland (project 319946) and the UK Engineering and Physical Sciences Research Council under grant EP/N005112/1.

REFERENCES

Aaron, S. and Blackwell, A. F. 2013. From Sonic Pi to Overtone: Creative Musical Experiences with Domain-Specific and Functional Languages. *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling & Design*. ACM, 35–46.

Akrich, M. 1992. *The De-Scription of Technical Objects*. Cambridge, MA: MIT Press.

Andersen, K. and Wakkary, R. 2019. The Magic Machine Workshops: Making Personal Design Knowledge. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 112.

Baird, D. 2004. *Thing Knowledge: A Philosophy of Scientific Instruments*. Berkeley: University of California Press.

Baalman, M. 2019. Personal Communication. Email, 2 April.

Bates, E. 2012, The Social Life of Musical Instruments. *Ethnomusicology* **56**(3): 363–95.

Bijsterveld, K. and Peters, P. 2010. Composing Claims on Musical Instrument Development: A Science and Technology Studies' Contribution. *Interdisciplinary Science Reviews* **35**(2): 106–21.

Bin, S. A., Bryan-Kinns, N. and McPherson, A. P. 2018. Risky Business: Disfluency as a Design Strategy. *Proceedings of the International Conference on New Interfaces for Musical Expression*. Blacksburg, Virginia, USA.

Blackwell, A. 2018. A Craft Practice of Programming Language Research. *Proceedings of the Psychology of Programming Interest Group (PPIG) Conference*.

Braun, V. and Clarke, V. 2006. Using Thematic Analysis in Psychology. *Qualitative Research in Psychology* **3**(2): 77–101.

Clayton, J. K. 2019. Personal Communication. Email, 11 April.

De Souza, J. 2017. *Music at Hand: Instruments, Bodies, and Cognition*. Oxford: Oxford University Press.

Dobrian, C. and Koppelman, D. 2006. The E in NIME: Musical Expression with New Computer Interfaces. *Proceedings of the International Conference on New Interfaces for Musical Expression*. Paris, 277–82.

Gaver, W. W. 1991. Technology affordances. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 79–84.

Goddard, C. and Tahiroğlu, K. 2013. Situating the Performer and the Instrument in a Rich Social Context with PESI Extended System. *Proceedings of the Sound and Music Computing Conference*. Stockholm, Sweden: SMC, 368–75.

Goldman, A. 2019. Live Coding Helps to Distinguish between Embodied and Propositional Improvisation. *Journal of New Music Research* **48**(3): 281–93.

Green, O. 2011. Agility and Playfulness: Technology and Skill in the Performance Ecosystem. *Organised Sound* **16**(2): 134–44.

Green, T. R. G. and Petre, M. 1996. Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages & Computing* **7**(2): 131–74.

Grill, T. 2019. Personal Communication. Email, 4 March.

Gurevich, M. 2017. Discovering Instruments in Scores: A Repertoire-Driven Approach to Designing New Interfaces for Musical Expression. *Proceedings of the International Conference on New Interfaces for Musical Expression*. Copenhagen, Denmark: NIME, 163–8.

Gurevich, M. and Treviño, J. 2007. Expression and Its Discontents: Toward an Ecology of Musical Creation. *Proceedings of the International Conference on New Interfaces for Musical Expression*. New York City: NIME, 106–11.

Gurevich, M., Marquez-Borbon, A. and Stapleton, P. 2012. Playing with Constraints: Stylistic Variation with a Simple Electronic Instrument. *Computer Music Journal* **36**: 23–41.

Harrison, J., Jack, R., Morreale, F. and McPherson, A. 2018, When is a Guitar not a Guitar? Cultural Form, Input Modality and Expertise. *Proceedings of the International Conference on New Interfaces for Musical Expression*. Blacksburg, VA: NIME, 229–304.

Haworth, C. 2015. Sound Synthesis Procedures as Texts: An Ontological Politics in Electroacoustic and Computer Music. *Computer Music Journal* **39**(1): 41–58.

Haworth, C. 2016. 'All the Musics which Computers Make Possible': Questions of Genre at the Prix Ars Electronica. *Organised Sound* **21**(1): 15–29.

Horn, M. S. 2013. The Role of Cultural Forms in Tangible Interaction Design. *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. ACM, 117–24.

Huron, D. and Berec, J. 2009. Characterizing Idiomatic Organization in Music: A Theory and Case Study of Musical Affordances. *Empirical Musicology Review* **4**(3): 103–22.

Ihde, D. 1990. *Technology and the Lifeworld: From Garden to Earth* (no. 560). Indianapolis: Indiana University Press.

Ingold, T. 2009. The Textility of Making. *Cambridge Journal of Economics* **34**(1): 91–102.

Jack, R. H., Stockman, T. and McPherson, A. 2017. Rich Gesture, Reduced Control: The Influence of Constrained Mappings on Performance Technique. *Proceedings of the 4th International Conference on Movement Computing*. ACM.

Johnston, A. 2016. Opportunities for Practice-Based Research in Musical Instrument Design. *Leonardo* **49**(1): 82–3.

Jordà, S. 2005. Digital Lutherie: Crafting Musical Computers for New Musics' Performance and Improvisation. PhD thesis, Universitat Pompeu Fabra.

Karana, E., Barati, B., Rognoli, V., Der Laan, V. and Zeeuw, A. 2015. Material Driven Design (MDD): A Method to desIgn for Material Experiences. *International Journal of Design* **9**(2): 35–54.

Lazzarini, V. 2019 Personal Communication. Email, 1 April.
Lazzarini, V., Yi, S., Heintz, J., Brandtsegg, Ø. and McCurdy, I. 2016. *Csound: A Sound and Music Computing System*. Heidelberg: Springer.
Lepri, G. and McPherson, A. 2019. Making Up Instruments: Design Fiction for Value Discovery in Communities of Musical Practice. *Proceedings of the ACM Conference on Designing Interactive Systems (DIS)*. San Diego, USA.
Lindell, R. 2014. Crafting Interaction: The Epistemology of Modern Programming. *Personal and Ubiquitous Computing* **18**(3): 613–24.
Lyon, E. 2002. Dartmouth Symposium on the Future of Computer Music Software: A Panel Discussion. *Computer Music Journal* **26**(4): 13–30.
Magnusson, T. 2009. Of Epistemic Tools: Musical Instruments as Cognitive Extensions. *Organised Sound* **14**(2): 168–76.
Magnusson, T. 2010. Designing Constraints: Composing and Performing with Digital Musical Systems. *Computer Music Journal* **34**(4): 62–73.
Magnusson, T. 2011. ixi lang: A SuperCollider Parasite for Live Coding. *Proceedings of International Computer Music Conference*. Huddersfield, UK.
Magnusson, T. 2018. Ergomimesis: Towards a Language Describing Instrumental Transductions. *Proceedings of the International Conference on Live Interfaces*. Porto, Portugal.
Magnusson, T. and Hurtado Mendieta, E. 2007. The Acoustic, the Digital and the Body: A Survey on Musical Instruments. *Proceedings of the 7th International Conference on New Interfaces for Musical Expression*. New York City. http://www.ixi-software.net/survey/interesting_comments.html (accessed 20 November 2019).
McCartney, J. 2002. Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal* **26**(4): 61–8.
McLean, A. 2014. Making Programming Languages to Dance to: Live Coding with Tidal. *Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design*. Gothenburg, Sweden.
McPherson, A., Morreale, F. and Harrison, J. 2019. Musical Instruments for Novices: Comparing NIME, HCI and Crowdfunding Approaches. In S. Holland, T. Mudd, K. Wilkie-McKenna, A. McPherson and M. M. Wanderley (eds.) *New Directions in Music and Human-Computer Interaction*. Cham: Springer: 179–212.
Morreale, F., McPherson, A. and Wanderley, M. 2018. NIME Identity from the Performer's Perspective. *Proceedings of the International Conference on New Interfaces for Musical Expression*. Blacksburg, VA.
Nash, C. 2015. The Cognitive Dimensions of Music Notations. *Proceedings of the International Conference on Technologies for Notation and Representation (TENOR)*. Paris: IRCAM.
Newton, D. and Marshall, M. T. 2011. Examining How Musicians Create Augmented Musical Instruments. *Proceedings of the International Conference on New Interfaces for Musical Expression*, Oslo, Norway: NIME, 155–60.
Puckette, M. S. 1997. Pure Data. *Proceedings of the International Computer Music Conference*. San Francisco, CA.
Puckette, M. 2002. Max at Seventeen. *Computer Music Journal* **26**(4): 31–43.
Puckette, M. 2019. Personal Communication. Email, 25 March.
Suchman, L. A. 1987. *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge: Cambridge University Press.
Sudnow, D. 1978. *Ways of the Hand: The Organization of Improvised Conduct*. Cambridge, MA: MIT Press.
Tahiroğlu, K., Gurevich, M. and Knapp, R. B. 2018. Contextualising Idiomatic Gestures in Musical Interactions with NIMEs. *Proceedings of the International Conference on New Interfaces for Musical Expression*. Blacksburg, VA.
Tahiroğlu, K., Vasquez J. and Kildal, J. 2017. Facilitating the Musician's Engagement with New Musical Interfaces: Counteractions in Music Performance. *Computer Music Journal* **41**(2): 69–82.
Théberge, P. 1997. *Any Sound You Can Imagine: Making Music/Consuming Technology*. Middletown, CT: Wesleyan University Press.
Tresch, J. and Dolan, E. I. 2013. Toward a New Organology: Instruments of Music and Science. *Osiris* **28**(1): 278–98.
Tuuri, K., Parviainen, J. and Pirhonen, A. 2017. Who Controls Who? Embodied Control within Human–Technology Choreographies. *Interacting with Computers* **29**(4): 494–511.
Vallgårda, A. and Fernaeus, Y. 2015. Interaction Design as a Bricolage Practice. *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction.* ACM, 173–80.
Vasquez, J., Tahiroğlu, K. and Kildal, J. 2017. Idiomatic Composition Practices for New Musical Instruments: Context, Background and Current Applications. *Proceedings of the International Conference on New Interfaces for Musical Expression*. Copenhagen, Denmark: NIME, 174–9.
Verbeek, P. P. 2005. *What Things Do: Philosophical Reflections on Technology, Agency, and Design*. University Park, PA: Penn State Press.
Wang, G. 2019. Personal Communication. Email, 28 February.
Wang, G., Cook, P.R. and Salazar, S. 2015. Chuck: A Strongly Timed Computer Music Language. *Computer Music Journal* **39**(4): 10–29.
Zappi, V. and McPherson, A. 2018. Hackable Instruments: Supporting Appropriation and Modification in Digital Musical Interaction. *Frontiers in ICT* **5**: 26.
Zicarelli, D. 1991. Communicating with Meaningless Numbers. *Computer Music Journal* **15**(4): 74–7.
Zicarelli, D. 2002. How I Learned to Love a Program that Does Nothing. *Computer Music Journal* **26**(4): 44–51.
Zicarelli, D. 2019. Personal Communication. Email, 30 March.