Interacting with Musebots

Andrew R. Brown
Griffith University
226 Grey Street
South Brisbane, Australia.
andrew.r.brown@griffith.edu.au

Matthew Horrigan
Independent Researcher
5869 Fremlin
Vancouver, Canada
matthorriganmusic@gmail.com

Arne Eigenfeldt
Contemporary Arts
Simon Fraser University
Vancouver, Canada
arne_e@sfu.ca

Toby Gifford
Monash University
900 Dandenong Road
Caulfield East, Australia
toby.gifford@monash.edu

Daniel Field
Griffith University
226 Grey Street
South Brisbane, Australia.
daniel.field@griffithuni.edu.au

Jon McCormack
Monash University
900 Dandenong Road
Caulfield East, Australia
jon.mccormack@monash.edu

ABSTRACT
Musebots are autonomous musical agents that interact with other musebots to produce music. Inaugurated in 2015, musebots are now an established practice in the field of musical metacreation, which aims to automate aspects of creative practice. Originally musebot development focused on software-only ensembles of musical agents, coded by a community of developers. More recent experiments have explored humans interfacing with musebot ensembles in various ways: including through electronic interfaces in which parametric control of high-level musebot parameters are used; message-based interfaces which allow human users to communicate with musebots in their own language; and interfaces through which musebots have jammed with human musicians. Here we report on the recent developments of human interaction with musebot ensembles and reflect on some of the implications of these developments for the design of metacreative music systems.

Author Keywords
Musebots, interaction, musical agents, generative music, metacreation.

CCS Concepts
• **Applied computing** → **Sound and music computing**; Performing arts; • **Human-centered computing** → **Collaborative and social computing;** Collaborative and social computing theory, concepts and paradigms; Collaborative and social computing systems and tools.

1. INTRODUCTION
Musical metacreation, a subfield of computational creativity, is the art of "endowing machines with the ability to achieve creative musical tasks" [1]; including realtime procedural music generation. Human performance with metacreative music systems has typically involved bespoke protocols for interaction, often based on machine listening techniques. This tendency toward idiosyncratic approaches to interaction is not dissimilar to the ad hoc manner in which algorithmic composition methods are often assembled, famously described as 'Frankensteinian' [2]. In this article, we explore how the musebot framework, designed to support interoperability, can be used to integrate human and machine performers through a protocol that does not overtly privilege either type of musical actor.

The open source musebot protocol was originally developed to coordinate generative music software ensembles [3] and facilitate modularised prototyping of designs [4]. It also addresses the very real issue of researchers developing ad hoc systems in a variety of languages that are often difficult to share and compare.

One important aspect of the musebot protocol is its open-endedness: musebots share their internal states through broadcast network messages. While some messages are suggested within the protocol, most have been developed for the specific requirements of an ensemble. To date, musebots have been used in performances amongst themselves, with live musicians, with mechatronic instruments, in responsive accompaniment for video, and in long-duration (up to 8 hours continuously) installations.

Live improvisation with musebot ensembles continues a long tradition of interactive music systems. There have been several previous discussions about interactive music systems seeking to achieve a partnership between human and computer performers that are inspirational for our attempts to adapt the musebot framework to that end. These include performance systems such as Cypher [5], Hyperinstruments [6], GenJam [7], Duet for One Pianist [8], Voyager [9], live algorithms [10], OMax [11], Frank [12], Shimon [13], and Odessa [14]. Our work differs from these systems, which were designed to interact with performers, whereas musebots were designed to interact with one another, with human performers not initially considered in the protocol. Musebots are available online[1], with the messages they react to as well as generate, provided in human-readable sy

In this paper, we describe experiments with integrating human musicians into musebot performance, and some recent modifications to the musebot framework that will (hopefully) be integrated into an updated Musebot 2.0 specification. We discuss how these modifications were

NIME'18, June 3-6, 2018, Blacksburg, Virginia, USA.

[1] http://musicalmetacreation.org/musebots/

driven, in part, by the requirements of enabling seamless human performer interactions with musebot ensembles.

2. ORIGINAL MUSEBOT INTERACTIONS

The musebot framework has evolved as demands on it to operate in different performance contexts have developed. One original goal of the musebot framework was to allow for autonomous metacreative systems to communicate without the need for human interaction; in other words, as a potential avenue to progress higher in the metacreative taxonomy [15]. We focus here on the changes, and artistic reasons for change, driven by a growing interest in human participation in musebot ensembles.

The original musebot protocol[2] required a Conductor and any number of musebots. The Conductor acted as a network hub through which all musebot messages passed; additionally, it provided a timecode that coordinated all musebots, as well as having the ability to launch and "kill" any active musebot. The original implementation passed messages via OSC [16], with each musebot communicating with the Conductor via a unique UDP port. The Conductor assigned these ports by writing port numbers into a configuration file read by the musebots when they launched. This implementation required all musebots to exist within a directory on the same hard drive as the Conductor.

Musebots, developed in a variety of coding environments (including MaxMSP, Java, Pure Data, Extempore, and SuperCollider) were curated into ensembles; in these cases, the musebots demonstrated an ability to communicate with one another through agreed upon messages, as well as a musical compatibility. The ensembles were presented as ongoing installations, in which each ensemble would be launched and perform for 5 to 7 minutes, then be "killed", allowing the next ensemble to perform. Eigenfeldt [17] describes these installations in more detail.

3. ADAPTING MUSEBOTS FOR HUMAN INTERACTION

While musebots were used in a variety of metacreative situations in the first two years since their inception, they remained autonomous—without the need for (or benefit of) live human interaction. However, several musebot developers, including some of the authors, experimented with musebots interacting with live performers in a concert in Vancouver in July 2017 entitled *Play Nice: Musical collisions between humans and intelligent machines*. The event is documented online, including video of all performances, as well as a repository of all software developed[3]. These experiences underscored the need for an updated Musbot specification.

Several of the works in *Play Nice* explored sound-as-interface [18] and performance-as-interface [19] methods of human interaction by representing the live performative actions of humans as musebot messages, thereby placing a human agent within the virtual ensemble. This was done in a variety of ways. In *Hewn from Living Rock*, audio and MIDI data from a guitarist's performance on a MIDI guitar was analyzed by a dedicated musebot so as to produce a '/density' measure of the guitarist's playing, as well as an estimate of their present key, which were then broadcast to the rest of the currently running musebots.

In *The Indifference Engine*, the live performer's audio was analysed for audio features, including spectral centroid, loudness, onset detection for activity measure, and spectral flux. These features were then mapped to more general musebot parameters: for example, activity level to /arousal, and flux to /valence.

In *Moments*, musebots attempted to fulfill a spectral goal, as provided through a spectrum generated by a structural musebot: the live performer's audio was analyzed, and included in the complete live spectrum, which the musebots then used to realign their own goals. In performance, this resulted in musebots slowly moving away from the live performer's spectral areas so as to fill other spectral regions.

In *Clasm*, live human control over musebot parameters was exploited. An ensemble of audio musebots performed with a video musebot on a networked computer. Audio musebots were "cajoled" and "prodded" through the use of an external USB controller that had its MIDI messages translated into messages understood by the audio musebots. While individual musebots continued to send messages, such as /density, /valence, and /arousal, human interaction tended to push these values to extremes, thus overriding the more moderate generative values. The result was one in which the musebots tended to interact with one another more gently, while the human could demand fast and immediate changes.

Because two of the works involved musebots running on multiple computers, the musebot framework had to be adapted to allow for network performance. A special musebot, ServerBOT, was developed to implement network bridging, sending the messages from the musebots hosted on one's local computer to those hosted on a remote computer also running a ServerBOT. This allowed two computers to run independent musebot ensembles, while only one Conductor was providing timing information. This enabled developers to use ensembles on their own computers as before, albeit with additional messages coming in from remote musebots, although it also required initiating the ServerBOT with IP addresses from all computers, a process that was not always seamless in performance.

4. IMPROVISATIONAL INTERFACES

In a recent code jam[4] in Byron Bay, Australia, five musebot developers continued to explore the potential for networked museboting. A new working method materialized through daily jam sessions over the course of a week. In each jam, all five developers networked their respective musebots together, with each developer's musebots hosted on their respective computer. The developers then live-coded changes to their musebots to refine the musebots' behaviours, such that the group of musebots produced increasingly more sophisticated and coherent music. Each jam lasted approximately an hour, with the duration of the jam allowing the developers to observe and learn to influence emergent behaviours of the ensemble over time.

One of authors dedicated the codejam to developing an interface for human interaction in the musebot ensemble. The approach was to implement a human-representing musebot, acting as a 'code-wrapper' around the human player—whimsically termed an *algoskin*—so that they appeared like any other musebot to the rest of the ensemble. In practice, this meant a combination of typical bidirectional musebot messaging with human interpretation of received messages (displayed on a computer screen) that supported the human

[2] https://bit.ly/2uOUJE2

[3] http://musebots.weebly.com/play-nice.html

[4] http://musebots.weebly.com/byron-bay-2017.html

player listening to the musical output of the ensemble and responding accordingly.

The design goal for this algoskin was to facilitate an accessible and meaningful musical experience for the human player, without requiring any additional level of musical expertise. Musical creativity support systems like this lie somewhere between Digital Musical Interfaces (DMIs) and Interactive Music Systems (IMSs) and align with the 'democratizing' agenda of the NIME 2018 conference.

Where DMI design tends to focus on properties such as controllability, expressiveness, diversity and the capacity to demonstrate virtuosity [20], and IMS design tends towards the implementation of autonomous computational music agents for collaborative human-computer creativity, this algoskin adopted a hybrid metaphor which some of the authors [21] have previously described as an 'improvisational interface'; where generative music processes are used to *elaborate* on human input in a stylistically appropriate manner, potentially scaffolding human creativity in circumstances where complete human control would be difficult (see [22] for some examples)

Musebot ensembles suggest the possibility of such scaffolded interaction. As they are designed for autonomous multi-agent real-time music generation, a human with an appropriate algoskin may participate in the ensemble leveraging the high-level musical context negotiated through musebot messaging. In the case of the Byron Bay codejam, the primary musical features communicated in the ensemble were /density, /notepool, and /chordscale, as well as a /downbeat message broadcast at the time of each new downbeat. In the context of Western tonal music, particularly in popular genres, a good starting point for musically appropriate improvisation is to play in-key and in-time. This algoskin opted to provide two layers of pitch quantisation (chord and scale), leaving the timing in human hands.

The physical interface to the algoskin was a MIDI controller comprising a 4x4 square of velocity sensitive pads and various other dials and buttons. The pads were connected to a software synthesizer via the algoskin musebot which remapped the pads so that the left column produced notes quantised to the chord, and the remaining pads produced notes between the chord notes quantised to the scale.

The algoskin also transmitted musebot messages in certain circumstances. When a dedicated 'mode' button was toggled, the MIDI pads were disconnected from the soft-synth and quantisation removed. The unquantised pitches of pads subsequently pressed were stored in a pitch-class set. When this mode was toggled off, the pitch-class set was broadcast to the ensemble as a /notepool message. Also, a dedicated 'density' dial sent /density messages according to the dial's position when turned. This was a subversion of the intended meaning of the density message to broadcast the current density, and thus an example of agent misbehaviour discussed below.

The consensus of developers and performers was that the experience of interacting with the musebot ensemble through this interface was enjoyable, and for the most part musically appropriate. The musebot etiquette, as described elsewhere, was considered by them to be reasonably successful in providing musical 'space' for the human to shine through at times, and conversely to pick up the slack at others.

5. MUSEBOTS 2.0

The 2017 musebot code jam aimed to find a new working method that allowed for more instantaneous—and ostensibly musical—decision-making while coding. Formerly, musebots were created as standalone applications, fixed in their capabilities, then placed in ensembles running on a single computer. Coders reacted to their musebots like composers might listen to their fully scored music performed by an ensemble: listen, take notes, and alter the music/musebot offline, then repeat the process. It was hoped that through allowing musebots to remain on developers' separate individual computers, and interact through networked communication, a more improvisational development practice would emerge, in which developers could alter and adjust their musebots as they interacted within the ensemble. The resulting approach was live-coding as software development: developers altered and re-coded their musebots live, during jam sessions of up to an hour. Development through live-coding complemented the prior, more conventional, development process.

The Conductor/Server application was eliminated in favour of using a single port number with a broadcast IP address for inter-computer communication. Each individual computer user was tasked with managing the allocation of UDP server port numbers to dispatch to the (potentially multiple) musebots running locally on their machine. The timekeeping function of the Conductor was delegated to a separate musebot on a single computer, broadcasting a count of elapsed downbeats together with a tempo; musebots distributed on separate computers were responsible for generating their own timecode between the downbeat pulses.

This also resulted in an alteration, or at least adaptation, of the musebot messaging protocol. Formerly, messages were divided into three types: /mc messages from the Conductor, which included timing information; /agent messages, also from the Conductor, which allowed the Conductor to control individual musebot gain, as well as kill them; and /broadcast messages, which were messages originating from the musebots, and passed to the ensemble.

With the elimination of a central messaging hub – the Conductor – there was no longer a need for most of the /mc messages, nor the /agent messages. This change is substantial enough that we label this revised protocol Musebots 2.0[5]. The remaining messages were all /broadcast messages (including the new /downbeat message), thereby making the /broadcast message itself superfluous. Instead, we found it more useful to divide the actual messages being sent into the following types:[6]

- /event - messages about actions the musebot takes; the time sent is typically significant, with /event messages including note-events and parameter or controller changes;
- /harmony - any messages suggesting, or informing other musebots of, harmonic constraints;
- /time - messages coming from the time-generating musebot, including /downbeat, but also extending to /beat, /meter, and /tempo;
- /activity - messages to the ensemble that a musebot is active, such as the /alive message;
- /change - upcoming changes, such as /tempo or /meter;
- /analysis - messages from musebots that examined live audio, or self-analysis. For example, measured density output of the sending musebot.

Methodologically, it is significant that these changes to the musebot protocol resulted from the demands of artistic requirements to include human participation in the runtime ensemble activity.

[5] https://tinyurl.com/ybb9k9ml

[6] The slash(/) is a delimiter in the OSC protocol.

6. EMERGING AESTHETIC PRINCIPLES

The original musebot manifesto suggested that musebots need not be modeled upon human modes of interaction. While a great deal of effort has gone into developing usable machine listening software, relatively straight-forward human tasks, such as reliable beat detection and source separation, remain somewhat elusive. In many cases, musebots have explored non-human modes of collaborative music-making, explicitly in their ability to communicate developing plans and intentions, which would be cumbersome for human improvisors. Despite the provocative potential of such non-human interactions, in practice the authors take most of their current inspiration from human musical behaviours and evaluations [23].

Treating their musebots as artworks, the authors use their own aesthetic impressions as well as those of human performers and audience to validate musebot ensembles' successes and/or problems. They share several general agreements about the desired musical output of musebot ensembles. Musebots should produce human-scrutable musical material that evidences interaction between multiple agents. Since groups of musebots, rather than individual musebots, produce the music, developers prioritize intelligent collectives of musebots over individually intelligent musebots. They should display what, in human communication, is called rhetorical reinforcement [25]: actions must reinforce each other, differing from what they would be in response to random stimuli. The authors have not yet tested this quantitatively, however, they intend that over time their collections of musebots will display increasingly organized behaviour. Based on their experience live-coding musebot ensembles, the authors have determined several processes that they believe tend to cause musebots to produce diverse, but comprehensible, output.

Firstly, individual musebots need not produce virtuosic, or complete, music. This approach is similar to that espoused by live coding performers or algorithmic composers who also seek to utilise parsimonious generative processes that can be interestingly combined [26]. Musebot ensembles gain their complexity and potential power through their interaction and collective action. Simple individual musebots display more easily codifiable behaviour than complex ones, and since it is the behaviour of the ensemble, not the individuals, that produces music, individual musebots need only be sophisticated enough to respond musically in the role they play and to the messages they receive. Once a musebot ensemble has produced a desired result, new group behaviours are decided upon by the developers, who then add capabilities to their musebots to afford those.

Secondly, when coding, musebot developers focus on composing creative algorithms rather than utilities. Musebot developers are composers and musicians first, and developers only insofar as needed in order to achieve their musical goals. So, code virtuosity is always in the service of musicianship, and individual musicianship always in the service of group musicianship.

Thirdly, the authors advocate developing individual musebots by dealing with behavioural questions one at a time, coming up with a reductive solution for each and assessing emergent behaviour, rather than imagining a desired musical result and then trying to encode the capacity to produce it. The authors believe, as articulated by Sorensen and Brown [27] in their algorithmic generation of orchestral music, that this process will yield better results than trying to either envision the output of the system all at once.

Fourthly, musebot development does not necessarily emulate human thinking or the natural world, but takes inspiration from both, and in so doing sometimes reveals novel music-theory concepts. Combined with the practical principle of producing simple code, musebot algorithms therefore function like the best music theory, making maps simpler than the territory. Musebot developers tend to address the problem of ensemble behaviour by thinking from the perspective of each musebot, one at a time, and applying their own musical intuitions in determining the behaviour of the individual software agent.

7. METACREATIVE MUSICIANSHIP

As was the case with the advent of electronic music, metacreative music requires new skills from the humans composing and performing within it, a phenomenon the authors found particularly important in their work with musebots. As part of the post-digital musicianship [28] used for making musebots, the authors have begun cultivating a new kind of ear training which they call *algorithmic listening*: the ability to accurately estimate – whether upon reflection or in the heat of what Leman calls "the expressive moment" [29] – the *procedure* causing a musical event. Where traditional ear training concentrated on replicating either the physical event that caused a given musical stimulus (such as a specific piano key being struck or a specific instrument being played) or some abstraction of it (such as a notated pitch), and technical listening concentrated on the electrical process being applied to a given signal [29], algorithmic listening prioritizes the regular behaviour of the sounding object – for example the particular pattern of a bird's song – thus presenting the acoustic environment as an *algorithmic ecology* of interacting agents.

Through algorithmic listening, developers can understand the algorithms of other developers' musebots, as well as become more aware of algorithms present in the natural world to be used as a basis for "art-as-it-could-be" [30]. While the authors apply algorithmic listening to the development of rule-based AI, they speculate that it may also apply to musical machine learning applications, where the goals would be to 'hear' both the characteristics of the corpus used (in what might be called "topical listening" or listening for genre) in addition to the learning algorithm derived from processing it.

Another aspect of metacreative musicianship that becomes apparent from attempts to integrate human performance with musebot ensembles is the ability to welcome serendipitous, aesthetically pertinent, occurrences. Coding musebot behaviours does not always go to plan and sometimes the emergent results are surprisingly interesting and/or provide inspiration for future development. This has always been the case for artistic creativity and continues to be even in the digital age with its technical emphasis on precision and optimisation. In line with Cascone's notion of the post-digital [31]– that looks beyond notions of precision to accept the inherent glitches in technological (including algorithmic) processes as potential sources of creativity – this aspect of metacreative musicianship is an "acknowledgement of technology's essential character and its situatedness, including its character flaws" [28].

8. DETERMINING ALGORITHMIC BEHAVIOUR

Troubleshooting a musebot involves the combined concerns of creative coding and artificial intelligence design. One of

the objectives has always been to create musebots that interact in a coherent musical fashion. The introduction of human performers into the ensemble made this objective especially pertinent because what might be considered as errors in a machine-only context became, in a musician-machine context, misbehaviours to be coped with in real time by the musician. The authors distinguish *individual* and *collective* misbehaviours.

Individual misbehaviours are musical activities that would be considered rude if performed by a human. Some examples include: soloing continuously without leaving space for other agents in the ensemble; not playing for an entire performance; and/or constantly playing out of time and/or harmony. To avoid individual misbehaviours, the authors try to give each musebot an ability to demonstrate 'good taste' within the musical idiom, through an iterative process of development and testing. However, since individual corrections deal exclusively with the actions of an individual agent, they do not necessarily result in better performance from the ensemble as a whole.

Collective misbehaviours are phenomena, enacted by a group of agents, which deviate from the aesthetic desired by its human developers. Examples have included situations in which groups of musebots all played at once with a high degree of complexity (creating chaos), all layed out at once (creating silence), or insisted on playing in different keys. To correct collective misbehaviours, developers agree on conventions to which they program their musebots to conform. Such conventions include: adopting the keys of other musebots but generating their own key if no other is provided; and playing their most soloistic material only when no other musebot is soloing.

Avoiding 'rude' musebot behaviours is an important step in facilitating pleasing human engagement with a musebot ensemble, because, while the structure of a musebot ensemble does not privilege the humans interacting with it *in the moment of playing together*, it does privilege humans within the overall project, because human developers determine rules for aesthetic relevance and social interaction.

Playing with(in) the ensemble gives a human musical agent a perspective different from that of an outside audience. Just as great human musicians typically develop reputations based on what they are like to *play with* as well as to *listen to*—with both these reputations feeding into the artistic practice upon which they are judged—musebots are, to the humans who play with them, interactive artworks as well as producers of artistic sound.

9. CONCLUSION

This article has outlined explorations in human interactions with musebot ensembles. In the process, the musebot protocol has evolved to be effective for networking live music generation algorithms with each other and human performers. Musebot ensembles have provoked the development of new musical skills in the humans programming and using them. Consistent with its purpose, the musebot framework is extensible and has evolved in the process of expanded performance practices. Now at version 2.0 the musebot protocol will surely be applied to varied future applications, prompting further musical, technical and theoretical responses from its developers in the process.

10. ACKNOWLEDGMENTS

This research was made possible by a Griffith University / Simon Fraser University Travel Grant and supported by the Australian Research Council Discovery Grant DP160100166. Our thanks to these organizations for their support.

11. REFERENCES

[1] Pasquier, P., Eigenfeldt, A., Bown, O., and Dubnov, S. 2016. An introduction to musical metacreation. In *Computers in Entertainment 14*(2).

[2] Todd, P., and Werner, G. 1999. "Frankensteinian Methods for Evolutionary Music Composition." In *Musical Networks: Parallel Distributed Perception and Performance*, edited by Niall Griffith and Peter Todd. Cambridge, MA: The MIT Press.

[3] Bown, O., Carey, B., and Eigenfeldt, A. 2015. "Manifesto for a Musebot Ensemble: A platform for live interactive performance between multiple autonomous musical agents." In *Proceedings of the International Symposium of Electronic Art*.

[4] Eigenfeldt, A., Bown, O., and Casey, B. 2015. Collaborative Composition with Creative Systems: Reflections on the First Musebot Ensemble. In *Proceedings of the International Conference on Computational Creativity,* Park City, 134-141.

[5] Rowe, R. 1993. *Interactive Music Systems: Machine Listening and Composing*. Cambridge, MA: The MIT Press.

[6] Machover, T., and Chung, J. 1989. "Hyperinstruments: Musically Intelligent and Interactive Performance and Creativity Systems." In *International Computer Music Conference*, 186–90. San Francisco: ICMA.

[7] Biles, J. 1994. "GenJam: A Genetic Algorithm for Generating Jazz Solos." In *International Computer Music Conference*, 131–37. San Francisco: ICMA.

[8] Risset, J, and Van Duyne, S. 1996. "Real-Time Performance Interaction with a Computer-Controlled Acoustic Piano." In *Computer Music Journal 20* (1):62–75.

[9] Lewis, G. 2000. "Too Many Notes: Complexity and Culture in Voyager." In *Leonardo Music Journal* 10:33–39.

[10] Blackwell, T., and Young, M. 2005. "Live Algorithms." In *Artificial Intelligence and Simulation of Behaviour Quarterly* 122:7–9.

[11] Assayag, G., Bloch, G., Chemillier, M., Cont, A., and Dubnov, S. 2006. "Omax Brothers: A Dynamic Topology of Agents for Improvisation Learning." In *ACM Workshop on Audio and Music Computing for Multimedia, International Multimedia Conference*, 125–32. Santa Barbara: ACM.

[12] Casal, D. 2008. "Time after Time: Short-Circuiting the Emotional Distance between Algorithm and Human Improvisors." In *Proceedings of the International Computer Music Conference 2008*. Dublin, Ireland: ICMC.

[13] Hoffman, G, and Weinberg, G. 2011. "Interactive Improvisation with a Robotic Marimba Player." *Auton Robot* 31:133–53.

[14] Linson, A. 2014. "Investigating the Cognitive Foundations of Collaborative Musical Free Improvisation: Experimental Case Studies Using a Novel Application of the Subsumption Architecture." PhD Thesis, London: Open University.

[15] Eigenfeldt, A., Bown, O., Pasquier, P., and Martin, A. 2013. "Towards a Taxonomy of Musical Metacreation: Reflections on the First Musical Metacreation Weekend." In *Proceedings of the Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 40–47. Boston, MA: AAAI.

[16] Wright, M., & Freed, A. 1997. Open Sound Control: A New Protocol for Communicating with Sound

Synthesizers. In *Proceedings of the International Computer Music Conference*.
[17] Eigenfeldt, A. 2016. Musebots at One Year: A Review. In *Proceedings of the Musical Metacreation Workshop,* Paris.
[18] Di Scipio, A. 2003. Sound is the Interface: from interactive to ecosystemic signal processing. *Organised Sound*, 8(03):269-277.
[19] Brown A.R. 2018. Creative improvisation with a reflexive musical bot, *Digital Creativity*, 29:1, 5-18
[20] McDermott, J., Gifford, T., Bouwers, A., and Wagy, M. 2013. "Should Music Interaction be Easy". In *Music and Human Computer Interaction.* Heidelberg: Springer
[21] McCormack, J., and d'Inverno, M. 2016. "Designing Improvisational Interfaces". In *Proceedings of the 7th Computational Creativity Conference,* Paris.
[22] T. Gifford, S. Knotts, S. Kalonaris, J. McCormack, M. Yee-King, and M. d'Inverno. Computational systems for music improvisation. *Digital Creativity*, 29(1), 2018.
[23] Eigenfeldt, A., Brown, A. R., Bown, O. & Gifford, T. 2017. "Distributed Musical Decision-Making in an Ensemble of Musebots: Dramatic Changes and Endings." In *Proceedings of the International Conference on Computational Creativity,* 88–95. Atlanta, GA: Association for Computational Creativity.
[24] Brandt, A. 2011. *Sound Reasoning* Chapter 4: "Musical Emphasis." Retrieved from http://www.soundreasoning.org/.
[25] Sorensen, A., and Brown, A. 2007. "aa-Cell in Practice: An Approach to Musical Live Coding." In *Proceedings of the International Computer Music Conference*, 292–99. Copenhagen: ICMA.
[26] Sorensen, A., and Brown, A. 2008. "A Computational Model for the Generation of Orchestral Music in the Germanic Symphonic Tradition: A Progress Report." In *Sound: Space - The Australasian Computer Music Conference*, edited by Sonia Wilkie and Anthony Hood, 78–84. Sydney: ACMA.
[27] Ferguson, J., and Brown, A. 2016. "Fostering a Post-Digital Avant-Garde: Research-Led Teaching of Music Technology." *Organised Sound 21*(02):127–37.
[28] Leman, M. 2016. *The expressive moment: How interaction (with music) shapes human empowerment*. MIT press.
[29] Letowski, T. 1985. Development of technical listening skills: Timbre solfeggio. *Journal of the Audio Engineering Society*, *33*(4), 240-244.
[30] McCormack, J., Eldridge, A., Dorin, A., and McIlwain, P. 2009. "Generative Algorithms for Making Music: Emergence, Evolution, and Ecosystems." In *The Oxford Handbook of Computer Music*, edited by Roger T. Dean. New York: Oxford University Press.
[31] Cascone, K. 2000. "The Aesthetics of Failure: 'Post-Digital' Tendencies in Contemporary Computer Music." *Computer Music Journal 24*(4):12–18.