Collaborative Composition with Creative Systems: Reflections on the First Musebot Ensemble

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

Oliver Bown
Design Lab
University of Sydney
NSA, Australia
oliver.bown@sydney.edu.au

Benjamin Carey
Creativity and Cognition Studios,
University of Technology Sydney
NSW, Australia
Benjamin.Carey@uts.edu.au

Abstract

In this paper, we describe the musebot and the musebot ensemble, and our creation of the first implementations of these novel creative forms. We discuss the need of new opportunities for practitioners in the field of musical metacreation to explore collaborative methodologies in order to make meaningful creative and technical contributions in the field. With the release of the musebot specification, such opportunities are possible through an open-source, community-based approach in which individual software agents are combined to create ensembles that produce a collective composition. We describe the creation of the first ensemble of autonomous musical agents created by the authors, and the questions and issues raised in its implementation.

Introduction

Musical metacreation (MuMe) is an emerging term describing the body of research concerned with the automation of any or all aspects of musical creativity. It looks to bring together and build upon existing academic fields such as algorithmic composition (Nierhaus 2009), generative music (Dahlstedt and McBurney 2006), machine musicianship (Rowe 2004) and live algorithms (Blackwell and Young 2005). Metacreation (Whitelaw 2004) involves using tools and techniques from artificial intelligence, artificial life, and machine learning, themselves often inspired by cognitive and life sciences. MuMe suggests exciting new opportunities for creative music making: discovery and exploration of novel musical styles and content, collaboration between human performers and creative software partners, and design of systems in gaming, entertainment and other experiences that dynamically generate or modify music.

A recent trend in computational creativity, echoing other fields, has been to develop software infrastructures that enable researchers and practitioners to work more closely together, taking a modular approach that allows the rapid exchange of submodule elements in the top-down design of algorithms, facilitating serendipitous discovery and rapid prototyping of designs. It is widely recognised that such infrastructure-building can accelerate developments in the field for a number of reasons: getting large numbers of researchers to work together on larger-scale projects, forcing researchers to develop their software in a sharable format, enabling the like-for-like comparison of different system designs, education, and directly providing a large framework for further software development. Charnley *et al.* (2014), for example, has proposed a cloud-based collaborative creativity tool, supported by a web interface, that allows the rapid creation of text-based, domain specific, creative agents such as Twitter bots.

Our research in MuMe, which risks being too localised and insular, will benefit from a similar direction, and for this reason we have proposed the "musebot ensemble", a creative context designed to bring researchers together and get their realtime generative software systems playing together. We present a recent effort to design and build the infrastructure necessary to bring together community-created software agents in multi-agent performances, an elaboration on the motivation for doing so and the opportunities it offers, and some of the challenges this project brings. So far, we have set up a specification for musebot interaction, involving a community engagement process for getting a diversity of thoughts on the design of this specification, and we have built a number of tools that implement that specification, including musebots and a musebot conductor.

Following the outline of the system, we describe the creation of our first exploratory attempts to create and run a MuMe ensemble. We describe our initial experiences working creatively with networks of musebots. We conclude the paper with several open questions that were raised in the implementation of this collaborative compositional experience.

Towards a Collaborative Composition by Creative Systems

The established practice of creating autonomous software agents for free improvised musical performance (Lewis 1999) – the most common domain of activity in MuMe research – often involves idiosyncratic, non-idiomatic systems, created by artist-programmers (Rowe 1992, Yee-King 2007). A recent paper by the authors (Bown *et al.* 2013) discussed how evaluating the degree of autonomy in such systems is non-trivial and involves detailed discussion and analysis, including subjective factors. The paper identified the gradual emergence of MuMe specific genres — i.e., sets of aesthetic and social conventions — within which meaningful questions of relevance to MuMe research could be further explored. We posited that through

the exploration of experimental MuMe genres we could create novel but clear creative and technical challenges against which MuMe practitioners could measure progress.

One potential MuMe genre that we considered involves spontaneous performance by autonomous musical agents interacting with one-another in a software-only ensemble, created collaboratively by multiple practitioners. While there have been isolated instances of MuMe software agents being set up to play with other MuMe software agents, this has never been seriously developed as a collaborative project. The ongoing growth of a community of practice around generative music systems leads us to believe that enabling multi-agent performances will support new forms of innovation in MuMe research and open up exciting new interactive and creative possibilities.

The Musebot Ensemble

A musebot is defined as a "piece of software that autonomously creates music collaboratively with other musebots". Our project is concerned with putting together musebot ensembles, consisting of community-created musebots, and setting them up as ongoing autonomous musical installations. The relationship of musebots to related forms of music-making such as laptop performance is discussed in detail in our manifesto (Bown *et al.* 2015).

The creation of intelligent music performance software has been predominantly associated with simulating human behaviour (e.g., Assayag *et al.*). However, a parallel strand of research has shed the human reference point to look more constructively at how software agents can be used to autonomously perform or create music. Regardless of whether they actually simulate human approaches to performing music (Eldridge 2007), such approaches look instead at more general issues of software performativity and agency in creative contexts (Bown *et al.* 2014). The concept of a "musebot ensemble" is couched in this view. i.e., it should be understood as a new musical form which does not necessarily take its precedent from a human band.

Our initial steps in this process included specifying how musebots should be made and controlled so that combining them in musebot ensembles would be feasible, and have predictable results for musebot makers and musebot ensemble organisers. Musebots needn't necessarily exhibit high levels of creative autonomy, although this is one of the things we hope and expect they will do. Instead, the current focus is on enabling agents to work together, complement each other, and contribute to collective creative outcomes: that is, good music.

This defines a technological challenge which, although intuitive and easy to state, hasn't been successfully set out before in a way that can be worked on collaboratively. For example, Blackwell and Young (2004) called on practitioners to work collaboratively on modular tools to create live algorithms (Blackwell and Young 2005), but little community consensus was established for what interfaces should exist between modules, and there was no suitably compelling common framework under which practitioners could agree to work. In our case, the modules correspond clearly to the instrumentation in a piece of music, and the context is more amenable to individuals working in their preferred development environment.

In order for musebots to make music together, some basic conditions needed to be established: most obviously the agents must be able to listen to each other and respond accordingly. However, since we do not limit musebot interaction to human modes of interaction, we do not require that they communicate only via human senses; machine-readable symbolic communication (i.e., network messaging) has the potential to provide much more useful information about what musebots are doing, how they are internally representing musical information, or what they are planning to do. Following the open community-driven approach, we remain open to the myriad ways in which parties might choose to structure musebot communication, imposing only a minimal set of strict requirements, and offering a number of optional, largely utilitarian concepts for structuring interaction.

Motivation and Inspiration

One initial practical motivation for establishing a musebot ensemble was as a way of expanding the range of genres presented at MuMe musical events. To date, these events have focused heavily on free improvised duets between human instrumental musicians and software agents. This format has been widely explored by a large number of practitioners; however, it runs the risk of stylistically pigeonholing MuMe activity.

For the present project, the genre we chose to target was electronic dance music (EDM), which, because it is fully or predominantly electronic in its production, offers great opportunities for MuMe practice; furthermore, metacreative research into this genre has already been undertaken (Diakopoulos *et al.* 2009; Eigenfeldt and Pasquier 2013). The 2013 MuMe Algorave (Sydney, 2013) showcased algorithmically composed electronic dance music, an activity originally associated with live coding (Collins and McLean 2014). However, rather than presenting individual systems with singular solutions to generating such styles, it was agreed that performances should be collaborative, with various agents contributing different elements of a piece of music. This context therefore embodies the common creative musical challenge of getting elements to work together, reconceived as a collective metacreative task. Although the metaphor of a *jam* comes to mind in describing this interactive scenario, we prefer to imagine our agents acting more like the separate tracks in a carefully crafted musical composition.

We acknowledge the relationship of musebot ensembles to multi-agent systems (MAS); however, rather than concentrate upon the depth of research within this field, we have designed the specification in such a way so as to combine generality and extensibility with domain specific functionality. As will be described, at heart the musebot project is simply a set of message specifications that are

domain specific to the idea of multiple *musical* agents. We feel that there is no need to draw on more specific MAS tools and specifications, as there is nothing that is not simply handled by the definition of a few messages. Taking this general approach has the advantage that if people want to incorporate the musebot specification into their MAS frameworks, they can. It is intentionally barebones so that it is simple for people to adapt their existing agents to be musebots. At the same time, we also acknowledge that MAS have been incorporated into MuMe in typically idiosyncratic ways, replicating the interaction between human musicians (Eigenfeldt 2007) while also exploring non-human modes (Gimenes *et al.* 2005); our intention is for musebots to explore both approaches.

We summarise the other opportunities we see in pursuing this project as follows, beginning with items of more theoretical interest, followed by those of more applied interest:

- Currently, collaborative music performance using agents is limited to human-computer scenarios. These present a certain subset of challenges, whereas computer-computer collaborative scenarios would avoid some of these whilst presenting others. Such challenges stimulate us to think about the design of metacreative systems in new and potentially innovative ways;
- It provides a platform for peer-review of systems and community evaluation of the resulting musical outputs, as well as stimulating sharing of code;
- It provides an easy way into MuMe methods and technologies, as musebots can take the form of the simplest generative units, whereas at present the creation of a MuMe agent is an unwieldy and poorly bounded task;
- It outlines a new creative domain, which explores new music and music technology possibilities;
- It encourages and supports the creation of work in a publicly distributed form that may be of immediate use as software tools for other artists;
- It allows us to build an infrastructure which can be useful for commercial MuMe applications. Specifically, it provides a modular solution for the metacreative workstations of the future;
- It defines a clear unit for software development. Musebots may be used as modular components in other contexts besides musebot ensembles.

The Musebot Agent Specification

An official musebot agent specification is maintained as a collaborative document, which can be commented on by anyone and edited by the musebot team[1]. An accompanying BitBucket software repository maintains source samples and examples for different common languages and platforms[2].

A musebot ensemble consists of one musebot conductor (MC) and any number of musebots, running on the same machine or multiple machines over a local area network (LAN). The MC is notified of each musebot's location and paths to its directories, allowing it to build an inventory of the available musebots in the ensemble. Thus, for the user, adding a musebot to the ensemble simply means downloading it to a known musebot folder. Musebots contain *config* files that are controlled by the MC, and human/machine readable *info* files that give information about the musebots.

The MC is responsible for high level control of connected musebot agents in the network, setting the overall clock tempo of the ensemble performance and managing the temporal arrangement of agent performances (see Tables 1 and 2). The MC also assists communication between connected agents by continuously broadcasting a list of all connected agents to the network, and relaying those messages that musebots choose to broadcast. The MC is not necessarily "in charge". Currently, it is just a simple GUI program that allows users to control musebots remotely. Ultimately we will automate ensemble parameters such as tempo and key *either* by making specific variants of the MC, *or* by writing dedicated planning agents that issue instructions to the MC, *or* by allowing a distributed self-organising approach in which different agents can influence these parameters. These are all valid designs for a musebot ensemble.

/mc/time <double: tempo in BPM> <int: ticks>
This is the clock source and timing information. A beat/tick count, starting at zero and incrementing indefinitely, is sent at a rate of 16 ticks per beat at the specified tempo, to be used for synchronising your client bot. The downbeat is on (tick % 16 == 0).
/mc/agentList <string: musebot ID> [<string: musebot ID> …]
List of connected musebots in your network. Use this list to reveal messages sent from specific musebots,
/mc/statechange <string: {first,next,previous,any}>
This parameter is designed to facilitate high level state changes, which could be anything, depending on the program; however some examples might be overall density of events, range/register, key changes, change in timbre etc.

Table 1. Example messages broadcast by the MC to all musebot agents.

/agent/kill (no args)
Exit gracefully upon receiving this message from the MC.
/agent/gain <double: gain> [<double: duration ms>]
Scale your output amplitude, used to apply a linear multiplication of your output audio signal.

Table 2. Example messages sent between the MC and specific musebot agents.

[1] tinyurl.com/ph3p6ax

[2] bitbucket.org/obown/musebot-developer-kit

Musebots may broadcast any messages they want to the network, providing they maintain their unique name space allocated for inter-musebot communication (see Table 3). Our musebot specification states that a musebot should also "respond in some way to its environment", which may include any OSC messages (Wright 1997) as well as the audio stream that is provided: a cumulative stereo mix of all musebot agents actively performing. It should also not require any human intervention in its operation. Beyond these strict conformity requirements, the qualities that make a good musebot will emerge as the project continues.

/broadcast/statechange <string: musebot ID> <string: {first,next,previous,any}>
Locally controllable high level state change. Use this parameter if you want to prompt other clients to make changes to their high level state. Equally, respond to this message if you want other musebots to prompt high-level changes.
/broadcast/notepool <string: musebot ID> <int_array: pitch class MIDI values>
A list of MIDI note values, pitch classes only, no octave info, to be shared with the network - e.g. chord or scale you are currently playing.
/broadcast/datapool <string: musebot ID> <double_array: datapoints>
Array of floating-point values.

Table 3. Example messages broadcast by musebot agents. These messages are speculative, and open for discussion.

The First Musebot Ensemble

At the time of writing, a draft musebot conductor is implemented and published and a call has gone out for participation in the first public musebot ensemble. Our first experiments with making musebot ensembles followed the obvious path of taking the systems we have already created and adapting them to fit the specification. This step constituted provisional user testing of the specification and support tools and also gave us a sense of what sort of creative and collaborative process was involved in working with musebots.

We present two studies here. In the first case, the first author built a musebot ensemble entirely alone. The first author works regularly with multi-agent systems within his MuMe practice, so this was a natural adaptation of his existing approach. In the second study, each of the authors contributed a system that they had developed previously, and we looked at the ways that these systems could use the musebot specification to interact musically.

First Author Working Alone

In the first study, several musebots were designed in isolation by the first author. While lacking the musebot goal of cooperative development, the situation did allow for the design of ensembles with a singular musical goal, including specific roles for each musebot. For example, a *ProducerBot* was created that functions to control various other instrumental bots – a *DrumBot*, a *PercussionBot*, a *BassBot*, a *KeyboardBot*, etc. – in a hierarchical fashion. The organisation of such an ensemble reflects one conception in achieving a generative EDM work, in which each run produces a new composition whose musical structure is generated by the *ProducerBot*, and the musical surface is produced and continuously varied by the individual instrumental musebots. Such a design has been previously implemented by the first author (Eigenfeldt 2014) to produce successful musical results. This top-down, track-by-track breakdown of relations between musical parts is of course completely familiar to users of DAWs, with the difference that each track is a generative process that receives high-level musical instructions from the *ProducerBot*. In this case, the *ProducerBot* sends out information at initialisation, including a suggested phrase length (i.e. 8 measures), and subpattern, which represents how the phrase repetition scheme can be represented (i.e. aabaaabc). It individually turns instrumental musebots on and off during performance, including syncronising them at startup. Furthermore, it sends a relative density request – a subjective number of possible events to perform within a measure – every 250 milliseconds, as well as progress through the current phrase. Lastly, at the end of a phrase, it may send out a section message (i.e. A B C D E). When an instrumental musebot receives this section descriptor, it looks to see if it has data stored for that section: if not, it stores its current contents (patterns), and generates new patterns for the next section; if it does have data for that section, it recalls that data, thereby allowing for large-scale repetition to occur within the ensemble.

As with a DAW, via the musebot specification, we inherently allow for community contributions that accept specific instructions from the *ProducerBot*: swapping a different *BassBot*, for example, in the ensemble would result in a different musical realisation, as it is left to the musebots to interpret the performance messages.

Multiple Authors Working Together

In the second study, the three authors brought together existing systems into the first collaboratively made musebot ensemble. No assumptions were made in advance about how the systems would be made to interact, except that the second and third authors drew their contributions from existing work with live algorithms in an improvisation context (Blackwell and Young 2005), where the audio stream is typically the only channel of interaction.

A *BeatBot* was created by the first author, which combines the rhythmic aspects of both the former drum and percussion musebots, together with the structure-generating aspects of the *ProducerBot*, resulting in a complex and autonomous beat generating musebot. With each run, a different combination of audio samples is selected for the drums and four percussion players, along with constrained limitations to the amount of signal processing applied. A musical form is generated as a finite number of phrases, themselves probabilistically generated from weightings of 2, 4, 8, 16, and 32 measures. Each phrase has

a continuously varying density, to which each internal instrument responds differently by masking elements of its generated pattern. The metre is generated through additive processes, combining groups of 2 and 3, and resulting in metres of between 12 and 24 sixteenths. Finally, the amount of active layers for each phrase is generated. All of the generated material – metre, phrase length, rhythmic grouping, density, and active layers – is broadcast to the ensemble as messages.

The second author's *DeciderBOT* was adapted from his live algorithm system *Zamyatin*, an improvising agent that is based upon evolved complex dynamical systems behaviours derived from behavioural robotics (Bown *et al.* 2014). The internal system controls a series of voices that are hand-coded generative behaviours. *Zamyatin* is most easily described as a reactive system that comes to rest when presented with no input, and is jolted to live when stimulated by some input. The stimulation can send it into complex or cyclic behavior.

The final contribution to the first musebot ensemble was *_derivationsBOT*, designed by the third author. An adapted version of the author's *_derivations* interactive performance system (Carey 2012), *_derivationsBOT* was designed to provide a contextually-aware textural layer in the musebot ensemble, responding to a steady stream of audio analysis from the other bots connected to the network. During performance, *_derivationsBOT* analyses the overall mix of the musebot ensemble by segmenting statistics on MFCC vectors analysed from the live audio. The musebot compares these statistics with a corpus of segmented audio recordings, retrieving pre-analysed audio events to process, that compliment the current sonic environment. Synchronised to the overall clock pulse received from the MC, a generative timing mechanism conducts six internal players that process and re-synthesise these audio events via various signal processing. Importantly, the choice of audio events made available for processing is based upon comparisons both between statistics analysed from the live audio stream, as well as statistics passed between the internal players themselves. Thus, without audio input for analysis *_derivationsBOT* self-references, imbuing it with a sense of generative autonomy in addition to its sensitivity to its current sonic environment. To facilitate this, *_derivationsBOT* is randomly provided an internal state upon launch, enabling the musebot to begin audio generation with or without receiving a stream of live audio to analyse.

With the three musebots launched, a quirky, timbrally varied, somewhat aggressive, EDM results. Like much experimental electronic music, the listening pleasure is partly due to the strangeness and suspense associated with the curious interactions between sounds. The *BeatBot* was not designed to respond to any input and so drove the interaction, with the other two systems reacting. Thus, although very simple and asymmetrical as an ensemble, the musical output was nevertheless coupled. Since the *BeatBot* is not limited to regular 4/4 metre, it creates dubious non-corporeal beats to which both *DeciderBot* and *_derivationsBot* respond in esoteric fashions. In addition, *BeatBot* kills itself once its structure is complete, and the other two audio-responsive musebots, lacking a consistent audio stream to which to react, tend to slowly expire, bringing an end to each ensemble composition.

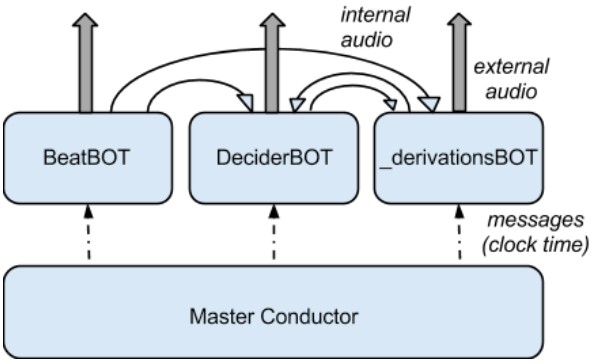

Figure 1. Audio and message routing in the second described musebot ensemble.

Example interactions between musebots, including this second example, are available online[3].

Issues and Questions

These studies give insights into how a musebot approach can serve innovation in musical metacreation. Two areas of interest are: (1) what can we learn by dividing up musically metacreative systems into agents and thinking about how the communication between these serves musical goals? (2) related to this, how do we work with others and negotiate the system design challenges?

1. Increasingly, musicians are incorporating generative music processes into their work. Thus, the situation described above — managing several generative interacting processes — is not uncommon. The creative process is different to traditional electronic music composition because rather than making a specific change and listening to a specific effect that results from that change, one is in a state of continuous listening, as the result of a change might have multiple effects or take time to play out. It is common for electronic music composers to work with complex systems of feedback, and this process is similar, if more algorithmic. One effect of this is that it can dull decision making, as one gives over to the nature of the systems, or is unclear on what modifications will influence them effectively. Placing these musebots together in an ensemble positions us as both curator and designer: in the former case, one is forced to decide whether the musebots are interacting in a fashion that is considered interesting, and whether fewer, or more, musebots would solve any musical issues. We foresee such decisions to be more common as we accu-

[3] http://metacreation.net/musebot-video/

mulate more musebots, particularly those with clear stylistic bents. In the latter case, as designers we are placed into a more traditional role, in which continual iteration between coding, listening, critiquing, re-designing, and coding again guides both technical and aesthetic decisions. While we have no control over the other musebots, we can individually control how our own musebot reacts to other musebot actions, even if those actions are seemingly unpredictable.

2. Working together in this way offers a new approach to musical metacreation, along with a new set of challenges. In building systems, we are typically free to pursue our own aesthetic directions, and make individual decisions, both technical and aesthetic, as to how these systems should act and react. In the case of *BeatBot*, such a "closed system" is maintained, albeit with the addition of transmitting messages regarding its current state. In the case of *DeciderBot* and *_derivationsBot*, these existing systems had previously interacted with human musicians, and could rely upon the performer's intuitive musical responses to enhance those decisions made computationally. Within the musebot ensemble, both systems are now reacting to other machines: one that is essentially indifferent, and another whose reactions had previously been keyed to human actions.

As is often the case in experimental music production, having set up the interaction between agents and listening to how this interaction unfolds, we found clearly musically interesting content in this first attempt at a musebot ensemble. We anticipate many more musebots being designed and contributed, and imagine that through the unexpected combinations of such autonomous music-generating systems new thinking about automating musical creativity, and making it available to a wide community of users, might arise.

The current work is a small affirmation of the potential of a musebot approach, and several questions have arisen regarding the next stage of development. Our next step is to curate a number of musebots to be presented in an ongoing installation of interchangeable ensembles across different genres. In order to reach such a stage of development, the following questions need to be addressed:

What kinds of interaction are useful – both computationally and musically? At the moment, the three musebots are not sharing any information in the form of network messages. Firstly, the *BeatBot* is generating beats, entirely unaware of any reactions to its audio, and while the two responsive audio musebots generate emergent musical material driven by audio analysis, they are oblivious to any structural decisions being made by the rest of the ensemble due to their lack of messaging. While such independence is one aesthetic solution, a more responsive and self-aware environment will need to be explored, if for no other reason than structural variety. In the present ensemble, one approach could be to augment the capabilities of *DeciderBOT* and *_derivationsBOT* to allow network messages from *BeatBOT* to have an affect on their internal generative capabilities, such as levels of density and musical timing. Alternatively, an augmentation of *BeatBOT*'s capabilities as a producer could enable it to direct high-level changes in state in each of the connected bots, a possibility anticipated in the musebot specification by the availability of the statechange message.

What is the minimum amount of information necessary to be shared between Bots to have a musical interaction? A next step is determining the kind of information that should be shared between musebots. The MC is generating a constant click, which affords an acceptance over a common pulse: how that pulse is organised in time (i.e. the metre) is a basic parameter of which each musebot should be aware. However, where should this be determined? Sharing of pitch information is also natural, but should an underlying method of pitch organisation also be shared (i.e. a harmonic pattern)? What happens when conflicting information is generated? Lastly, how should form be determined? An accepted paradigm of improvised music is the evolutionary form produced by self-organisation resulting from autonomous agents (human or computer); however, EDM tends to display a more rigorous structure. How should this be determined?

What relationship to human composition and performance should be incorporated? Within the MuMe community, research has been undertaken to model human interaction within an improvisational ensemble of human performers (Blackwell *et al.* 2012). We suggest that musebots are not merely a "robot jam". To quote from the Call for Participation, "'human musicians having a jam' can make for a useful metaphor, but computers can do things differently, so we prefer not to fixate on that metaphor. Either way, getting software agents to work together requires thinking about how music is constructed, and working out shared paradigms for its automation."

What aspects of the interaction can go beyond human performance modeling? A great deal of what humans do in performance has been extremely difficult, if not impossible, to model. For example, simply tracking a beat is something we assume any musician can do with 100% accuracy, while computers are seldom better than 90% at this task. However, there are limits to human interaction, which computers can potentially overcome. For example, computers can share and negotiate plans, and thus exhibit a collective telepathic series of intentions. Young and Bown (2010) have offered some interesting possibilities for interaction between agents that could certainly be explored between musebots.

What role should stylistic and aesthetic concerns play in formulating ensembles? We imagine that in the future, musebots can query one another as to their stylistic proclivity, and generate interesting and unforeseen ensembles on their own. At the moment, the notion of human curation is still necessary. With only three musebots, the variety of musical output is obviously limited, but we imagine musebots being designed to produce specific stylistic traits. A

related question is how the musebots can, or should, deal with expectation: certain styles of EDM exhibit certain expectations in the listeners; while we acknowledge that we are not constrained to existing stylistic limitations, we are expecting humans to listen to, and hopefully appreciate, the generated music. Ignoring musical expectation outright is perhaps not the best strategy when offering a new paradigm in music-making.

What steps would we need to take to make this a more intelligent system of interaction and/or coordination? Many existing MuMe systems have already demonstrated musical intelligence in their abilities to self-organise, execute plans, and react appropriately to novel situations. However, the designers often rely upon ad hoc methodologies to produce idiosyncratic, non-idiomatic systems. How can such systems communicate their internal states efficiently, or is this even necessary?

What are the emerging decisions that we would make about messaging? How could we categorise these and generalise them? While audio analysis is one possible method for musebots to determine their environment, relying upon such analyses alone would take up huge amounts of processing cycles, without any guarantee as to an accurate cognitive conception of what is actually going on musically. Furthermore, given that each musebot would require its own complex audio processing module, the hardware demands would be inordinate. For this reason, having musebots simply tell other musebots what they are doing through messages seems much more efficient. However, how much information does a musebot need to broadcast about its current, or possibly future, state, in order for other musebots to interact with it musically?

What is the furthest we could get with just "in the moment?" From the above discussion, it is clear that an important concern for musebot ensembles is addressing the tensions that exist between self-organised generativity and coordinated, hierarchical musical structures. Clearly, 'in the moment' generation of musical materials is a trivial task for complex musical automata like the musebots described in this paper. A balance between autonomy on the one hand, and controlled, structural decisions will need to be carefully considered in the design of both musebots themselves, and their curation into musical ensembles. Ultimately, curatorial decisions surrounding style and musical aesthetic also go hand in hand with concerns regarding determinacy/indeterminacy in musical composition and performance, and we are excited to see how this ongoing tension will influence musebot designers and curators into the future.

Conclusion

A primary goal in developing the musebot and musebot ensemble is to facilitate the exchange of ideas regarding how developers of musical metacreative systems can begin to collaborate, rather than continue to build individual idiosyncratic, non-idiomatic systems that rely upon ad hoc decisions. As we are targeting existing developers of MuMe and interactive systems, we recognize the variety of languages, tools, and approaches that are currently being used, and the reticence at adopting new frameworks that might inhibit established working methods. As such, our goal is to make the specification as easy as possible to wrap around new and existing systems and/or agents.

The specification uses a standard messaging system that can be incorporated within almost any language; however, we purposefully have not specified the messages themselves. Our intention is for these messages to evolve naturally, in response to the musical needs of developers. For example, through the use of machine- and human-readable info files, musebots and musebot developers can determine the messages a specific musebot receives and sends, while the open source specification allows for developers to propose new messages. Once these agents are performing together at a basic level, we feel that a community discussion will begin on the type of information that could, and should, be shared.

We have presented a description of our successful, albeit limited, first implementation of what we feel is an extremely exciting new paradigm for musical metacreation. Complex, autonomous musical producing systems are being presented successfully in concert, and the musebot platform is a viable method for these practitioners to collaborate creatively.

References

Assayag, G., Bloch, G., Chemillier, M., Cont, A., and Dubnov, S. 2006. Omax brothers: a dynamic topology of agents for improvisation learning. In *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, 125–132.

Blackwell, T., and Young, M. 2004. Self-organised music. In *Organised Sound*, 9(02): 123-126.

Blackwell, T., and Young, M. 2005. Live algorithms. In *Artificial Intelligence and Simulation of Behaviour Quarterly*, 122(7): 123.

Blackwell, T., Bown, O., and Young, M. 2012. Live Algorithms: towards autonomous computer improvisers. In *Computers and Creativity*, 147–174, Springer Berlin Heidelberg.

Bown, O., and Martin, A. 2012. Autonomy in music- generating systems. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference,* Palo Alto.

Bown, O., Eigenfeldt, A., Pasquier, P., Martin, A., and Carey, B. 2013. The Musical Metacreation Weekend: Challenges Arising from the Live Presentation of Musically Metacreative Systems. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*, Boston.

Bown, O., Gemeinboeck, P., and Saunders, R. 2014. The Machine as Autonomous Performer. In *Interactive Experience in the Digital Age*, 75–90, Springer International Publishing.

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 2015*, Vancouver.

Carey, B. 2012. Designing for Cumulative Interactivity: The _derivations System. In *12th International Conference on New Interfaces for Musical Expression*, Ann Arbor.

Charnley, J., Colton, S., and Llano, M. 2014. The FloWr Framework: Automated Flowchart Construction, Optimisation, Alteration for Creative Systems, in *Proceedings of the Fifth International Conference on Computational Creativity*, 315–323, Ljubljana.

Collins, N., and McLean, A. 2014. Algorave: A survey of the history, aesthetics and technology of live performance of algorithmic electronic dance music. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, London.

Dahlstedt, P., and McBurney, P. 2006. Musical agents: Toward Computer-Aided Music Composition Using Autonomous Software Agents. *Leonardo*, 39(5): 469–470.

Diakopoulos, D., Vallis, O., Hochenbaum, J., Murphy, J., and Kapur, A. 2009. 21st Century Electronica: MIR Techniques for Classification and Performance In *International Society for Music Information Retrieval Conference*, Kobe, 465–469.

Downie, S. 2008. The music information retrieval evaluation exchange (2005-2007): A window into music information retrieval research. In *Acoustical Science and Technology*, 29(4): 247–255.

Eigenfeldt, A. 2007. Drum Circle: Intelligent Agents in Max/MSP. In *Proceedings of the 2007 International Computer Music Conference,* Copenhagen.

Eigenfeldt, A., and Pasquier, P. 2013. Evolving structures for electronic dance music. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, Amsterdam, 319–326, ACM.

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 Artificial Intelligence and Interactive Digital Entertainment Conference*, Boston.

Eigenfeldt, A. 2014. Generating Structure – Towards Large-scale Formal Generation. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*, Raleigh.

Eldridge, A. 2007. *Collaborating with the behaving machine: simple adaptive dynamical systems for generative and interactive music*. PhD diss., University of Sussex.

Gimenes, M., Miranda, E., and Johnson, C. 2005. A Memetic Approach to the Evolution of Rhythms in a Society of Software Agents. In *Proceedings of the 10th Brazilian Symposium on Computer Music*, Belo Horizonte.

Lewis, G. 1999. Interacting with latter-day musical automata. In *Contemporary Music Review*, 18(3): 99–112.

Nierhaus, G. 2009. *Algorithmic composition: paradigms of automated music generation*, Springer Science & Business Media.

Rowe, R. 1992. Machine composing and listening with Cypher. In *Computer Music Journal*, 16(1): 43–63.

Rowe, R. 2004. *Machine musicianship*, MIT press.

Whitelaw, M. 2004. *Metacreation: art and artificial life*, MIT Press.

Wright, M. 1997. Open Sound Control-A New Protocol for Communicating with Sound Synthesizers. In *Proceedings of the 1997 International Computer Music Conference*, Thessaloniki, 101–104.

Yee-King, M. 2007. An automated music improviser using a genetic algorithm driven synthesis engine. In *Applications of Evolutionary Computing*, Springer Berlin Heidelberg, 567–576.

Young, M., and Bown, O. 2010. Clap-along: A negotiation strategy for creative musical interaction with computational systems. In *Proceedings of the International Conference on Computational Creativity*, Lisbon, Department of Informatics Engineering University of Coimbra.