Rapid Composition for a Multi-Device Networked Music Platform.

Poster · June 2020

CITATIONS
0

READS
56

4 authors, including:

Deborah Jane Turnbull Tillman
UNSW Sydney
10 PUBLICATIONS **20** CITATIONS
SEE PROFILE

Oliver Bown
UNSW Sydney
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Angelo Fraietta
21 PUBLICATIONS **47** CITATIONS
SEE PROFILE

Rapid Composition for a Multi-Device Networked Music Platform

Oliver Bown
Interactive Media Lab,
UNSW
o.bown@unsw.edu.au

Angelo Fraietta
Interactive Media Lab,
UNSW
a.fraietta@unsw.edu.au

Sam Ferguson
Creativity and Cognition Studios
Faculty of Engineering & IT
University of Technology Sydney
samuel.ferguson@uts.edu.au

Deborah Turnbull Tillman
Creative Robotics Lab,
UNSW
d.turnbull.tillman@unsw.edu.au

ABSTRACT

In this paper we discuss the results of a workshop study for the HappyBrackets system – a development framework for creatively coding multi-device musical performances, sound installations and interactive media artworks – in which new users using the system are invited to create new multi-device music compositions in a rapid creative and collaborative hacking session. We consider the types of works made, the problems encountered and the methods used, including how some of the new features we have added to the system support exploratory creative search. We develop our observations into design principles that we speculate will better support more rapid creative exploration of multi-device creative musical compositions.

1. INTRODUCTION

In this paper we discuss the results of a workshop study for the HappyBrackets system – a development framework for creatively coding multi-device musical performances, sound installations and interactive media artworks – in which new users using the system are invited to create new multi-device music compositions in a rapid creative and collaborative hacking session.

Our research interest lies in how creative coding tools, in this complex domain of multiple-networked devices, influence and support the production of creative work, given how laborious and slow it can often be to configure and code multiple devices in many different bespoke configurations. This is an area that has received recent attention through concepts such as the hackability of digital musical instruments (e.g., [1]) and the Internet of Musical Things [2], and more generally through networked music creation (e.g., [3]).

Participants with little or no previous experience of HappyBrackets, and varying levels of coding skill, were invited to prototype original works on a 25 device system in a collaborative and open workshop. We used this workshop as an opportunity to study what kinds of works the participants conceived of, how their design ideas were influenced by their effort to implement the design, what problems they encountered, how they went about solving problems, and how their time was divided up between different kinds of problem-solving and creative tasks. This study seeds a new direction in our research focused on adding a new API layer to HappyBrackets which will support conceptualising and hacking works for multiple devices, taking into account issues like spatial relationships, synchronisation of system state and communication.

This paper begins by giving an overview of the HappyBrackets system and the design considerations involved in some of the recent features we've added, which are focused on addressing creative productivity and flexibility. We then describe our workshop study, report results, and consider new principles arising from our observations.

Figure 1. The 25 device DIAD system in its original setup at the Powerhouse Museum. Devices are wired for power but communication takes place via WiFi.

2. HAPPYBRACKETS

HappyBrackets is a system that allows the live deployment of compiled programs from an integrated development environment (IDE) to multiple remote, networked client devices, which we refer to as distributed interactive audio devices (DIADs). Each remote client includes a system runtime that contains a real-time audio DSP framework, simplified network communication functionality, and a mechanism for synchronising all the devices through a virtual clock over the network. The associated plugin for the In-

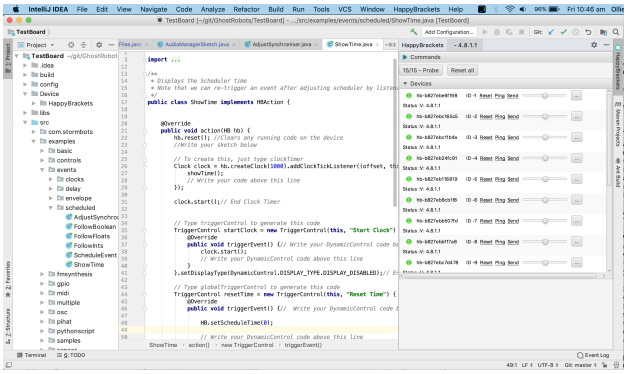

Figure 2. A HappyBrackets project running in the IntelliJ environment. The plugin window to the right shows a set of connected devices.

telliJ Java IDE (Figure 2) provides tools for monitoring and remote controlling devices, deploying programs in realtime to devices, rapidly setting up GUI controls for controlling specific programs running on devices, sending audio and data files to devices, and configuring devices. Although live coding performance is one possible use of this system, we primarily conceive of live coding in our work as a strategy and set of technologies that supports rapid development, experimentation and hands-on ideation.

2.1 Existing and New Features

HappyBrackets is described in a number of previous papers [4–6], and has been used in a variety of exhibited artworks [7–9] going back to an initial instantiation in 2013 (presented at the International Computer Music Conference 2013, Perth, Australia, and previously at the Tin Shed Spots in Sydney, Australia) [10], which as far as we know was the first instance of multiple Raspberry Pi units being used in live coded, networked music performance. HappyBrackets takes inspiration from the Processing creative coding environment, a key feature of which is the speed at which a computational design 'sketch' can be created. However, like another creative coding environment, SuperCollider, HappyBrackets works with a client-server model, with a runtime system on the client device responding to incoming sketches (or compositions) sent from a programming integrated development environment (IDE) running on what we call the 'controller' device (usually a laptop computer). HappyBrackets sketches are sets of instructions that are run on the client immediately upon being received from the controller. The code in a sketch can be used to dynamically create audio signal chains (or any event for that matter, but historically HappyBrackets has been audio-focused) and activate clocks and network listeners that in turn enable any number of other events to be triggered in the future. Multiple HappyBrackets sketches can be run on any client and can be programmed to interact with each other. It should be noted that although HappyBrackets can and has been used as an environment for live coding performance, our main interest in the 'liveness' of HappyBrackets is for the purpose of rapid prototyping and experimental creation, not performance.

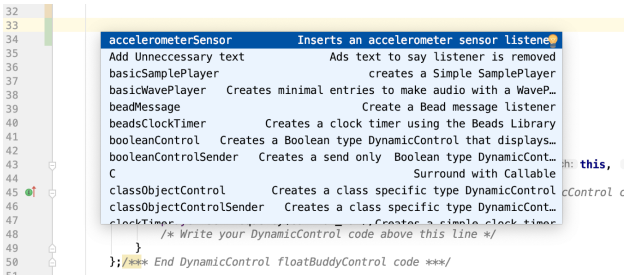

Figure 3. IntelliJ's list of "live templates", including those added by the HappyBrackets plugin.

Basic interaction can be programmed very easily. Setting up a sensor listener or network listener involves a few lines of code, and we make use of IntelliJ's "Live Templates" mechanism to support this. A Live Template is a keyword which triggers a generic code block to be generated. For example, an existing Live Template in IntelliJ is the keyword "sout" which generates the code:

```
System.out.println();
```

It also inserts the caret into the brackets. An example of one of our Live Templates is "accelerometerListener" which generates a block of code to respond to the accelerometer and places the caret at the start of the block so that users can immediately start writing the accelerometer action (for supported sensors, the HappyBrackets runtime will automatically find a connected sensor, meaning that configuration effort is minimised). Users can also look up Live Templates from a list (Figure 3).

At heart, HappyBrackets is built around the Java programming language. It uses the capability of Java to serialise compiled classes and load these dynamically. Code written on the controller device is compiled and sent over a network to the client devices, which ingest the Java class, identify it as a HappyBrackets action (HBAction), and run it. Whilst Java has a fraught history in the audio domain, particularly because of its memory management system (which can unexpectedly interrupt program execution, a problem which has largely been resolved with newer garbage collectors), we now find that we can run moderately complex audio programs on very low cost *Raspberry Pi Zero W* devices, at CD quality (stereo, 44.1kz, 16bit) with moderate latency (e.g., 1024 samples). With many very fast embedded audio systems in use, some very popular, such as Teensy, we have found that the Pi Zero is comparable on cost, reasonably strong although not superior in terms of form factor, and a somewhat inferior option in terms of power consumption, latency and startup time. We note that in many situations the latter negatives are not significant; this is largely the case in our work, except in interactive performance situations where latency and battery consumption is an issue. In such cases, we believe that the combination of a mature operating system and a rich, multi-threaded programming environment like Java, combined with the ease of deployment of HappyBrackets to multiple devices, is superior in supporting flexible creative coding development.

In addition, HappyBrackets is set up as an entirely decen-

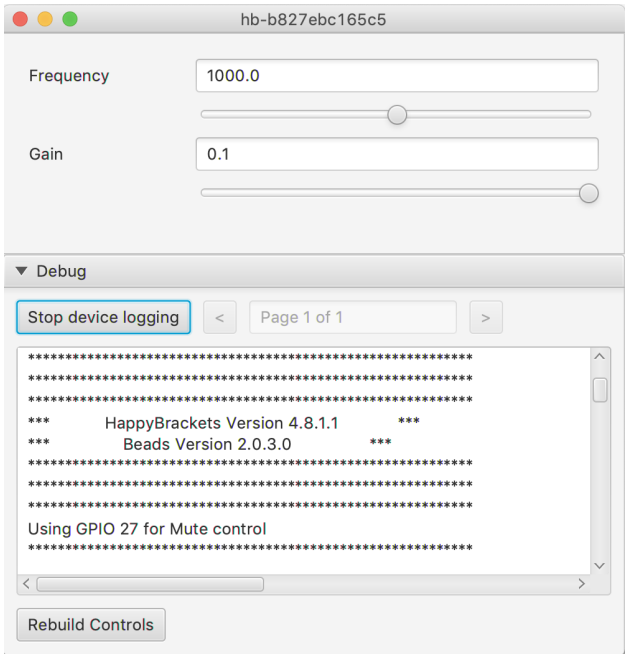

Figure 4. A device interface auto-generated from parameters specified in the sketch code, rendered on the controller computer.

tralised system where any number of clients can connect to any number of controller computers. Each connected controller has the ability to send sketches to, and control, each device. This has proven useful for collaborative work. Users can also set their controllers to operate only a specific subset of clients in situations where there may be potential confusion. They can also set their controller computer to operate *as* a client, which can be useful for testing when it is not convenient or possible to set up devices. Recent developments have been focused on making this last feature more robust, including more streamlined use of network traffic in complex multi-device, multi-user situations.

2.1.1 Newest Features

Several newer features have been reported in previous papers but remain more active areas of exploration. One key development, previously discussed in [11], is a system for sharing parameters between devices and controllers. A program may set up shared parameters in a sketch using any of the common numeric data types: booleans, integers and floats. These parameters are given names and ranges (where applicable), and on the controller computer, a GUI can automatically be generated that presents these parameters as sliders or number boxes (Figure 4). Two way binding between controller and device is automatically managed, although it is the user's responsibility to write the code to control how the device manipulates and responds to this parameter at its end. Parameters can also be scoped to be global, which means that all devices and all GUI instances of that parameter running on any controller computer will share that parameter (referenced by name).

Most recently, we have implemented a time synchronisation mechanism by which users can instruct all devices on a network to synchronise their internal clocks, and then write simple code to create timestamped events in the future, or to write beat-event code blocks that respond to a shared metronome, which will stay in time even in response to tempo changes. The performances of the time synchronisation is variable, but regularly manages to fall within around 20ms. We found that clock drift on the Pis was very minor, and that they could stay in synch for over 24 hours. We also found in practice that network broadcast messages would tend to arrive at multiple devices almost simultaneously (although without certainty, due to jitter in Java's program execution), and that, knowing this, users can choose between various alternative methods for synchronisation. Higher accuracy synchronisation is of interest to achieve specific acoustic effects, but is not a high priority, given our general philosophy of treating our networks of devices as independent sound objects, not as components in a spatial audio rendering system.

Lastly, we added additional functions into IntelliJ that allow users to easily send additional files besides live code sketches to one or all devices (also discussed in [11]). The main files that can be sent are audio files and additional classes and jar files (Java libraries), which are organised in a "data" folder on the device (as used in Processing sketches), and which can be mirrored in the file structure of the IntelliJ project. Users can configure their "data" folder however they like to manage other file types.

All of the above features were implemented by lead developer Angelo Fraietta, and have been in regular "beta" use by the authors in their own creative work. These features are presented in greater detail in [4]. In addition, we have engaged in more or less formalised user research into how tools such as these influence the creative process of working with multi-device audio systems.

3. STUDIES

Evaluation of creative artefacts developed within the domain of interactive art [12] can focus on aspects of experience such as creative engagement [13], play [14], or curation of digital art [15]. Our focus is on the productivity of the creator, but this necessarily takes into account the creative goals of their work. Supporting creative productivity can involve designs that speed up iterative cycles, reduce barriers to productivity such as the need to perform menial tasks (e.g., configuring systems and writing boilerplate code), and in particular, reduce cognitive load distractions that draw away from exploratory search. In creative software design research we draw on Shneiderman and Resnick's [16, 17] design principles for creativity support, in particular the idea of supporting low floors, wide walls and high ceilings. Related frameworks include Blackwell's cognitive dimensions of notations systems analysis [18, 19], which include attributes such as viscosity (resistance to change) and premature commitment, and Nielsen's heuristics for interaction design [20], which include factors such as consistency, visibility and cognitive load. In this paper, we build on a framework started in [21] where we categorise creative coders' activities, consider when and why they might get stuck in less creatively

exploratory activities, and also consider the overhead of switching between these activities. We add to this an attempt to categorise the types of works that people naturally conceive of with when faced with a multiplicitous media context, broadly imagined as 'design pattern', and consider how realisable such categories of work are given our current framework, and how we might bring the more complex patterns within reach. We also consider how collaboration occurs and can be supported in this context.

The study presented in this paper was conducted in an informal one-day workshop setting, in the style of a hackday, as this was essential to creating a situation in which participants could comfortably engage with our 25 device rig, with the training and support required to create work. The 25-device system, set up at UNSW's Interactive Media Lab, was originally built for an installation at the Powerhouse Museum in Sydney (Figure 1). The devices were hung from the ceiling at various heights just above headheight, and randomly scattered across two connected rooms.

From several previous workshops with people of different skill levels, such as a National Science Week workshop for kids [22], we have found informal creative activities to be important to creating a stimulating environment. In our research therefore we focused on qualitative results derived from observations and a post-event survey, which also included a small number of Likert-scale questions relating to the relative time cost of activities. We also collected participants' code at the end of the session. We considered it obstructive and costly to record detailed data of the participants usage, for example via video, self-reporting on the go, or input logging, since participants were moving around, discussing ideas and taking breaks throughout the day. Self-reporting at the end of the session is an imperfect way to gather data, but has the advantage of being efficient and unobtrusive. Reports were checked against observation throughout the day by the team. Recruits were primarily drawn from the pool of students and staff from UNSW. Although it is preferable to recruit without this constraint and proximity to participants, this was also a practical measure given the informal but in-depth nature of the workshop. We consider this an efficient precursor to more in-depth and arms-length studies, with the aim of seeding design concepts that can be tested thoroughly once implemented.

Participants were from a wide variety of coding backgrounds, but all self-reported some prior experience writing code. They were all from the surrounding academic community and research cohort at UNSW Art & Design and other local universities, at varying education levels (undergraduate to doctoral candidates and recipients). The gender split of male to female participants was 6:2. They were given a 2-hour introduction to using the HappyBrackets system and library, through a number of demonstrative examples. They were then encouraged to develop their own simple creative concepts and implement them, with 3 members of the HappyBrackets team supporting them. We report these new users' responses to working creatively with the system, including observation, interviews, a brief survey, and analysis of the code they wrote. In all, 8 participants responded to our survey and submitted examples of the code. We also discuss our own experiences as creative practitioners.

The following subsections organise responses according to the themes we are interested in. We intersperse further elaboration on our design thinking throughout this discussion of responses as it helps to clarify what the issues being raised are, as they come up.

3.1 Working with Multiples

Aside from specific usability issues, one of our key interests was how people take to working with a multi-device sonic system for which each device is individually programmable but there is no notion of spatial abstraction away from the devices themselves, *i.e.*, there is no pre-defined way to render sounds as abstract objects moving between the devices. As we mention above, we find the DIADs system invites a slightly different way of thinking about multi-speaker audio as compared to traditional multichannel systems. One participant commented specifically on how the DIADs paradigm directed their thinking: "I think that the modular nature of the system forces you to think in individual MODULES, which makes you more likely to design systems that scale. I could realize my project easily by having 8 speakers hooked up to my computer, but I wouldn't write it in a way that scales up to more units. i think that's GREAT" (P1). Another participant noted: "I learnt that it's incredibly important to think about the physicality of the system - asking whether the network can handle what you want to send through to it and how simplicity that is replayed throughout all of the speakers can generate a profound effect. The power of replication using networked systems can be just as effective as a technically complex program" (P3).

Participants developed a number of creative visions in the workshop. These included:

- a simple implementation of Tristan Perich's "sound wall" (P1);
- an exploration of temporal synch patterns between devices via simple click rhythms, including random clouds, random 16th-note quantised clouds, and unison 16th notes (P9);
- a concept to capture audio and have it disperse through the system like a distributed echo or reverb;
- a sound that ricochets around from one device to another (P10);
- "an addressable array of notes making up a pretty chord with an aim to push waves of activation through the array" (P5);
- and "a random sample player which selects a new sample at the end of the current sample." (P7)

We briefly consider how such concepts might be categorised in terms of how the code being deployed to devices needs to take into account the relationship between action taking place on each device. We assume a static configuration of devices. We propose the following categorisation (which is an adaptation of work initiated in [10], in response to these and other designs we have encountered):

Forests (term first used in our previous paper [10]): each

device plays the same sound, with some variation in parameters, or a different sound, but with no attempt to dictate spatial relations between the devices. The overall effect is a rich orchestration of sound that is manually or statistically determined. The devices may play in synch: although this is clearly a form of coordination between the devices it happens automatically and does not require the programmer to think about the relationship between devices.

Fields: Fields are like forests, except sonic parameters are dictated by the spatial locations of the devices, and an abstract 'parameter field' may be involved in determining the variation in these parameters. For example, devices at one end of the room may be quieter than at the other end of the room, but over time the volumes gradually alter as the field changes. Here the programmer needs to know the positions of devices. Additionally, there is clearly some potential to abstract away the need to think about actual devices and their positions.

Networks: the devices are considered to be connected by some network topology (such as nearest neighbours or a "scale free network"), which dictates how sound behaviours or parameters move around the system. Here typically the programmer would need to know the positions of devices and to build a network of relations between them. They are more likely to be concerned with thinking of devices individually. This may be more likely to correspond to certain spatial configurations too, such as grids and other regular geometric shapes.

Substrates: the devices render abstract sonic objects. For example, a sound moves across the space, gradually panning between different neighbouring devices, or, through this method, a specific sound scape is recreated. In this case there is a clear need for the programming environment to support abstraction away from individual devices, towards virtual objects.

In the very short time frame in which works were developed in this project, the majority of works can be seen to be based on the forest model, which places a minimal requirement on the programmer. One respondent commented that because the devices were randomly scattered in space, and not in any obvious formation, they were more inclined to think in terms of random variation and not specific spatial relationships. A grid might have promoted greater consideration of spatial parameters.

Some works required a field approach, which needn't be considered significantly more complex than the forest approach: it can be as simple as querying each device's position and using that position data in the determination of parameters. During the workshop, a manual map of device positions was drawn, and these were provided to participants as a Java `Map` object, which mapped device hostnames to 2D coordinates, that could be pasted into their sketches and used to query the position of any device. Specifically, each device could then query its own position because it could query its own hostname. More complex field-based systems might take network inputs communicating a field of data from which devices could extract parameters based on their location within the field. This could easily be supported with some general purpose code.

One project attempted to use a network configuration instead, in which devices would send and receive information from their immediate neighbours. In this case, the positions needed to be mapped into information about neighbourhood relations. No existing library code supported this process, so a great deal of the team's time was spent considering strategies for doing so. Again, library code could be added that would support querying neighbourhood relations, and as mentioned, in many situations we might expect the use of more regular spatial configurations for projects where this was the underlying concept.

We note that some of the concepts involved spatiality but were sufficiently loosely defined that they could be implemented either as fields, networks or substrates, requiring teams to then consider in greater detail what they expected of their designs and what the easiest paradigm was to get them to a satisfactory outcome. For example, the idea of recorded audio propagating through the system could be achieved in any of these ways, but added the additional complexity of thinking about how the recorded audio might be shared over the network.

3.2 Design Processes

In previous research [21] we considered how creative coders move between a series of activity categories:

- setting up devices: getting things set up and switched on;
- connecting to devices: ensuring all devices are responding on the network;
- in-depth programming: implementing some specific feature, with some thought given to the design of an algorithm;
- debugging and testing: making some design or feature work properly by iteratively modifying and examining its behaviour;
- creative programming: exploring a creative idea rapidly through code, for which the metaphor of sketching is sometimes used;
- playing with parameters and interaction: exploring various possibilities in a program using other means besides writing, such as tweaking specific variables or interacting with an interface;
- discussing and designing: stepping entirely away from the system and code and engaging in conceptual development of the work.

We assume that the last three of these items are where the creative action lies and time spent on these tasks should be maximised in relation to the earlier items on the list. We recognise that some in-depth programming may also be where the creative innovation of a project lies, as we expect projects to come up with original technical concepts, but we nevertheless expect that in many cases in-depth coding could be minimised significantly given powerful frameworks to create with.

Our survey asked participants to detail where they found themselves spending most time (Figure 5). We also observed participants and regularly checked in to discuss their projects with them.

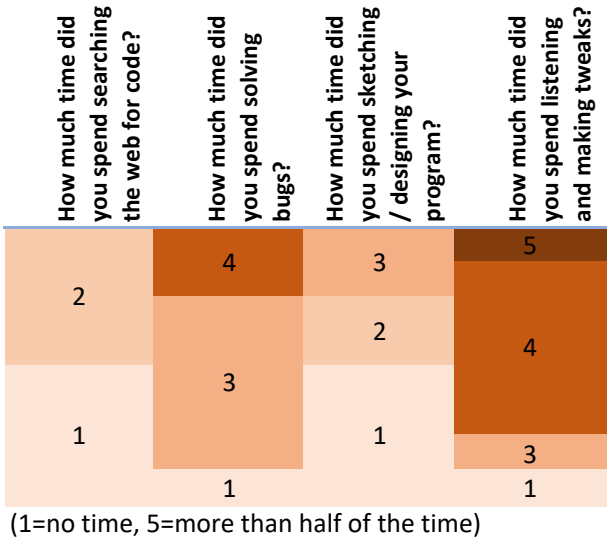

Figure 5. Survey responses to questions concerning how much time participants spent in certain activities.

We found that participants spent the bulk of their time split equally between fixing bugs (or more generally unknown issues), and spending time tweaking and listening; participants' projects were generally simple in scope and derived from a short tutorial introducing the capabilities of the system through examples, so programs were short and would not have taken a long time for an experienced coder to write, but understanding the system and the libraries through problem-solving was significant work for new users. Participants also did not report spending a lot of time looking online for solutions to bugs or problems, possibly because of a lack of faith they'd find answers (this being an 'in house' system), and also because they had experienced developers in the room; attempts to fix bugs and issues were more exploratory. One participant noted: "I'm in this place where I am advanced enough to code on my own and explore the examples and then adapt them to what I want to do. But I do not understand how the methods work from looking at the API. so I am completely dependent on the examples or people that know how to code in that environment" (P2).

As we hoped, a lot of time was spent listening and tweaking, which we consider tentatively to be a more creatively-oriented activity; still potentially associated with 'problem solving' their designs, just with a more aesthetic focus. Participants reported that they did not spend a lot of time sketching, discussing or otherwise engaging in design thinking away from the code. This is also as we might expect given the spirit of exploratory search and hacking from the examples that was established in the workshop. However, participants did clearly spend some time sketching and ideating in groups.

One participant noted that they struggled to apply concepts from one example to a similar area due to lack of consistency. Specifically they found that concepts in the WaveModule example (code to create a simple oscillator), did not clearly map to a similar SampleModule application (the same idea but using sample playback). Another mentioned that they couldn't easily understand the principles underlying device synchronisation in a way that enabled them to easily make a synchronised sample player.

In other comments about the ease of the workflow, one participant noted: "The sample player function is powerful and exciting for me at the moment, largely due to its ease of use. For someone that is not so great at coding, it was a simple framework that allowed me to get things going quickly. It means that I can focus more on the sound itself, I can design sounds at home using gear that I am familiar with and then bring them into the space and create something interesting with them with not too much effort" (P8). In other cases, there was clearly a greater need to provide easy entry points to the functionality, with participants expressing difficulty finding out how to do things or look up functions, examples and templates.

Another participant was frustrated about: "saving my different sketches so they don't overwrite the examples" and "file management in general" (P1). We suspect this would be less of an issue for someone experienced with advanced IDEs like IntelliJ, but acknowledge that depending on an advanced IDE may place a greater strain on the user getting started. Another participant added that "a lighter weight (maybe purpose built) IDE and not needing to install java would be nice" (P5).

3.3 Managing Parameters and Getting Feedback from Devices

The parameter control feature described above was demonstrated to participants in depth, illustrating how GUI sliders could be created and used to control parameters on individual devices or simultaneously across a number of devices. Our hope was that participants would find ways to use these parameters both as performative controls and as ways to explore parameter possibilities. Indeed, three survey respondents indicated that they used this feature in three different ways: in one case, in order to tweak parameters to seek a sweet spot, in another case to be able to manually trigger simultaneous playback at a specific moment, and in a final case as a user control to control sound density. We feel that this ability to both code and dynamically manipulate parameters is emerging in our designs as an important creative-coding principle, drawing on related work in this field such as the concepts of Bret Victor [23] and the kinds of tight GUI-code integration seen in visual-programming environments like MaxMSP, TouchDesigner and QuartzComposer.

Other comments related to the need to better visualise what was going on. Requests included: "A VU meter display that shows you when the different devices are playing and what their levels are" (P1); "Perhaps a way to see the flow of waves / uGen chain visually, like a summary map diagram" (P4); "Having a GUI based modular synth for beginners" (P7).

3.4 Collaboration

Allowing multiple users access to control the group of devices from different laptops was a conscious design feature in the sense that we wanted to avoid clunky or unclear

scenarios where two users both tried to connect their controller computers to the group. When we started running workshops with multiple users simultaneously deploying code from their own laptops we found that this feature was actually essential: the complexity of having a multiple wifi networks or establishing a system of binding between controllers and devices would have introduced additional complexity, and the idea of a group of people having to take turns accessing a controller computer was too inflexible. Initially, this multi-user experience was clunky, suffering from clogged networks and confusion around which devices were which, and what they were doing. As well as improving network communication and stability, which reduced the clunky experience, an additional feature that supported this scenario was an option to allow users to 'favourite' specific devices, and then limit interaction to this favourites list.

This multi-user scenario proved effective in certain modes of collaboration. Two coders could work independently and send sketches to the network. Given that individual devices allow two sketches to be run simultaneously, with precise control over when the state of the device is reset, it was feasible for two users to work together in a fluid 'detached' way, as if jamming together on the network. Participant feedback supported this view: one participant stated that "the collaborative potential of being able to overlay sketches is great" (P5). Other comments referred to the pleasure of being able to work in this shared space, hearing each others' work on the system during a hack session. This builds on a commonly held view that creative coding is enhanced by the speed of seeing or hearing responses, and we may speculate that being able to do so in a shared environment of experimentation can also support innovative and exploratory concept development.

However, this framework did also cause some confusion. One participant suggested a need for "being able to collaborate live more effectively by easily targeting and killing specific active sketches." They added: "I found I'd have to stop audio entirely and start from scratch building something up if there was one element I wanted to change"(P5).

4. CONCLUSION

Our observations reinforce some general design principles that have been in development in our own practice-based research for some time, and that relate to other work into creative computing research. We consider three broad conclusions here:

Multiplicities paradigms and time burden: In the creative space of media multiplicities, there are certain go-to system architectures, and some are more easily accessible than others. Forest architectures, those where multiple devices behave in similar ways but needn't be coordinated, are trivial to program, but still support a great wealth of creative possibilities. Field, network and substrate architectures all benefit from more framework support that we do not currently offer, but could easily do. Without such framework support users are forced to engage in in-depth coding, a distraction from creative productivity. Libraries to support these paradigms would need to be clearly outlined, first conceptually, and then via the API and its affordances. Learning from examples, and innovating new works from the starting point of existing 'proof-of-concept' examples is a powerful way to learn and produce, the latter being considerably lower-risk and requiring less experimentation and what we have called 'in-depth coding' (framework coding). We suggest that developing these paradigms independently, without too much concern about seeking a single overarching framework that covers them all, will be necessary. This might be analogous, say, to the paradigms of vector and bitmap graphics.

Collaboration: A multi-user system supporting simultaneous running of multiple sketches was seen to be creatively productive, supporting collaborative ideation. But to be really effective such a system clearly requires richer feedback so that users are aware of the state of the system, and possibly a clearer conceptual grounding for what users might expect from the state of the system. Such issues are also present in server-based live coding environments like SuperCollider. Generally this is a hard problem to solve and the user is typically expected to master practices that help maintain their mental model of the system without perfect system visibility. An example, as proposed by one of our respondents, is to frequently stop everything and rebuild the system: not ideal, but practical.

Parameter manipulation, tweaking and extensibility: The free and flexible manipulation of parameters at different stages of the creative design process was shown to be creatively empowering, but needs to be more rigorously and consistently supported. This poses design challenges associated with moving between code and "direct manipulation" interfaces without adding some degree of complexity to the process, which may be counter-productive. Contemporary GUI design frameworks (as used in Apple's XCode or Google's Android Studio) are a source of inspiration for how such integration can take place, as are conceptual demonstrations such as Bret Victor's designs [23]. Supporting the natural creation of two-way bindings and simple arithmetic combinations of parameters are examples of ways this more creatively productive approach to parameters could work.

Whether in the HappyBrackets framework or elsewhere, developing clear creative coding architectures and principles such as these, for the composition of media multiplicities, will, we believe, have a great impact on creative productivity in this emerging field.

Acknowledgments

We acknowledge the support of the Australian Research Council through their Linkage Grant (LP180100151), and our project partners BitScope Designs, ArtworksRActive (Linda Candy), Squidsoup, and the Casula Powerhouse Arts Centre. We also acknowledge the support of the Museum of Applied Arts and Sciences in the creation of the 25 device system, originally developed for the Spiral artwork in 2019. We thank the anonymous participants for their participation in the research.

5. REFERENCES

[1] A. P. McPherson, A. Chamberlain, A. Hazzard, S. McGrath, and S. Benford, "Designing for exploratory play with a hackable digital musical instrument," in *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*. ACM, 2016, pp. 1233–1245.

[2] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, "Internet of musical things: Vision and challenges," *IEEE Access*, vol. 6, pp. 61 994–62 017, 2018.

[3] L. Gabrielli and S. Squartini, "Wireless networked music performance," in *Wireless Networked Music Performance*. Springer, 2016, pp. 53–92.

[4] A. Fraietta, O. Bown, S. Ferguson, S. Gillespie, and L. Bray, "Rapid composition for networked devices in the internet of things," *Computer Music Journal*, vol. 43, no. 2, p. forthcoming, 2020.

[5] O. Bown, L. Loke, S. Ferguson, and D. Reinhardt, "Distributed interactive audio devices: Creative strategies and audience responses to novel musical interaction scenarios," in *Proceedings of the 2015 International Symposium on Electronic Art, Vancouver, Canada*, 2015.

[6] O. Bown and S. Ferguson, "Creative media+ the internet of things= media multiplicities," *Leonardo*, no. Early Access, pp. 53–54, 2017.

[7] W. Marynowsky, S. Ferguson, A. Fraietta, and O. Bown, ""the ghosts of roller disco", a choreographed, interactive performance for robotic roller skates," in *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction*, ser. TEI '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 631–637. [Online]. Available: https://doi.org/10.1145/3374920.3375284

[8] L. Loke, O. Bown, S. Ferguson, L. Bray, A. Fraietta, and K. Packham, "Your move sounds so predictable!" in *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, 2018, pp. 121–125.

[9] O. Bown and S. Ferguson, "A musical game of bowls using the diads," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2016, pp. 371–372.

[10] ——, "Understanding media multiplicities," *Entertainment Computing*, vol. 25, pp. 62–70, 2018.

[11] A. Fraietta, "Transient relics: Temporal tangents to an ancient virtual pilgrimage," in *Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction*, ser. TEI '20. New York, NY, USA: ACM, 2020.

[12] E. A. Edmonds, A. Weakley, L. Candy, M. Fell, R. Knott, and S. Pauletto, "The studio as laboratory: combining creative practice and digital technology research," *International Journal of Human-Computer Studies*, vol. 63, no. 4-5, pp. 452–481, 2005.

[13] Z. Bilda, E. Edmonds, and L. Candy, "Designing for creative engagement," *Design Studies*, vol. 29, no. 6, pp. 525–540, 2008.

[14] B. Costello, "A pleasure framework," *Leonardo*, vol. 40, no. 4, pp. 370–371, 2007.

[15] D. Turnbull and M. Connell, "Curating digital public art," in *Interactive Experience in the Digital Age*. Springer, 2014, pp. 221–241.

[16] B. Shneiderman, G. Fischer, M. Czerwinski, M. Resnick, B. Myers, L. Candy, E. Edmonds, M. Eisenberg, E. Giaccardi, T. Hewett *et al.*, "Creativity support tools: Report from a US national science foundation sponsored workshop," *International Journal of Human-Computer Interaction*, vol. 20, no. 2, pp. 61–77, 2006.

[17] M. Resnick, B. Myers, K. Nakakoji, B. Shneiderman, R. Pausch, T. Selker, and M. Eisenberg, "Design principles for tools to support creative thinking," in *National Science Foundation workshop on Creativity Support Tools*, Washington DC, 2005.

[18] A. Blackwell and T. Green, "Notational systems–the cognitive dimensions of notations framework," *HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science. Morgan Kaufmann*, 2003.

[19] L. Church, M. Marasoiu, and A. Blackwell, "Sintr: Experimenting with liveness at scale," in *Proceedings of ECOOP 2016*, 2016.

[20] J. Nielsen, "10 usability heuristics for user interface design," *Nielsen Norman Group*, vol. 1, no. 1, 1995.

[21] O. Bown, A. Fraietta, S. Ferguson, L. Loke, and L. Bray, "Facilitating creative exploratory search with multiple networked audio devices using happybrackets," in *Proceedings of New Interfaces for Musical Expression (NIME2019).*, 2019.

[22] A. Fraietta and O. Bown, "Creating a sonified spacecraft game using HappyBrackets and Stellarium," in *Procedings of the 17th Linux Audio Conference (LAC-19).* CCRMA, Stanford University, USA, 2019, pp. 1–7.

[23] B. Victor, "Inventing on principle," Video re-posted by Wired Magazine, February 2012.