Supporting Creative Practice in Wireless Distributed Sound Installations Given Technical Constraints

Author:
Bown, O; Ferguson, S; Dos Santos, ADP; Mikolajczyk, K

Publication details:
AES: Journal of the Audio Engineering Society
v. 69
Chapter No. 10
pp. 757 - 767
1549-4950 (ISSN)

Publication Date:
2021-10-01

Publisher DOI:
https://doi.org/10.17743/jaes.2021.0039

Supporting Creative Practice in Wireless Distributed Sound Installations given Technical Constraints

Oliver Bown[1], Sam Ferguson[2], Augusto Dias Pereira dos Santos[1], Kurt Mikolajczyk[2]
(o.bown@unsw.edu.au) (samuel.ferguson@uts.edu.au) (augusto.dias@unsw.edu.au) (kurt.mikolajczyk@uts.edu.au)

[1] *UNSW Art and Design, UNSW, Sydney, Australia*
[2] *Creativity and Cognition Studios, University of Technology Sydney, Sydney Australia*

In this paper we present creative practice-led research into building large, scalable ”multiplicitous media” artworks in which many networked devices control lights and speakers and are coordinated over WiFi to create holistic artistic and environmental experiences. We discuss competing constraints, in particular the creative constraints associated with the challenge of coding complex multi-device behaviours, maximising creative freedom and simplifying complex engineering and design decisions. Based on recent experience building multi-device digital installation works we propose an approach, the ”broadcast first recipe” that aims to simplify the space of creative possibilities, with a trade off between expressive power and creative efficiency that we argue is worth adopting. We examine this approach in light of hard technical constraints such as CPU and WiFi bandwidth budgets which we discuss in a concrete example. We consider how the effectiveness of the proposed approach could be further leveraged in the provision of support tools.

0 INTRODUCTION

We use the term “media multiplicities” to refer to networked systems of lights, speakers, screens and other actuators or sensors that work together to create rich, immersive media experiences [1]. With the evolution of physical computing technology, we are increasingly seeing buildings and public spaces adorned with complex, custom, site-specific media installations. This emerging domain of creative production poses not only technical challenges but challenges related to the ease of creative production, and related issues of user experience design for creative producers, leading to a complex interplay of constraints. The evolution of technologies related to the “internet of things” (IoT) is particularly transformative [2], with the emergence of very low cost, computationally powerful networked devices that opens up the creation of custom multiplicitous media installations to a wider base of creators in a wider set of contexts.

Fig. 1. The Mind At Work installation at the Casula Powerhouse Arts Centre, Sydney, 2020-2021. The system consists of 200 lights and 100 speakers controlled by 50 Raspberry Pi computers. Video link.

This paper reports on our creative practice-based and design research into building IoT powered sound and light media multiplicities. Specifically, it looks at the question of successful software architectures for the practical achievement of specific creative goals, given a complex of constraints: device processor power; overall network bandwidth, latency and stability; device-specific bandwidth, latency and stability; processor power of the controller computer; and, most important to us, the convenience and effectiveness of creative workflows for coding, composing and otherwise creating media content. We report on a real case study building a specific multiplicitous media artwork, and combine our own practice-based observations and de-

sign thinking with specific processor and network performance experiences that support our evaluation.

The outcome of this paper takes the form of a recommended general purpose architecture and creative practice approach, the broadcast first recipe, which we argue maximises creative freedom and simplicity, whilst satisfying other constraints reasonably. We make a case for this architecture and approach drawing on evidence from our practice-based observations.

0.1 Related Work

Systems of multiple media devices have been investigated by various researchers [3, 4, 5, 6, 7, 8]. Media multiplicities has been defined by Bown and Ferguson by describing the characteristics shown in various systems and artworks [1]. Other authors have built a variety systems composed of individual computational devices with sensing, actuation and computation [9, 10]. Coelho et al. developed [11] a system whereby each light-emitting device was independent of the other devices, but communicated colour instructions between the devices through the body of the participants interacting with the system. Media Facades [12, 13] are another area of related research, where usually light-based elements are brought together to become a contiguous substrate as part of the facade of an architectural envelope. Squidsoup's work [14] has explored the creation of spaces through the individual control of grid-like systems of lights for many years. Their continued systems development has moved from LED voxel grids controlled by a single central computer [15], to groups of many wifi-networked independent computing devices that were controlled over a wireless network [16, 17].

Programming media multiplicities is complex given the range of architectural options available to a media designer presented with a system of this nature. At the more forward-looking extreme, Vallgarda et al. [18] discusses *Material Programming*, whereby computation, sensing, and output, are all embedded within the physical fabric of a material. More practically, Bown et al. [19] investigated the use of a multiplicitous media system in the context of novice programmers and media artists, while Fraietta et al. [20] aim to describe approaches to sharing data between devices which improve on existing networking approaches, and methods for programmatically composing for systems built from programmable devices [21].

1 PHYSICAL ARCHITECTURE

Although the research topic of this paper could warrant lengthy discussion about possible alternative architectures for implementing media multiplicities, we will keep such discussion to a minimum in the interests of brevity. We focus the advantages and motivations for choosing the specific architecture that was developed for the case study work, "The Mind At Work" (Figure 1), first presented at the Casula Powerhouse Arts Centre, Sydney, Australia, in November 2020. This work was developed by the Interactive Media Lab at UNSW Sydney, in collaboration with Bitscope Designs, the Creativity & Cognition Studios at University of Technology Sydney, and with consultation from Squidsoup and ArtworksRActive in the UK.

Our system uses the Raspberry Pi computer, specifically the very low cost, small form-factor Raspberry Pi Zero W. Since its arrival, we have been attracted to working with the Pi and related low-cost, full-OS, general purpose computers. We consider these to be preferable in certain ways to microprocessor systems such as Arduino, primarily due to their flexibility, ease of programming and the out-of-the-box capabilities provided by the default Linux operating system (on the flipside, microprocessors can be cheaper, have lower power usage and faster startup times and can have greater realtime predictability). A single Raspberry Pi is capable of delivering multi-channel audio, although limited by its relatively low CPU performance, and can control multiple LEDs and other actuators via GPIO connections. Control of sound and light can be delivered via any programming framework that is supported by Linux and has the required IO libraries. Another advantage we see of working with general purpose computers instead of microprocessors is the ubiquity and expected longevity of the Linux OS. We expect our programs to work on new products from Raspberry Pi and other manufacturers for years to come with only minor configuration tweaks. These same programs also run without modification on Mac and Windows computers as well.

Over several years we have developed the HappyBrackets Java programming framework (henceforth HB) [21, 19], consisting of a realtime audio library, networking tools, IO tools and, most importantly, a means to deploy and run code "sketches" on the fly to multiple devices over a network. Supporting this, we have developed a custom HB plugin for IntelliJ, a leading professional integrated development environment (IDE) for Java, along with a custom disk image consisting of the default Raspbian Linux OS, augmented with our own scripts and a Java runtime that pre-configures a Raspberry Pi to run HB. Such devices boot up, join a pre-specified network and identify themselves to the IntelliJ plugin, ready to receive commands.

Additionally, Bitscope Designs have worked with us to develop a custom HAT, the Biotica board, for the Raspberry Pi providing a stereo 3W amplified audio output, and four data connections for driving LEDs. Communication with the HAT takes place over an audio driver for sound and a serial interface for controlling the LEDs. Bitscope's design shifts some of the processing work to the HAT, with a concept of programmable virtual circuits. It also allows for a modular architecture in which a Biotica master can control multiple separately powered Biotica nodes over a serial connection (to date, we have not used this feature).

Although previous successful work by Squidsoup in this domain has focused on an integrated media pixel type unit (one microcontroller, one compound LED and one speaker, all in the same housing) [16, 17], we have made units consisting of one Raspberry Pi Zero with Biotica HAT controlling two physically separate speakers and four separate LEDs. The integrated solution is conceptually simpler and can be cost effective with cheaper microprocessors, but a

cost and power effective Raspberry Pi-based design necessitates a higher ratio of actuators to control units. There is also the question, both aesthetically and practically, of whether it is better to co-locate speakers and lights into single objects. Single light-speaker units are conceptually more 'readable' by audiences, while hybrids of speakers and LEDs may be considered more messy and disconnected. A non-integrated design also adds some programming complexity, as we will discuss. However, integrating these elements places additional constraints on speaker and LED housing design that may disadvantage other criteria such as sound quality. In our current work we have kept LEDs and speakers as separate devices, partly because we are working with a 2:1 ratio between LEDs and speakers which necessitates at least a standalone LED option. We expect that a 4:1 ratio would work just as well and be even more cost and power efficient. Separating the actuators from the processors also results in more complex wiring, with potentially long cables running to LEDs and speakers that can introduce failures. This was a concern during development but turned out to be unproblematic with cable lengths tested up to 10m.

In summary, we have a system in which an array of Raspberry Pis can be controlled over a network with each device, in turn, controlling a specific subarray of LEDs and speakers. We claim that this is an effective design for many distributed sound installation contexts because: (a) working with single board computers instead of microprocessors enables (primarily) greater coding flexibility, and often higher bandwidth WiFi and greater processing power; (b) due to cost, the ratio of processors to media outputs (lights and speakers) must be greater than 1:1, and therefore should be physically distributed (this is also beneficial to reduce network demands, and may well apply in the case of micro-processors too); (c) although more complicated in terms of physical installation, this one-to-many configuration increases flexibility of form, with the possibility to connect different light and speaker units to a core reusable unit.

We further assume that there is also one central device included in such an architecture, a controlling computer, typically something more powerful than a Raspberry Pi (a Mac Mini in our case), that runs a program to send control messages to the other devices.

2 SOFTWARE ARCHITECTURES

Although hardware and software design choices are interdependent, we have found that the above hardware architecture choices has been relatively uninfluenced by software design issues, arising largely from basic cost and physical design considerations given the decision to work with single board computers rather than microprocessors.

We now consider the possible software architectures (Figure 2), given this hardware arrangement. The prominent question in terms of a software architecture is how to distribute the various computational processes given the choice of the various control units and the controller computer. In response to this essential question we propose that

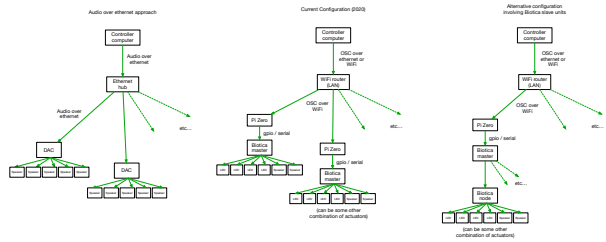


Fig. 2. Different basic architectures for media multiplicities. Left: Audio over ethernet driving distinct audio channels from a central computer to different speakers. Centre: A distributed architecture sending WiFi control messages to devices which generate audio. Right: An alternative federated version (not explored in this paper) in which each main device controls a number of additional devices down the chain.

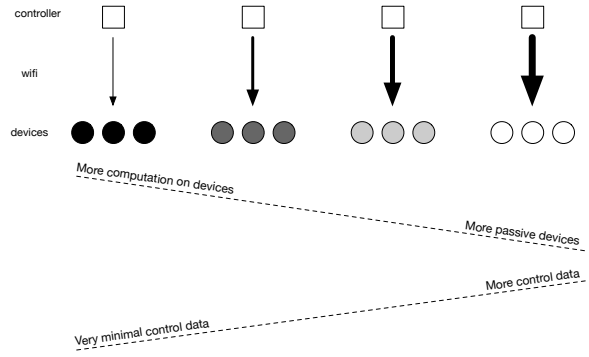


Fig. 3. A trade off between control bandwidth overheads and work done on devices.

software architectures can be loosely arranged along a continuum based on how active or passive the devices are (and conversely how much bandwidth is used to communicate with them) (see Figure 3):

1. (Active device extreme). Zero communication between devices. Each device runs its own separate program and most or all of the computation is performed on devices. The controller computer is redundant.
2. Devices run various models relating to the global behaviour of the system, controlled occasionally by global state variables (e.g., every 2 minutes).
3. Devices act as "synthesisers" controlled by frequent low-level control data (e.g., every 50ms).
4. (Passive device extreme). Devices are simple near-stateless receivers for streaming waveforms and RGB values.

The hardware architecture outlined above is cost and power efficient but introduces complexity in terms of the programming of devices (as well as in terms of the physical build). As creative practitioners our goal is to create spatial sculptures of light and sound and program those structures as if they are unified entities, just as one may make images

appear on a screen without thinking about how to program individual pixels.

At one extreme (we will refer to as the "passive device extreme"), each device is a node, like a pixel, that receives regular state updates that it directly and instantaneously renders. There is minimal processing demand on the device but a high bandwidth demand on the network, as a vast number of individual messages need to be sent from a controller machine to each individual device. With sound, in the extreme, this means streaming audio at thousands of samples per second to each individual device, which has a massive bandwidth requirement. For LEDs, this may require tens of RGB values per second and is more feasible. The passive device extreme has the greatest and most poorly scalable bandwidth usage and becomes less viable in circumstances with many devices.

At the other extreme (the "active device extreme"), the bulk of computation and scheduling is done on devices with only the most minimal messaging over the network to keep devices synchronised. The network demands become negligible, but such a system is relatively limited in function: the potential to manage coordinated system behaviours, especially in realtime, is significantly reduced. Whilst such a system can create many types of beautiful synchronised effects, there is a vast space of effects it cannot achieve.

Between these extremes lie systems in which dedicated behaviours run on devices and are controlled over the network. Along that continuum we may imagine programs running on devices that behave like synthesisers (in the musical instrument sense), responding to incoming messages that vary from a relatively high "continuous control" rate (e.g., every 10-100ms), or slower. As with music synthesiser arpeggiators, such programs might include sequential behaviours that perform independent actions without requiring fine-grained control signals. Our design uses the concept of a "Renderer", a software object that performs rendering tasks (for LED lights, sound or other media) in response to incoming broadcast messages. A single renderer controls a single LED or speaker, and using our framework a correctly configured device will create one renderer instance for each actuator it controls, and then forward incoming control messages to its array of renderers.

From working with such systems, we note the factors of creative freedom, design simplicity, and realtime capability all favour software architectures that make the most use of a fine-grained control stream, nearer to the passive device extreme. The simple reason for this is that it most closely approximates working with a traditional timeline-based program, such as a digital audio, video or animation workstation. From an interaction design perspective such tools maximise the capacity for the direct manipulation of events and materials, and reduce the cognitive load and premature commitment of having to configure and maintain a mental model of a complex remote client program [22]. Assuming this to be the case, we can state our challenge as seeking specific designs, and more general design principles, for deciding how to assemble on-device programs and their relation to a central controller program sending commands over a network, that fall within the bandwidth constraints of the physical architecture, and are creatively productive.

One immediate consideration here is the practical difference between broadcast and unicast messaging in computer networking. Unicast messages target a single recipient whereas broadcast messages are received by all nodes on a given local network. Thus typically, though not always, designs that broadcast global states, rather than send specific device states to individual devices via unicast, will have lower bandwidth demands. A downside of broadcast may be that each individual device may have to filter out lots of unnecessary data. Another problem with broadcast is that it has quite a complex underlying implementation that can vary across networking devices with unexpected performance, as we have found frequently. Indeed counter-intuitively we have found cases where it is more efficient to send a series of unicast messages, one per device, than a single broadcast message.

Many types of content are compatible with the idea of broadcasting global data rather that sending specific data to each device, reducing network usage more or less significantly depending on the application. A good design principle in this instance is to start by asking what the global data demands are for a specific type of content. For example, in our composition *The Mind at Work* (discussed in [REFERENCE FORTHCOMING]) we created complex phase patterns between devices using a global "phase offset" variable.

3 THE BROADCAST FIRST 'RECIPE' FOR CREATIVE CODING MULTIPLICITIES

Stemming from these considerations, we propose the "broadcast first" recipe for creative coding multiple remote devices. The rationale for the recipe is that for any multiplicities system basic decisions have to be made about what programs are written on devices and what programs run on a central controller computer, taking into account bandwidth, programming ease and programming flexibility. By programming ease we mean: firstly how conceptually clean the breakdown is between these pieces of code; is there a clearly followable design concept that dictates how these elements should be divided up, that a creative practitioner can easily hold in their mind (as with MIDI and analogue synthesisers); and secondly, how practical it is to create such programs, or even better, to remix ready-made software components where possible. For example, writing a complex model that runs on a device and keeping that model in lock-step synchronisation with other devices may turn out to be difficult and error prone (discussed below in relation to a 'boids' model). Programming flexibility extends the issue of programming ease by considering how such elements can be reconfigured, tweaked, debugged, hacked or remixed for other applications, in a creative workflow.

We claim that in the majority of situations, we can maximise programming ease and programming flexibility by taking the broadcast fist approach, which works with the

simplifying assumption that the best solution is the one that: (1) places everything that can be decided on a central computer; in so far as (2) only broadcastable (i.e., global) messaging is sent. This corresponds to item 3 in our spectrum of software architectures above. We situate simple renderers (synthesisers) and basic spatially-aware asynchronously activated behaviours on devices, and manage most of the scheduling and high-level management on a controller computer.

The broadcast first recipe is as follows:

1. select an audio rendering paradigm (e.g., FM synthesis, granular synthesis) and consider the suitability of its processor demands and control parameters for running on the remote devices.
2. identify everything that can be represented as global data and what mappings are needed to get from global values to specific renderer parameters.
3. check that there is the bandwidth to send all of this global data.
4. if not either (a) reduce what is included in the global data list or (b) simplify the creative concept or (c) abort the 'broadcast first' recipe.
5. implement the device-side behaviours.

The purpose of such a recipe is to offer a clear path through the solution space. We propose that creative practitioners treat the broadcast-first recipe as their go-to solution to be abandoned only if ineffective.

Having decided to adopt this recipe we can also identify ways that it can be easily formalised in higher level end-user tools. Specifically, a formal framework supporting the broadcast-first recipe would help define exactly what the bandwidth and CPU demands were of any specific implementation. This would support a more fully-fledged simulation toolkit that would allow a developer to test in simulation the performance of specific content designs before fully implementing them. Conceptually, this can be presented in terms of a bandwidth and CPU budget that the user can allocate more creatively, much as they contend with track CPU demands, polyphony and MIDI channel constraints in existing music tools.

4 WIFI AND CPU PERFORMANCE IN A PRACTICAL EXAMPLE

Whilst programming ease and programming flexibility is harder to define and to measure, we can more precisely measure how WiFi bandwidth and CPU constraints play out in a practical example. The result of this is not usable numbers, but an examples of how the recipe is effectively applied.

The example considers how a 'boids' flocking model [23] can be used to activate light and sound effects on our massively distributed light-and-speaker artwork. Imagine 100 (or more) virtual birds ('boids') flocking in murmuration patterns above an audience's heads. Imagine further that the system must run in realtime (i.e., not pre-recorded) so that the boids can be programmed to flock around people being tracked in the space. These are rendered on a distributed network of lights and speakers hanging from the ceiling. Whether there are 100, 1,000 or 10,000 lights and speakers, the concept remains the same: the boids are virtual objects moving through the physical space, and 'activate' a light or speaker whenever they pass near. This is one example of content one may wish to run on such a system and has properties that are specific to this content type, being a complex multi-agent model. Other types of content would have different properties, though we will find that many content types pose similar constraints.

The number of boids, the number of lights and speakers, and the expected framerate and latency of the system are all variables that impact how one might implement this content on the given hardware. Our goal is to come up with designs that will scale well to high numbers of lights and speakers and run at a reasonable frame rate and latency. Frame rate expectations are well-defined by human perceptual constraints: below about 30FPS will look stuttery. Above 60FPS is ideal. Over 100FPS is unnecessary. Latency expectations vary depending on the application. For digital musical instruments such as MIDI synthesisers, latency needs to be less than 10ms, and ideally below 1ms to avoid any performer awareness of it. For the example considered here, the boids movement in response to the audience can be much slower, and a latency up to 1s may be acceptable. We might expect that as the number of speakers scales we would also want to increase the number of boids to fill a larger space. A related question is what information we actually intend to use from the boids model: individual boid positions (x,y,z values), boid directions and speeds, and other types of metadata such as the density of boids at any given point may all be relevant to the visualisation and sonification process. In addition an enhanced boids model might include other state information pertaining to individual boids, such as a changing colour or shape.

We can now consider some of the basic constraints of our hardware arrangement.

4.1 CPU usage in a distributed approach

Each Raspberry Pi controls two distinct speakers and hence two distinct instances of the audio renderer. We have designed a simple monophonic synthesiser that can switch modes between sample playback, granular sample playback (multiple overlapping sound grains) and FM synthesis. Sample playback can be varispeed and looped or re-triggered according to a network-synchronised global clock. A further LFO can be assigned to a filter, ring modulator or amplitude modulator. A similar setup exists for LED rendering (with 4 or 8 LEDs per controller device) but has significantly lower CPU overhead since audio requires thousands of sample calculations per second, versus tens of calculations for LEDs. There are also additional fixed costs in our system relating to other runtime functions. Once this 'fixed cost' CPU cost is accounted for, the remaining CPU budget can be used for any behavioural actions that are run in response to incoming commands, or are driven from the internal clock according to the programmer's instructions,

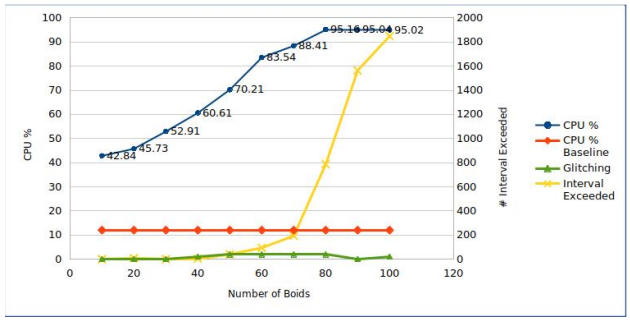


Fig. 4. CPU usage given the number of boids in a boids model updating at 30ms intervals, compared to a baseline CPU usage when no boids model is running. Interval exceeded refers to the clock update calculation taking longer than the clock interval, counted automatically. Glitching refers to audible interruptions to the audio, counted manually.

including running a model such as the boids model, performing spatial calculations and so on.

For comparison to the broadcast first approach, we experimented with running a boids model on a Raspberry Pi on top of the default fixed CPU cost renderer code described above (as if running the model in parallel on multiple devices). With our system we could can run a boids model up to around 40 boids before the model update calculations start to interrupt the audio thread causing glitches. We emphasise again that one can create far more optimised systems with the same behaviour and better CPU/GPU performance. These numbers do not describe the full capability of the Pi, but our experience in a specific creative project where coding ease and flexibility are amongst the highest ranking criteria, with a system that has proven effectiveness in this area.

4.2 WiFi usage in the broadcast first approach

The boids example is interesting because it is a complex multi-agent system and it cannot be easily parallelised across devices: the devices act as a substrate to render the model and all devices must have all information pertaining to the model. You *can* parallelise a spatial model such as boids by dividing space into regions, but this is far more complex than just writing a boids model (which can be done very easily) and suffers from a wide variation in effectiveness, with potential worst-case scenarios that are undesirable. In other words, attempting to parallelise the model is a poor solution in terms of ease and flexibility of programming, as well as predictability. There are therefore two immediate solutions: (1) centralise the computation and then broadcast the entire model in realtime; (2) run the exact same model on every device and take steps to ensure that these models remain in lockstep. In (REFERENCE FORTHCOMING) we show that for option 2 to be robust it must be possible for any device that has fallen behind to receive a full state update over the network quickly enough to be able to catch up. This means that although the average and best-case bandwidth usage of option 2 is significantly lower than option 1, the bandwidth constraints

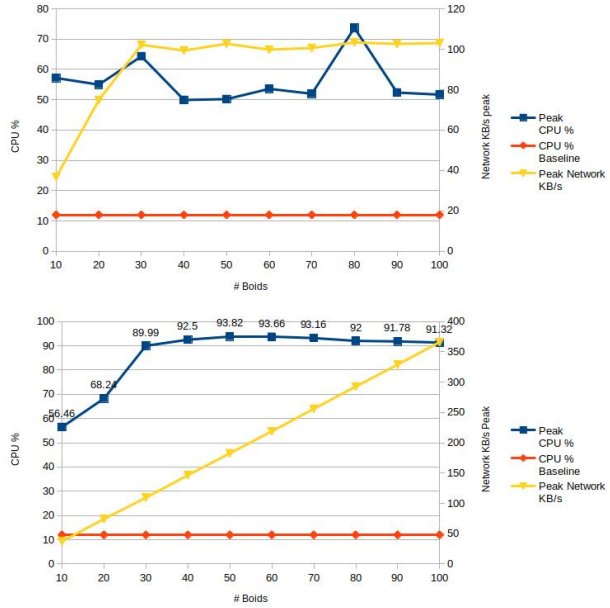


Fig. 5. Network and CPU demands sending boids model state over the network at 30ms intervals for broadcast (top) and unicast (bottom) contexts. The broadcast speed is capped by the router, fixed by the router manufacturer's hardware implementation, despite there being significantly more bandwidth available. Counterintuitively, therefore, sending multiple unicast messages outperformed broadcast. A significant CPU load, comparable to actually running the model itself, is also noted for the work of receiving and interpreting the incoming message.

are actually identical: it must be possible to send the entire model's state over the network at each time step of the model.

We experimented with the success of sending data from boids models of different sizes over a WiFi network in a practical realworld context (using the real hardware setup of our installation). Again, these results do not describe the optimal design but a design that has arisen naturally from the development of a creative work. For example, sending the data via OSC/UDP as a command and floating point argument list at each time step is not the most bandwidth efficient method, but it is incredibly convenient to write, reduces the complexity of code, easing refactoring, and highly interoperable across different creative coding platforms. We found that we could send a model of 100 boids without hitting the constraints of our router (an affordable domestic gaming router[1]). Due to this router's broadcast settings actual broadcast performance was worse than quickfire unicast performance, being throttled at just over 100KB/s, which is incredibly slow compared to the system's physical network capacity. It is also worth noting that the CPU demands of processing the incoming data was also taxing on devices, in fact comparable to running the model itself.

[1]ASUS RT-AC86U Dual-Band AC2900 MU-MIMO Nitro-QAM Gigabit WiFi Gaming Router

Thus there are significant factors that make the broadcast first recipe suboptimal. However, these are in part due to the network hardware. Regardless, even though it was advantageous to replace convenient broadcast messaging with counterintuitive and more cumbersome batch unicast messaging, we consider this still to conform to the broadcast first recipe as a conceptual approach to build work. As we will discuss below, our use of a boids model was by far the most data rich of the content types we used.

5 Applying the Broadcast First Recipe

Applying the broadcast first recipe, we begin with the assertion that the boids model should run on a central computer and not be distributed or run separately in parallel on the network devices: based on the reasonable assumption that it is impractical and overly complicated to do either, at least pre-emptively. This determines that we must broadcast the relevant model state information, and make decisions about what data is needed. Decisions about how data will be sonified and visualised are creative decisions which are informed by the technical constraints and possibly iterated over time. We have good evidence from our own experiences, discussions with fellow artists, and the academic literature (e.g., [24, 25, 26]) that it is common for artist visions to be quite flexible in light of emerging practical constraints, and even often to creatively leverage the constraints of the media as presented to the artist in any given configuration (pers. comm. Liam Birtles, Squidsoup).

Following the broadcast first recipe leads to a quick and low-complexity first draft implementation which helps define a set of known constraints. If we are not happy with performance we may then choose between optimisation or altering the creative goals. Thus we posit that this recipe gets us towards a functional working juncture in the most efficient way possible, maximising creative freedom.

For many other types of content we can see that this is also a workable solution, if not optimal. Even if not optimal, we claim that the broadcast first approach is the best all-rounder starting solution. Implementing this as a design framework means that we can support great savings in terms of ease and flexibility of programming by adding further support tooling built around this strategy, discussed further below.

5.1 Where does the broadcast first recipe break?

As we increase the number of boids in a model, then at some point we will hit our network's capacity. As we have discussed, although there may be designs that may be more robust and potentially optimisable than the broadcast first approach they would almost certainly be more complex to program, debug and maintain. Restating the great importance of flexibility for artistic practitioners employing a creative coding approach, the broadcast recipe seems a sensible general strategy and if the artist is able to understand and work with the bandwidth constraints of the system then this approach will best support creative flexibility and freedom by maintaining the conceptual simplicity of the system.

Many other content scenarios present similar profiles in terms of CPU and network bandwidth. In *The Mind At Work* another content type used was a starburst of light and sound that could be directly controlled from a controller computer. This type of content was event-based, like MIDI note events: a specific burst event could be initiated, updated and destroyed, allowing a polyphonic approach to making starbursts. In this case the broadcast data contained a position, size, colour and soundfile ID for each starburst; a very compact message, enabling complex, realtime controlled movement of light and sound across the space. In this case, some lightweight code is placed on each device to interpret the incoming broadcast messages. Distinct from low-level render code, this code deals with spatial calculations and handling the synchronisation of sounds, forwarding commands to the lower-level rendering system.

Such event types are compact geometric representations, much like vector graphics data. By contrast to vector graphics, bitmap data is not compact and presents the highest demand on such a framework; the easiest way to tax the broadcast first approach is to broadcast rich uncompressed video data. Even then, the bandwidth capacity of a regular WiFi network can easily handle 4K video and beyond and still satisfies the broadcast-first approach.

Unlike broadcasting an image to be rendered, streaming audio to individual speakers would not qualify as a broadcast first approach since that requires unicast addressing of speakers. This is likely to blow out the bandwidth budget quickly as we scale. However, we may broadcast audio source data combined with spatial position data for that source, as is common in 3D audio systems. Again, we can calculate the number of audio channels that our WiFi bandwidth would permit for a given audio file quality and work creatively within that constraint.

In all cases there is nothing we can do to overcome the hard latency constraints of WiFi beyond standard latency tuning in our device software and choosing the lowest latency hardware, but existing latencies have become quite acceptable for a number of realtime performance contexts.

6 Opportunities to further ease programming load and flexibility

We have claimed that a broadcast first recipe is a strong heuristic approach to effective creative coding, recommended as a basic strategy to be adopted by creative coders working with massively multi-device systems. Additionally, having established this recipe, a number of supporting framework features can be conceptualised that further ease working in this way. These are briefly discussed here:

6.1 Simplified coding

In HappyBrackets we have started to develop a reduced API based on this approach with some simplifying features. One is to use code annotations in our on-device sketch code that automatically maps variables and

functions to incoming OSC commands. For example, if we add the function `lightBlob(float x, float y, float z, float radius)` to a HappyBrackets sketch then the sketch will automatically respond to the associated OSC command. Likewise, declaring a primitive field, `positionX` with the associated annotation enables that field to be set directly over OSC. Combined with API support for doing spatial calculations, and a handful of readymade renderer behaviours for audio and light synthesis, we have started to significantly reduce the coding work and conceptual overhead in hacking dynamic behaviours. We refer to these code blocks as 'spatial behaviours': they sit between the master controller responsible for the global control and scheduling, and the low-level renderer behaviour. Additionally, this reduces most sketches to a simple list of fields and asynchronously-called functions, which makes programs less complex and interconnected and subsequently supports 'lazy' copy-and-paste coding: we can provide a reference library of useful functions to copy and paste from. This remains work in progress.

6.2 Simulating and swapping hardware

This clear architectural breakdown between controller computer (broadcasting control data), spatial behaviours and low-level renderers offers clearer ways in which common protocols can be established that define interoperability between different hardware and software elements. Our on-device behaviours, coded in Java and running on our HappyBrackets runtime, include both the spatial behaviours and the low-level renderers, but these could be easily separated, and indeed our electronics partners Bitscope are working on a PCB design that offers programmable sub-processors accessed over serial. We have also built a simulator in Unity which interoperates directly with our coding environment (a plugin to the IntelliJ Java IDE). A CSV file describing a physical installation configuration can be loaded into the simulator, or directly onto the devices.

6.3 Automating behaviour predictability and architecture design decisions

Clearly defining the operational bounds of network code means that we can more easily profile the network demands of a given program in a test-driven way. For example, having defined the `lightBlob()` function above, a test function could be run to tell us the network and CPU budget of the function under a range of circumstances. An entire on-device sketch could be automatically queried in this way to profile the network behaviour. In this way a creative coder could directly visualise the expected network and CPU demands of any code they write as they go. This gives the coder rapid feedback on whether their design is going to hit a wall as they are developing it, and is thus very well suited to a creative coding workflow where rapid feedback is invaluable in guiding creative directions.

6.4 Supporting parametric design

Lastly, we can also query the annotated code present in any on-device sketch and use that to broadcast an OSC API that could be used by any other program to build a control GUI or other system for managing control of values. Following the same example, we could advertise the `lightBlob(float x, float y, float z, float radius)` function, and a compatible program could build a GUI on the fly containing the relevant variables. This is also something we are yet to implement, but in our most recent work we have found that even manually constructing GUIs in MaxForLive that can then be controlled directly, via MIDI controller hardware, or from the Ableton Live timeline offers a huge amount of creative flexibility. Ableton Live is just our preferred choice and any number of other live or timeline-based workstation programs can fill the same role, using the common language of OSC. A final potential here is that such an API can be channelled into a parametric design process, where a user could parametrically explore the space of behaviours that the system offers.

7 CONCLUSION

In this paper we have presented our experiences developing creative multiplicitous media installations considering a wide range of constraints combining creative constraints such as programming ease and flexibility, and the hardware constraints of the technology involved. We have presented a basic physical architecture that we have found to be cost effective and scalable, and have considered how the demands of creative programming flexibility point to a preferred creative design strategy given this architecture. We have shown how this approach plays out in terms of CPU and bandwidth constraints and what options are available to improve performance if these constraints prove a problem. We have also speculated on how following this approach as a paradigm allows for additional support structures that will further ease the creation of media multiplicities works.

8 REFERENCES

[1] O. Bown, S. Ferguson, "Understanding media multiplicities," *Entertainment Computing*, vol. 25, pp. 62–70 (2018).

[2] L. Turchet, C. Fischione, G. Essl, D. Keller, M. Barthet, "Internet of musical things: Vision and challenges," *IEEE Access*, vol. 6, pp. 61994–62017 (2018).

[3] H. Hauesler, T. Barker, K. Beilharz, "Interactive Polymedia Pixel and Protocol for Collaborative Creative Content Generation on Urban Digital Media Displays," presented at the *Proceedings of the International Conference on New Media and Interactivity*, pp. 1–7 (2010).

[4] M. Sato, A. Hiyama, T. Tanikawa, M. Hirose, "Particle Display System - Virtually Perceivable Pixels with Randomly Distributed Physical Pixels," *Journal of Infor-*

mation Processing, vol. 17, pp. 280–291 (2009), doi:10.2197/ipsjjip.17.280.

[5] J. A. Paradiso, J. Lifton, M. Broxton, "Sensate media - Multimodal electronic skins as dense sensor networks," *BT Technology Journal*, vol. 22, no. 4, pp. 32–44 (2004).

[6] S. Seitinger, D. S. Perry, W. J. Mitchell, "Urban Pixels: Painting the City with Light," presented at the *Proceeding of the 27th Annual CHI Conference on Human Factors in Computing Systems - CHI '09*, pp. 839–848 (2009), doi:10.1145/1518701.1518829.

[7] H. Hörtner, M. Gardiner, R. Haring, C. Lindinger, F. Berger, "Spaxels, Pixels in Space," presented at the *Proceedings of the International Conference on Signal Processing and Multimedia Applications and Wireless Information Networks and Systems*, pp. 19–24 (2012).

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

[9] M. Deshpande, S. Sarwar, A. M. Goloujeh, D. Papanikolaou, "Pneuxels: A Platform for Physically Manifesting Object-based Crowd Interactions in Large Scales," presented at the *UbiComp/ISWC 2019- - Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, pp. 45–48 (2019).

[10] J. Hallam, C. Zheng, N. Posner, H. Ericson, M. Swarts, E. Y. L. Do, "The Light orchard: An immersive display platform for collaborative tangible interaction," presented at the *UbiComp/ISWC 2017 - Adjunct Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, pp. 245–248 (2017).

[11] M. Coelho, J. Zigelbaum, J. Kopin, "Six-forty by four-eighty: The post-industrial design of computational materials," presented at the *Proceedings of the 5th International Conference on Tangible Embedded and Embodied Interaction, TEI'11*, p. 485 (2011).

[12] M. Haeusler, *Media Facades – History, Technology, Content* (Avedition) (2009).

[13] P. Dalsgaard, K. Halskov, "Designing urban media façades: cases and challenges," presented at the *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2277–2286 (2010).

[14] A. Rowe, "Within an Ocean of Light: Creating Volumetric Lightscapes," *Leonardo*, vol. 45, no. 4, pp. 358–365 (2012).

[15] A. Rowe, G. Bushell, L. Birtles, C. Bennewith, O. Bown, "Submergence 2013," *Leonardo*, vol. 49, no. 4, pp. 356–357 (2016).

[16] S. Ferguson, A. Rowe, O. Bown, L. Birtles, C. Bennewith, "Networked Pixels: Strategies for Building Visual and Auditory Images with Distributed Independent Devices," presented at the *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition*, pp. 299–308 (2017).

[17] S. Ferguson, A. Rowe, O. Bown, L. Birtles, C. Bennewith, "Sound Design for a System of 1000 Distributed Independent Audio-Visual Devices," presented at the *In proceedings of NIME 2017* (2017).

[18] A. Vallgårda, L. Boer, V. Tsaknaki, D. Svanaes, "Material Programming," presented at the *Proceedings of the 2016 ACM Conference Companion Publication on Designing Interactive Systems - DIS '16 Companion*, pp. 149–152 (2016), doi:10.1145/2908805.2909411.

[19] O. Bown, S. Ferguson, L. Bray, A. Fraietta, L. Loke, "Facilitating Creative Exploratory Search with Multiple Networked Audio Devices Using HappyBrackets," presented at the *New Interfaces for Musical Expression* (2019).

[20] A. Fraietta, O. Bown, S. Ferguson, "Transparent Communication Within Multiplicities," presented at the *2020 27th Conference of Open Innovations Association (FRUCT)*, pp. 61–72 (2020).

[21] A. Fraietta, O. Bown, S. Ferguson, S. Gillespie, L. Bray, "Rapid composition for networked devices: HappyBrackets," *Computer Music Journal*, vol. 43, no. 2-3, pp. 89–108 (2019).

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

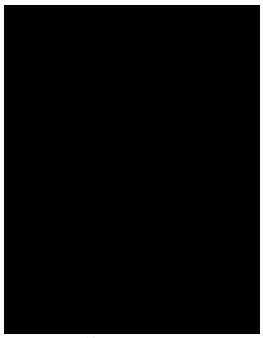
[23] C. W. Reynolds, "Flocks, Herds and Schools: A Dstributed Behavioural Model," presented at the *Computer Graphics (SIGGRAPH '87 Conference Proceedings)*, vol. 21, pp. 25–34 (1987).

[24] M. Biskjaer, K. Halskov, "Decisive constraints as a creative resource in interaction design," *Digital Creativity*, vol. 25, no. 1, pp. 27–61 (2014).

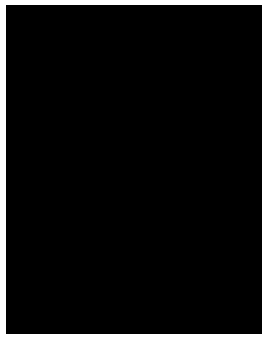
[25] D. N. Perkins, "Creativity: Beyond the Darwinian Paradigm," in M. Boden (Ed.), *Dimensions of Creativity*, chap. 5, pp. 119–142 (MIT Press) (1996).

[26] D. K. Simonton, "Creativity and discovery as blind variation: Campbell's (1960) BVSR model after the half-century mark." *Review of General Psychology*, vol. 15, no. 2, p. 158 (2011).

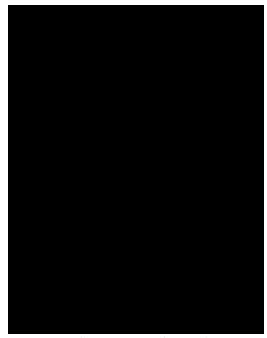
THE AUTHORS



Oliver Bown



Sam Ferguson



Augusto Dias Pereira dos Santos

Kurt Mikolajczyk

Oliver Bown is senior lecturer and co-director of the Interactive Media Lab at the faculty of Art and Design at the University of New South Wales, in Sydney, Australia. He is a researcher and maker working with creative technologies, with a highly diverse academic background spanning social anthropology, evolutionary and adaptive systems, music informatics and interaction design, with a parallel career in electronic music and digital art spanning over 15 years. He is interested in how artists, designers and musicians can use advanced computing technologies to produce complex creative works. His current active research areas include media multiplicites, musical metacreation, the theories and methodologies of computational creativity, new interfaces for musical expression, and multi-agent models of social creativity.

●

Sam Ferguson is a Senior Lecturer within the School of Computer Science at UTS. He has a background in music performance, cognitive science, and psycho-acoustics and acoustics. He focuses on sound and music and its relationship with creativity and human experience, in contexts such as installation art, creative coding and machine learning, as well as focusing on cognitive science. He has more than 80 publications in areas as diverse as spatial hearing and loudness research, to datasonification, emotion, and tabletop computing. He is currently undertaking an ARC Linkage project on creative coding and multiplicitous media, and has collaborated with industry in many previous projects. He has served as Deputy Head of School (Teaching and Learning) for one the highest ranked Computer Science departments in Australia, and has taught a wide variety of subjects within the Interaction Design discipline.

●

Augusto Dias is a Research Associate at the UNSW working in the space of media multiplicities. He was awarded his PhD at the University of Sydney working with machine learning and dance. His publications range from social media analytics and educational data mining to dance and creative media technologies.

●

Kurt Mikolajczyk is a musician, composer and creative coder currently undertaking a PhD in interaction design at The University of Technology Sydney. In 2019 he completed a Masters of Music at the Sydney Conservatorium of Music, developing software tools for composing polytemporal music and a portfolio of works for jazz ensemble and laptop. In 2020 he was part of the design team for The Mind At Work at Casula Powerhouse Arts Center, a dynamic light and sound installation of 200 LEDs and 100 speakers controlled by a networks of 50 Raspberry Pis.