Hyperproduction: an audio-centric framework for the abstract modeling of live performance to guide audience attention and perspective using connected real-time systems

Benjamin A. Bloomberg

B.S., Massachusetts Institute of Technology, Cambridge (2012)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2014

Author _____

Program in Media Arts and Sciences
August 18, 2014

Certified by _____

Tod Machover
Muriel R. Cooper Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by _____

Prof. Patricia Maes
Interim Academic Head
Program in Media Arts and Sciences

Hyperproduction: an audio-centric framework for the abstract modeling of live performance to guide audience attention and perspective using connected real-time systems

Benjamin A. Bloomberg

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning on August 18, 2014,
in partial fulfillment of the requirements for the degree of Master of Science
in Media Arts and Sciences

Abstract

Hyperproduction is a conceptual framework and a software toolkit which allows producers to specify a descriptive computational model and consequently an abstract state for a live experience through traditional operating paradigms such as mixing audio, operation of lighting, sound or video systems. The hyperproduction system is able to interpret this universal state and automatically utilize additional production systems, allowing for a small number of producers to cohesively guide the attention and perspective of an audience using many or very complex production systems simultaneously. The work focuses on exploring the relationship of conventional production systems and techniques to abstract computational models of live performance. The conceptual framework will identify key elements of an effective model in each case, with attention and perspective as the cornerstones of this exploration. Several examples of hyperproduction systems will be constructed and evaluated in a variety of live experiences, including sound-only performance, live broadcast, and remote interactive audience extension.

Thesis Supervisor: Tod Machover
Title: Muriel R. Cooper Professor of Music and Media

Hyperproduction: an audio-centric framework for the abstract modeling of live performance to guide audience attention and perspective using connected real-time systems

Benjamin A. Bloomberg

The following person served as a reader for this thesis:

Thesis Reader

Joseph A. Paradiso
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Hyperproduction: an audio-centric framework for the abstract modeling of live performance to guide audience attention and perspective using connected real-time systems

Benjamin A. Bloomberg

The following person served as a reader for this thesis:

Thesis Reader _____

Chris Chafe
Director, Center for Computer Research in Music and Acoustics (CCRMA)
Duca Family Professor of Music and Technology
Stanford University

Acknowledgments

Thanks to Tod Machover, my advisor, for mentoring me for the last seven years, and to Chris Chafe and Joseph Paradiso for their feedback as readers.

To Peter Torpey and Elly Jessop for many useful discussions throughout all stages of this work, and their assistance and moral support in debugging many problems and thought experiments. Thanks especially to Peter for spending countless hours designing and producing work, building systems and imagining new paradigms for performance. To Charles Holbrow for his help and guidance in the studio. To Akito van Troyer for loaning every cable and dongle on the face of the earth. To Simone Ovsey for her incredible production skills, and for keeping me in line and on schedule despite high pressure, high stakes, and ambitious goals. To Luke, Garrett, Justin and Dash for their prowess as software developers and undergraduate researchers. And to Sarah Ryan and Kelly Donovan, for helping us order all of the parts and equipment for countless productions.

Contents

Chapter 1

Introduction

Control Systems for Live Production have been around for quite a long time. Even the first "systems", Deus Ex Machina—the theatrical effect, rather than the cliché plot device—from thousands of years ago, required the development of a protocol to coordinate the execution of many elements with the right sequence and timing.[20] Initially these protocols were borrowed from sailors' whistles. It has since become taboo to whistle on a stage; you may cause a sand-bag or drape to be dropped from the heavens.

This thesis presents Hyperproduction as a new paradigm for control in live production. Hyperproduction is a conceptual framework and a software toolkit which allows producers to specify a descriptive computational model and consequently an abstract state for a live experience through traditional operating paradigms such as mixing audio, operation of lighting, sound or video systems. The hyperproduction system is able to interpret this universal state and automatically utilize additional production systems, allowing for a small number of producers to cohesively guide the attention and perspective of an audience using many or very complex production systems simultaneously. Hyperproduction builds on theatrical *traditions*, as well as practices we've employed in the Opera of the Future group, by thinking about meaningfully *modeling* what happens in a live performance and the creative skill of performance *operators*.

Traditions

There are several paradigms today that have become standard practice (replacing whistling) for production system control. The advent of low-cost personal computers and digital systems has made these available to even the smallest of productions. We will begin by understanding how and why current technologies and protocols work and what their advantages are. Note that this document is not an in depth technical brief on existing technologies; we will cover enough to understand the difference between existing technologies and proposed newer methods in this thesis. For an in-depth technical explanation of traditional show control, please see John Huntington's Control Networks for Live Entertainment.[23] It is important to note that since many of these systems have developed around technology as it has become available and evolved through history, even the most advanced are sometimes based on philosophies that may no longer be pertinent given the capabilities of technology today.

Opera of the Future

While these existing systems work well and are used quite extensively in the entertainment industry, there are many types of production for which they are ill-suited. If flexibility in timing and a high degree of synchronization are required, many existing commercial systems become very complex to operate and configure. To address these shortcomings, I will detail several specific Opera of the Future Group strategies that augment the traditional methods of control. The Hyperinstrument was a first attempt at creating a live production system which had great synchronization and nuance in timing. We will also understand how these strategies are used to great success in recent productions of Tod Machover's *Death and the Powers*[31] and the *Sleep No More* Internet Extension.[9]

The Model

Dynamic production systems capable of intricate synchronization and nuanced timing offer much capability for live performance production. However, even

with that capability, a higher-level abstract model of the performance is sometimes necessary to give technological systems a representation of how elements of the production should be articulated. *Powers* and *Sleep No More* are particularly good examples of this, because their abstract representations are critical to the realization of the production, but also very different in approach and implementation. In *Death and the Powers* the model which drives the entire production infrastructure is derived from the emotional state of a single character.[44] *Sleep No More* uses a model where time and space do not necessarily adhere to the same rules as our physical world— this becomes more complex when the model needs account for people and events that *are* in the physical world.

The Operator

In understanding these more recent methods, we will also realize how important the role of the operator is to the success of interconnected production systems. As systems become increasingly dynamic, an operator takes on much more responsibility. In the recent past, operators' roles, especially for Oprea of the Future productions, have become performative in nature. In many cases, the mix engineer, visuals operator and other personnel are responsible for the ultimate shaping of the piece— it is they who ultimately determine what the audience perceives. Until recently, this has been achieved by constraining the *output* of the dynamic systems. For example, a hyperinstrument may generate additional accompaniment and effects which are all fed to a mixing console, then balanced, emphasized and ultimately fit together in a musical way by the mixing engineer.

Hyperproduction

The question then becomes whether it is possible to take advantage of the nuanced input and shaping an operator affords in a more effective way. We will see that for *Powers* and *Sleep No More*, an operator was integrated into both systems— not just for shaping and controlling output of systems— but to alter the input to systems as well.[44] Another simple analogy is that of electronic

music, where controllers are used to give some meaningful state to a representation (ex. an Ableton[2] set) which renders the performance based both on input from an operator. In the case of electronic music, we generally consider the operator to be a performer. Indeed, Philip Glass credits an "Onstage audio engineer" in his ensemble[36] for such landmark pieces as *Einstein on the Beach*.[14]

We should imagine the future of production to be performance-dominated art. As technology affords more capability to live systems, it should be the goal of system designers to create infrastructure that involves humans in emotional and nuanced fashions. Hyperproduction is a first attempt at addressing this phenomenon head on. It is a system designed to accommodate many types of abstract representations and allow performers, operators and other systems to contribute state to those models. A model can then be translated or mapped to data that are suitable for controlling production elements.

In this thesis, I will show that *Death and the Powers* and *Sleep No More* are examples of rudimentary Hyperproduction systems. We will explain the design and implementation of a completely new framework for facilitating the creation of hyperproduction systems, then proceed to show how systems for *Powers* and *Sleep No More* might be implemented in this newer framework. Most importantly, we will understand the additional capability gained by using a purpose-built infrastructure to address this particular type of interaction. Hyperproduction offers the ability to integrate operators virtuosic abilities into production systems along with performance data. More importantly this integration of performer and operator input takes advantage of abstract representations that are more flexible and capable than any system previously used in Opera of the Future. We will see the benefits of this expanded flexibility and capability in the following chapters.

Chapter 2

Related Work

Hyperproduction is a departure from previous attempts at modeling performance and large scale performance control systems. Below is a brief summary of related systems and research with capabilities similar to Hyperproduction.

2.1 Abstract Representation

Mapping and abstract modeling research generally focuses on a single strategy to try and represent a piece or performance. We have seen this type of approach in existing efforts by Camurri[6] using the Kensei model, Wanderley [22] looking at the relationship between controllers and sound generation, the work of Elly Jessop[26] using the Laban effort model[29] for gesture quality analysis, and even for prior Opera of the Future Productions[17] using a single strategy for analysis and representation. Figure 2.1 shows an example of a mapping created for a production in Peter Torpey's Disembodied Performance System[44]. Although the mapping can be switched for different sections of the piece, only one mapping is active at time in this system. Hyperproduction allows designers to combine many different models, representations and mappings together.

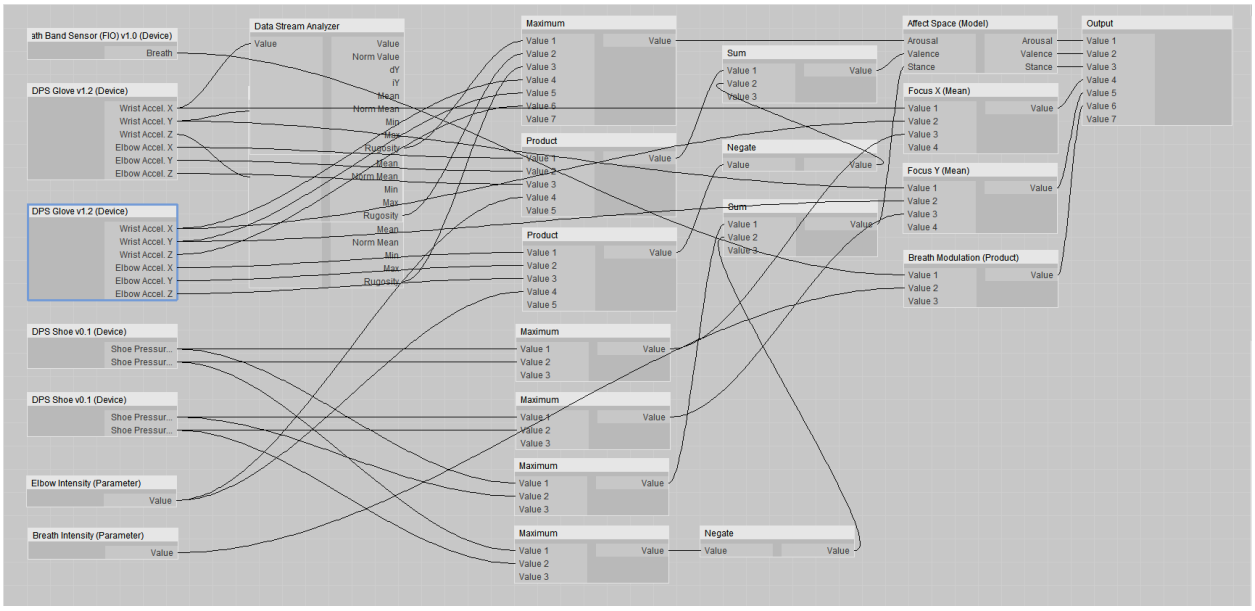

Figure 2.1: A single model abstract representation for *Death and the Powers* created with Peter Torpey's Disembodied Performance System[44]

2.2 Automated Mapping

Other approaches, such as Wekinator[18] and Marc Downie's Field[12], involve self-training or intelligent systems intended to allow one to specify input and output while letting the computer do the mapping. These systems assume that it is possible manually specify what the output should be for a given input, but this is generally not feasible on many production systems. Often there are hundreds of thousands of configurable parameters in a large production. For example, *Death and the Powers* has over 400 lighting instruments each with 40+ parameters of control. To set any sort of output on such a large system requires some way to manage a very large parameter set.

2.3 Literal Mapping

Another set of commercial products attempts to model live performance in a literal way, directly using sensor derived data, such as location or orienta-

tion of objects, to control production infrastructure. Ubisense[16], a tracking platform most commonly used in industrial applications for factory machinery automation and Blacktrax[4], a full blown theatrical tracking and show control package, are well known solutions for this type of control. Still these products are not able to represent abstract qualities, which are often much more important factors in production control than direct measurements such as location or speed.

These types of systems make perfect input systems for Hyperproduction. They can contribute useful data that might be incorporated into an abstract model, combined with input from operators and other sensors and *then* used to control production infrastructure.

2.4 Asynchronous Architecture with Flexible Frame-Rate

There are several existing systems which take a similar approach to control and architecture. These are designed to be ultimately flexible and accept input and output data from a variety of sources and are commonly found in performance systems.

SuperCollider[42] and Chuck[8] are examples of two systems which use a very similar architecture but are ultimately designed for sound synthesis rather than representation, mapping or analysis of performance. Of the two systems, SuperCollider is the more closely related package, since it's DSP and control components are separated. SuperCollider is broken into scsynth, responsible for generating audio and sclang, which communicates to scsynth via OSC and provides all control. Hyperproduction is similar to the sclang portion of SuperCollider but is based on NodeJS, which provides a greater degree of flexibility. Type-checking in Hyperproduction is left to module designers since javascript is not inherently typed.

2.5 Large-scale production

Large-scale production such as Broadway, Arena touring, or Vegas-like permanent installations have some of the most involved production infrastructure. Systems used for 8 performances a week, year-round must be designed carefully to ensure there are appropriate failure mechanisms and that systems will work even with unexpected input. Interactivity in these systems is often kept to a minimum for this reason.

In the next chapter we will explore typical protocols and technologies employed in large-scale productions, but before we explore those, it is worth discussing a few interesting and not-so-traditional systems that have managed to break into the high-end market and provide capability not often found or trusted at this level of scale and stakes.

2.5.1 Meyer D-Mitri formerly LCS

Meyer D-Mitri[10] is a completely configurable, programmable audio environment comprising of both software and hardware to implement very unique and customized DSP systems. D-Mitri and LCS are found on many Broadway and Westend productions. Every permanently installed Cirque du Soleil production in Las Vegas uses a D-Mitri system for control and automation. Systems constructed with D-Mitri are built from scratch out of modules connected to the same network. Control surface modules[28] can be chosen and assembled in any way to allow for unique interactions with operators. Figure 2.2 shows an example of a surface constructed from customizable modules.

On the Tonight Show with Jimmy Fallon, an operator uses a small puck to specify if and where any performers go in the audience. The location of the puck over a floorplan allows the D-Mitri system to mute nearby speakers and prevent feedback from the performer microphones. An operator moving the puck while watching a top down camera of the audience seating is much more reliable than using a tracking system in this case.

Figure 2.2: A CueConsole modular audio control surface[28]

Cirque du Soleil uses D-Mitri to control over 512 independent speakers in their production, KA. The system is able to spatialize live microphones on performers to the speakers, located in the head-rest of the audience seats. The performers are continuously moving, occasionally through the air, so the system is able to compensate for their movement to ensure that they remain time-aligned in the PA system.

Westend and Broadway productions often use D-Mitri for complex routing challenges. In a recent production of Rocky on Broadway, designer Peter Hylenski used D-Mitri to manage a transition into the finale, in which the theater transformed completely from a traditional proscenium to in-the-round. Simultaneously, a boxing ring is lowered from the fly loft and cantilevered into the auditorium, while audience under the ring is relocated onto the stage. At this point the performers, all wearing microphones, are in-front of the sound system in the boxing ring. A second sound system is lowered from the ceiling of the auditorium and all time-alignment parameters are set to use the newly placed speakers correctly.

D-Mitri has many desirable qualities, but it is ultimately a DSP system. It is

very flexible in it's capabilities and control— entirely scriptable, network distributed and customizable with respect to operator interactions— but it does not have a way to define any model or representation that is not a traditional mixer model. Like BlackTrax and Ubisense, most interactions are literally translated to DSP processes.

2.5.2 TiMax

For an Arena production, it is difficult to correctly place performers in a PA system because the distances are so great. The problem is magnified when a production, like an ice show, happens in the round. Audience all over the arena must be able to localize voices to their respective performers. When speakers are hanging hundreds of feet above the people on stage, pointing in many different directions this becomes quite a challenge.

TiMax[43] uses real-time tracking to dynamically pan performers' audio around a large array of speakers. Because the movement of the sound reinforcement is linked with the movement of the performer, this provides a convincing way to associate the artificial reinforcement with people on stage.[34]

2.6 Hyperproduction with high-end systems

Hyperproduction aims to be a common platform for connecting all of these systems together. In our recent past, Opera of the Future has created many custom systems which do this type of combination, using operator input, live sensing, and performance capture to control high-end DSP, robotics, scenic automation, etc... We have interfaced with infrastructure big and small, and it is the goal of this platform to be able to connect to and control all types of systems using many varieties of input and different levels of interpretation.

Next, we will look at some of the more traditional technologies and protocols used in large-scale production (especially in large pop and rock music tours), but also found at smaller scales as well.

Chapter 3

Control Systems for Live Production

To make the presentation of these systems straight forward, I'll define several terms that we will use in our discussion and characterization of modern production systems:

"Production elements" are those elements which help the preformance achieve some stated artistic intention and are not executed by a performer. Scenic elements, lighting, sound, visuals, projection, automation and rigging are examples of production elements.

A "production control system" (or a "control system" for short) choreographs production elements throughout the duration of a performance. Sometimes a control system may talk to other control systems to orchestrate many production elements. In this case, we can separate the control systems into tiers. The lowest "first tier" systems generate data used to directly control production elements, while higher tiered systems may synchronize and link the lower tiers together.

First tier control systems send "control-data" to production elements. We can think of this data in two ways:

- a low level represention— i.e. how many bits of resolution, data rate, overall bandwidth.
- a high level represenation— i.e. a color, a point in 3-d space, a trajectory, an exponential fade.

Control data can be continuous, such as a representation of intensity of a lighting instrument over time, or momentary and binary, such as a trigger to open a trap door.

3.1 The Cue

A large majority of production systems are triggered directly by a human operator using cues. This type of control originated because it is simplest to communicate verbally, by gesture, whistle, noise, etc... Cues generally comprise of small segments of choreographed movement or changes in production elements. These segments are called by a stage manager or other operator and can be assembled in any order and refined in isolation. It gives a natural organizational structure and allows many complicated components to happen simulaneously.

In modern large-scale productions, cues are used for triggers at the most important moments— when the synchronization of on-stage action and production elements is critical to the performance. Important moments may include the start of the production, large movements of automated set pieces, effects involving danger to performers such as flying, or more mundane events, such as a change in lighting, or a sound clip that occurs with action on stage. When triggered, a cue (the choreography of the production elements) begins immediately and runs to completion. Once a cue is called, it generally cannot be altered and runs on a fixed timeline.

Lighting consoles (ETC[15], WholeHog[21], Chamsys[7], GrandMA), sound mixing consoles (Studer[41], Avid[3], Digico[11], Yamaha[46], Midas[30]), playback systems (QLab[19], SFX[38]), and rigging and automation systems (JR Clancy[25]) use cues to organize their functionality and their control-data output.

3.1.1 Digital Cues

The MIDI Show Control protocol is a digital implementation of a cuing system and allows control systems to cue other control systems. In fact, a whole class of systems called "show control" systems generate control-data which can be consumed by the first tier systems mentioned above. A show control platform can synchronize the operation of many production control systems so that lighting, sound, automation and rigging consoles can be synchronized together.[23] Figure 3.1 shows a typical arrangement of show control and first tier systems for controlling production elements.

3.1.2 Transitions and Interpolation

Often a production system must be able to make a transition smoothly from one cue to another. Depending on the type of control data, different types of transitions may be necessary. For example, the following control data types require different transitions:

- Transitioning colors smoothly requires the traversal of some form of color space, whether it is Red-Green-Blue, Hue-Saturation-Lightness, etc...
- Transitioning between 3 dimensional points (or n-dimensions) requires some trajectory between those points.

Interpolation provides a way for systems to make these transitions without explicitly defining every part of transition itself. We can define a default path from one state of the system to another. In the case of 3 dimensional space, this might be the shortest path in cartisian, polar, cylindrical, or any other representation of space.

3.1.3 Keyframes

Keyframes are important to cued systems that take advantage of interpolation methods. Keyframes give the state of the system at a particular moment

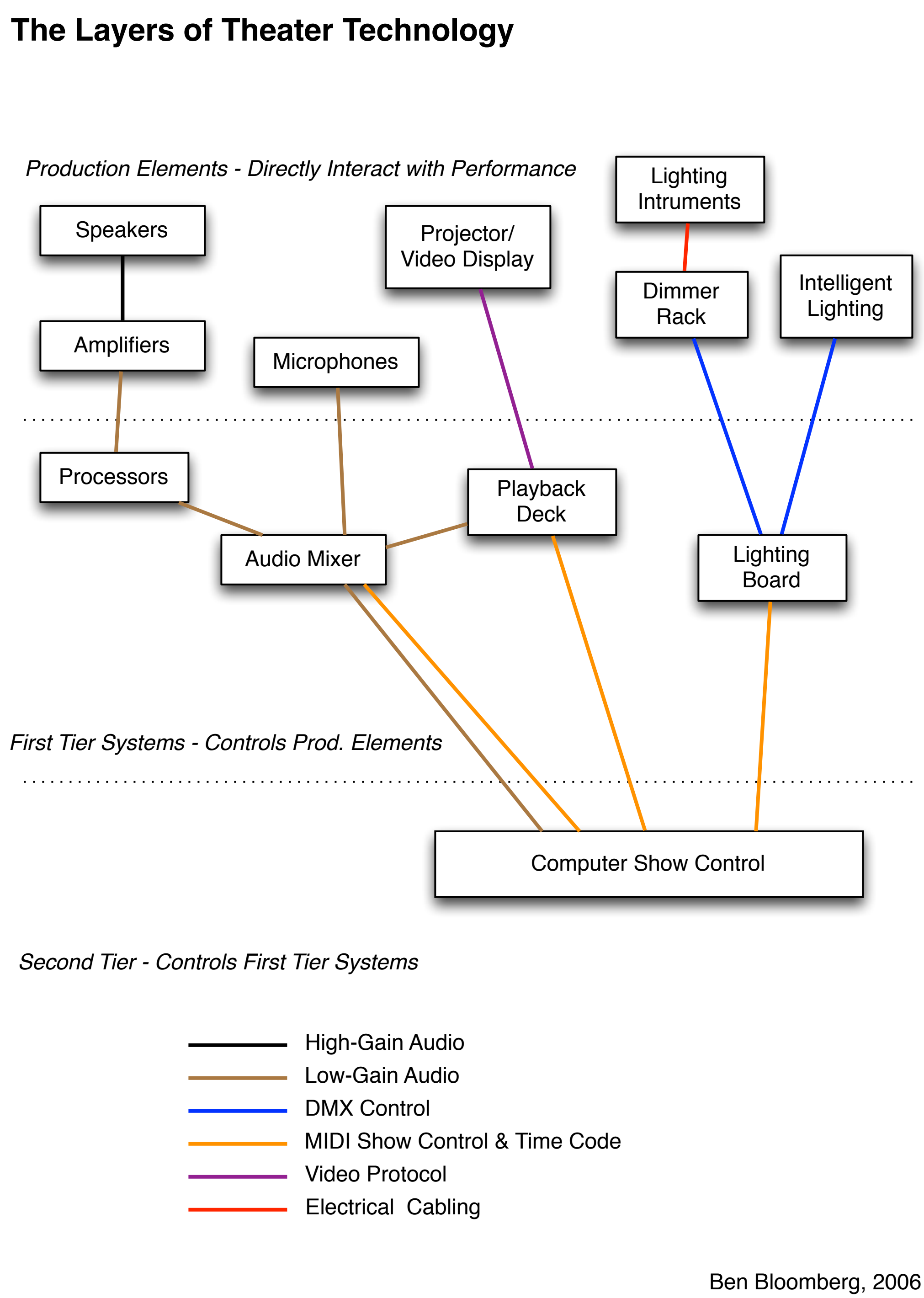

Figure 3.1: A typical arrangement production elements, first tier and show control systems

in time. An interpolation method handles any state of the system between keyframes. Many visuals and post production systems work in this fashion, where the timeline is fixed and the state of any element in the system can be derived by looking at the two adjacent key frames and the interpolation method between them.

3.1.4 Tracking

Tracking is a term that effects a cuing system's behavior when cues are modified. When the state of an element is changed in a cue, it is possible to have that changed state track through all cues following the edited cue. This essentially makes the change in that cue "stick" throughout the rest of the production. In certain cases, a change in a cue should be reverted when the next cue happens or it should only be kept through a certain subset of cues. The term tracking refers originally to the stage-manager's ability to track changes to blocking and set movements through a script. Production control systems need to emulate this capability to be effective during programming rehearsals.

Certain production systems are not capable of this sort of modification. Often systems for controlling the automation of set and rigging systems are very inflexible in their cue editing capability. This means that a single programming change can sometimes take serveral hours because the system must be run through each cue in real time to verify the state of a single paramater in each setting.

3.2 Time Code

Cues represent synchronization points where many elements must occur exactly at the right time, but after a cue is triggered, elements run on their own time-line. For example, a set change made by humans, a lighting fade to black, and a sound cue playing from tape may drift in synchronization even if all three are started at the same moment.

With the advent of tape and talkies, a new standard was developed for synchronizing not just specific moments in time, but all moments in time between multiple systems. Time code was first developed to synchronize projector and audio playback so that the two remain locked together continuously; the audio and video do not drift out of sync over time because the playback rate is kept the same between both systems.

It may seem like playback rate should not drift this way, but in reality many systems use different methods to keep track of time. Small differences in playback speed accumulate over time, especially in the era of tape and celluloid. When the "clocks" controlling the speed of devices are not perfectly synchronized, time-flow is not the same across multiple systems. A drift of even 20 milliseconds in audio (less than a single frame of 30fps video) is quite noticeable, so time code systems became widespread especially when dealing with audio.

It wasn't long before time code began to be applied to production control systems for live entertainment as well as post production and cinema. With it, it became possible for elements of a production such as playback, lighting, and video to be very tightly synchronized over longer durations.[23]

3.2.1 Speed vs. Location

It is important to understand the difference between timecode and clock synchronization. There are protocols which synchronize the speed of two systems exactly. These are clock protocols which can ensure that a frame, beat or tick is the same length between systems. Examples of clock protocols are MIDI Beat Clock, Word Clock, Tri-sync, Black burst, and Loop sync. This is enough to prevent drift between multiple systems, but to ensure complete synchronization they must all be started at exactly the same time. Obviously this becomes increasingly difficult as the number of systems increases (can you push 47 play buttons all in the same millisecond!?)

In contrast, a timecode protocol does this synchronization but also gives sys-

tems an exact timestamp, so not only are they at the same speed, but also the same point in time as well. Timecoded playback systems can be started independently at different times and each one immediately jumps to the correct location and speed.

3.2.2 SMPTE Linear Time Code

SMTPE LTC is the most popular timecode format used today. It is a binary coded audio signal which uses a manchester-like encoding to send full timestamp of each frame. LTC is transmitted between systems just like any audio signal, digital or analog, balanced or unbalanced.[23]

Modern systems reading LTC often require special hardware to recover a digital audio clock from the LTC stream. This ensures that any digital audio converters in the system are sampling at the same rate, ensuring the systems run at the same speed.

3.2.3 MIDI Time Code

MTC is transmitted via MIDI equipment and contains the same information as LTC. Support for MTC in modern Digital Audio Workstations, sequencers and playback systems is generally better than LTC because a synchronized audio clock is not required to decode MTC. This does make MTC systems more suseptible to drift, if they are poorly implemented. This is because a system must dynamically resample its output to the incoming timecode and ignore its own system clock, essentially chaining the speed of the output to match the timecode and not the system clock. That is not a trivial prospect and most systems are not capable of such fine grained resampling.[23]

3.3 Performance with Cues and Time Code

All of these methods of control have had quite an effect of live performance over the years. As one can imagine, it is these time code and cue-based systems that are the oldest and thus, the most robust. When it comes to large-scale, high budget productions, producers are loath to trust any systems that are not tried and proven. Because these types of control are the most common, tested, robust and well matured, much thought has been spent on conforming live performance to work within the bounds of the unique requirements.

For live music performances, often the drummer is given a click track or strobe to follow the tempo of the running timecode exactly. In other cases, a conductor is responsible for listening to both a backing track and a live orchestra and must try her best to keep the two closely aligned. Another common arrangement is that a stage manager or musician will trigger preset materials at a specific moment to try and align it with the live performance. Once triggered, the live performers must not change speed. In essence, the one "system" that can never natively accept time code are the human performers.

Still once a performance is properly imagined along these lines, both methods of control provide a great deal of capability for live systems and experiences. But as we will see, they each have strengths that do not overlap, so we are left with a gap in capability and functionality that is hard to fill with these existing methods. Let us first look at two categories where existing systems excel:

3.3.1 Nuanced Timing

With a human calling cues, it is possible to acheive quite nuanced timing with relation to the action on stage. A cue may be called in a pause between two words, whether a performer has skipped the line before or not. A followspot operator can keep the spotlight trained on a performer even if the performer decides to make a change in his or her blocking. A sound mixer might know to mute a wireless microphone before a hug each night, although the hug does

not happen exactly the same moment.

With a human operator in the loop, performers can be fluid in time and not worry about production elements following along. Often this results in a compelling and natural performance. Humans focus on the elements of performing that are most important to the audience's experience of the piece rather than logistics or mechanics that uphold suspension of disbelief or some other effect.

3.3.2 Nuanced Synchronization

With timecode, an experience may be refined and perfected like a film or album because each run of the system is identical. It is possible to have hundreds of separate components perfectly arranged together and intertwined in incredible complex ways.

3.3.3 A Musical Approach to Timing and Synchronization

Let us take a step back and imagine the way an orchestra performs. Often there are well over 100 people working together to excute what amounts to a compelling and intensely complex sequence of actions resulting in (beautiful) music. The combination and output is powerful and compelling; it can bring one to tears or laughter. All the elements of an orchestra are perfectly synchronized often with such careful nuance that it is almost impossible to quantify exactly what makes an orchestral performance so moving.

Unlike timecode and cue-based systems, orchestras are able to have both nuanced timing and nuanced synchronization. No two performances are quite the same, the humans focus on the experience and on the performance, rather than being exactly aligned. Yet, for all intents and purposes, the many individual elements are exactly aligned. But as well as being aligned they are flexible: the orchestra can pause an indeterminate amount of time, and with a single breath or upswing of the conductor's baton, a whole new tempo or mood is

set.

To achieve this sort of tight synchronization and flexible timing with digital production systems, many have tried to combine cues and time code. Other approaches involve changing the rate at which time passes using a hardware or software interface to slow down or speed up systems to follow the performers. In some cases this can work well, but the fact that there is a fixed timeline, no matter its speed, makes it impossible to skip if the material before is moving too slowly. This fact, combined with the notion that these types of systems are often unresponsive to the performers, makes the whole approach to systems result in quite artificial feeling experiences.

Many of the Opera of the Future group productions have created custom solutions to address these issues by discarding timecode and building systems based around cuing and interactivity. By attaching production elements to live input from the performers, a system can be tightly synchronized while maintaining its flexibility. There are many challenges associated with interactive production systems; the biggest being that is it hard to constrain the behavior of such a system. A designer, producer or director wants to know that the system will not have unexpected output. Artistically, it is important to make these systems limited so that they maintain some artistic intent instead of being completely random. An important aside— in Opera of the Future we consider random output or input to be not useful for most types of production, where the production elements should be somewhat connected to the performance happening by the humans on stage. If it is impossible to understand this connection between human and technology, the next question is often why they two should be associated at all. Some types of installation and performance art ask this question explicitly, but much of the work in Opera of the Future is focused on using technology to aid in story telling. In the best productions, the audience does not concentrate on the technology involved or it's relationship to the performance, only the emotion being communicated.

In the next chapter, we'll explore the Opera of the Future group approach to designing systems as instruments. An instrument is reactive, yet constrained

and incredibly nuanced. An instrument is a true medium for expression. I argue that production systems can and should be designed to work at the same level of expressivity.

Chapter 4

A History of Opera of the Future Production Systems

In the Opera of the Future research group there are many examples of novel production systems that do not follow traditional control paradigms. Indeed, the original concept of the Hyperinstrument challenged many of the standard assumptions in live performance technology. In this section, we present a summary of the different systems created by Opera of the Future and used in performance. The later systems are responsible for much of the motivation leading to the creation of the Hyperproduction framework. These will be discussed in more detail to set the stage for the conceptual elements that underpin the platform presented in this thesis. We will look at each of these systems from a production perspective. Later on, I will argue that the best production systems are performative in nature and the difference between production and performance perspectives is small.

4.1 Hyperinstruments

The Hyperinstrument was the group's first foray into unified production technology. The motivation behind the project was to allow virtuosic performers the ability to produce experiences with great complexity in a live performance setting. Previously these experiences could only be achieved with post-production in a recording studio because so many elements had to be carefully

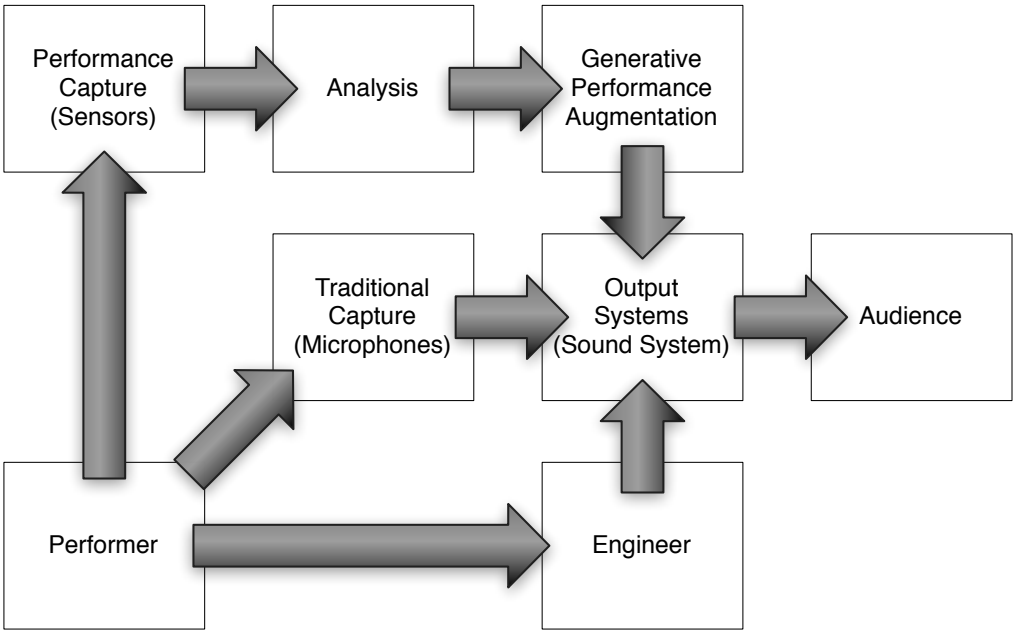

Figure 4.1: Basic components of a Hyperinstrument system

crafted and fit together. The aim of a Hyperinstrument was to give performers control over these elements via an interface which was already a natural extension of their expressive tendencies.[33]

Over the years, Opera of the Future has crafted many of these systems for live music performance. In the systems have the following components shown in figure 4.1.

4.1.1 Performance Capture

The system has a set of sensors which capture the performance. These sensors take a variety of forms; accelerometers, IR beacons, microphones, resistive or capacitive sensing, cameras, RFID, etc...

The original Hypercello used a combination of capacitive sensing on the finger board, IR proximity sensing between the bow and the instrument, and accelerometers on the bow to capture movement.[32][35]

Sensing and capture of performance elements is a broad field and is outside of the scope of this thesis, but much of the capability of a Hyperinstrument system is tied to the quality of data received from performance capture mechanisms.

4.1.2 Analysis

Once there is captured data, analysis provides a way of creating "meaning" from that data. We cannot extract this meaning directly because the data from sensors is often imperfect. For example, it may be necessary to understand how a bow is being moved. An accelerometer will only provide a measurement of the bow's acceleration, so more calculation is necessary to understand position or speed. The best analysis systems take data from sensors and provide a technological representation of the performer's intention. It is this representation of expression that is translated into control for additional layers of the performance. There are many types of analysis techniques ranging from simple scaling to FFT to very complex classification systems taking advantage of machine learning techniques.

Many of the original hyperinstrument systems were created with one or two very basic methods of analysis.[32] However, we have found that different analysis strategies work well in different scenarios– it is most advantageous to use a combination of systems.

4.1.3 Generative Performance Augmentation

Once there is some representation of the performer's intention, a set of systems take that representation and actually create additional content to the composer's specification. Content can be created from real-time input (i.e. transformed audio) as well as from playback of pre-authored material (sampled audio or MIDI). Pre-authored content can also be transformed in real time. Because the representation is constantly modified by the performance, the content is generated in a way that is closely connected to the performer and their

performance.

4.1.4 Output Systems

Output systems are responsible for communicating the additional content and the performance to the audience. A large sound system may deliver a reinforced version of the performance combined with all the augmentation created by the hyperinstrument system. The design and creation of these output systems have a large effect on the aesthetic of the piece, however their design is not compositional the same as the generative performance augmentation design. These elements are similar to the type of canvas or paint an artist might choose when starting a new work.

4.1.5 The Role of the Engineer

In all of the hyperinstrument systems there is a role which is often not discussed. For the best result, an engineer is in the loop shaping and guiding the hyperinstrument system. In many cases this role is filled by the sound mixing engineer, but in certain cases there may be other operators (lighting, visuals, etc.) as well.

These operators are not strictly necessary; it would be possible to create a system with enough complex analysis and automation that the piece could run itself completely without interaction from anyone but the performers. However, in practice this degree of automation isn't realistic and time is better spent in other areas, given a tight production schedule.

The engineer provides other benefits as well; the rehearsal becomes much more flexible when a person is there to make small corrections and changes in the moment— it becomes possible to orchestrate much more complex interactions. The most important benefit of the engineer is to allow the process of producing and rehearsing the piece to remain a strictly "musical" endeavor. It is hard to exactly quantify this quality of rehearsal and performance. To

put it in a simplified way, an engineer has extraordinary capture, analysis and augmentation capabilities. In a scenario where technological capture, analysis and augmentation systems may not be completed yet, an engineer can provide a bridge toward a completed experience that allows the rehearsal process to retain the feel of a music rehearsal. When the systems are completed and the performers are used to them, the engineer often does less, but any abnormalities can be taken care of in a uniquely flexible way that even the most advanced AI and machine learning could not accomplish.

4.2 Opera of the Future Production Control Paradigms

These control paradigms have become standard architectural building blocks in our system designs. Previous productions and pieces (including most hyperinstrument pieces) have each included one or more of these elements. They augment the traditional methods described in the previous chapter.

4.2.1 Piano Keyboard Automation

Perhaps the most common production paradigm is that of piano keyboard automation. The performance is represented as a collection of patches. Each patch contains parameters for one or many synthesizers and samplers. Patch changes are notated throughout the performance in a score or script and recalled by the player.

This type of system is found on almost every modern production but the granularity of the patch parameters varies wildly. The system currently used on many Opera of the Future productions uses a virtual rack to run components of the system that are shared between all patches. A "chunk" keeps track of parameters and components that change in each patch. The chunk contains audio and MIDI routing, a mixer and a time-line which is completely isolated to that patch. Chunks can be recalled by the keyboardist at any point in the

production.

Another complex aspect of keyboard automation is the patch transitions. Modern automation systems are able to intelligently cross-fade from one patch to another, or even let two patches overlap. This is critical to ensure that there are not artifacts, clicks, pops, or abrupt cutoffs when patch changes occur during a musical phrase.

4.2.2 Modes

A mode system is a generalized version of a Keyboard patch system. Modes are often applied to change the behavior of a system completely. In Opera of the Future productions, mode-based systems are often used when the same input devices must serve very different roles. A mode might be created for each segment of a piece to allow similar production elements to change interactions and behaviors. In the case of a hyperinstrument system, slightly different analysis and performance augmentation may be configured in each mode.

4.2.3 Triggers

Triggers allow small pre-authored segments of content to be replayed. These segments are intended to overlap slightly and are granular enough that they can synchronize with live performance at tempos slightly faster and slower. Generally, each trigger is notated in the score as an instrumental part, and it is cued by the conductor along with the orchestra so that the segments are well timed with the rest of the players and feel organic in the context of the music.

Triggers often control content that may not be specifically musical in nature. However, visual elements, real time effects, automation, lighting and other systems can be triggered musically in a way that feels more connected for a large ensemble than if time-code or stage management or other traditional means are employed.

4.2.4 Mappings

When sensors or continuous controllers are involved (knobs, stretch sensors, tracking systems, etc.), control of production systems takes an entirely different form. Mapping systems provide a toolset for working with continuous data to allow translation and manipulation of any inputs into appropriate control sources for output systems.

The first general purpose mapping system created for Opera of the Future was Joshua Strickon's SecretSystem[40] which could integrate control over a variety of lighting, scenic, audio and robot elements based on input from sensor data. SecretSystem was a perfect tool for creating simple mathematical relationships between sensors and production systems.

The concept of the mapping system was refined for *Death and the Powers*, where many continuous data sources from the stage, performers, and orchestra needed to be manipulated and combined to control all aspects of the production on a much larger scale. The key element of the system for *Powers* was the fact that rather than mapping input directly to output, an intermediate model was used to abstractly represent state within the piece. We will explore this concept in much more depth the next chapter. This system was developed with the name, Disembodied Performance System[44], since it was created to allow the main character in *Powers* to perform without being present on stage. Despite the name, the system became applicable in many other contexts as well. Several modern hyperinstrument pieces[17][26] were created with the same system, with the DPS components completely replacing the analysis stage described in section 4.1.2.

Mapping systems often contain a great deal of analysis and can be thought of as an analysis toolkit. The Disembodied Performance System works only with continuous data, however, so analysis for strings, booleans, or other complex data are currently impossible.

4.2.5 Distributed Control

Distributed control is becoming more prevalent as internet-based performance grows more popular. There are a variety of ways of assembling the previous building blocks. Originally the Open Sound Control protocol (OSC) provided a flexible and simple way to connect all of these systems together on the same local network. Unfortunately, OSC does not route well over the internet. Most OSC implementations are UDP based, and TCP OSC does not have session management. Websockets are quickly becoming a viable transport for performance data over the internet. For the recent production, *A Toronto Symphony*[1], OSC was utilized as a communications protocol to control lighting elements on and in the CN Tower. OSC was sent directly from the orchestra to the tower via the internet. While OSC via TCP should have been a viable transport— in practice no commercial production systems supported TCP OSC well. Tools for understanding the state of the link from the hall to the tower were not dependable. Ultimately, an SSH tunnel was employed to wrap the OSC link to provide some sort of encryption and session management. The final system flow is shown in 4.2. For later projects, Websocket implementations in Java, NodeJS and Python provided session management, authentication and encryption without having to use an additional tunnel to ensure reliability and security.

A secondary example: for the 2014 Sochi Winter Olympics, the NBC broadcast of all curling events was produced out of a control room New York City. Jpeg2000 video, uncompressed audio feeds, and control data to operate cameras remotely were sent back and forth over a 10Gbps network connection from the Olympic venue in Sochi to the US. Being able to utilize a crew and production facilities in the states freed up resources in Sochi for larger and more complex events.

Distributed control allows systems to be situated in locations all over the world, contributing input and taking control data for a unified performance. Distributed control also allows multiple incarnations of a performance to be as-

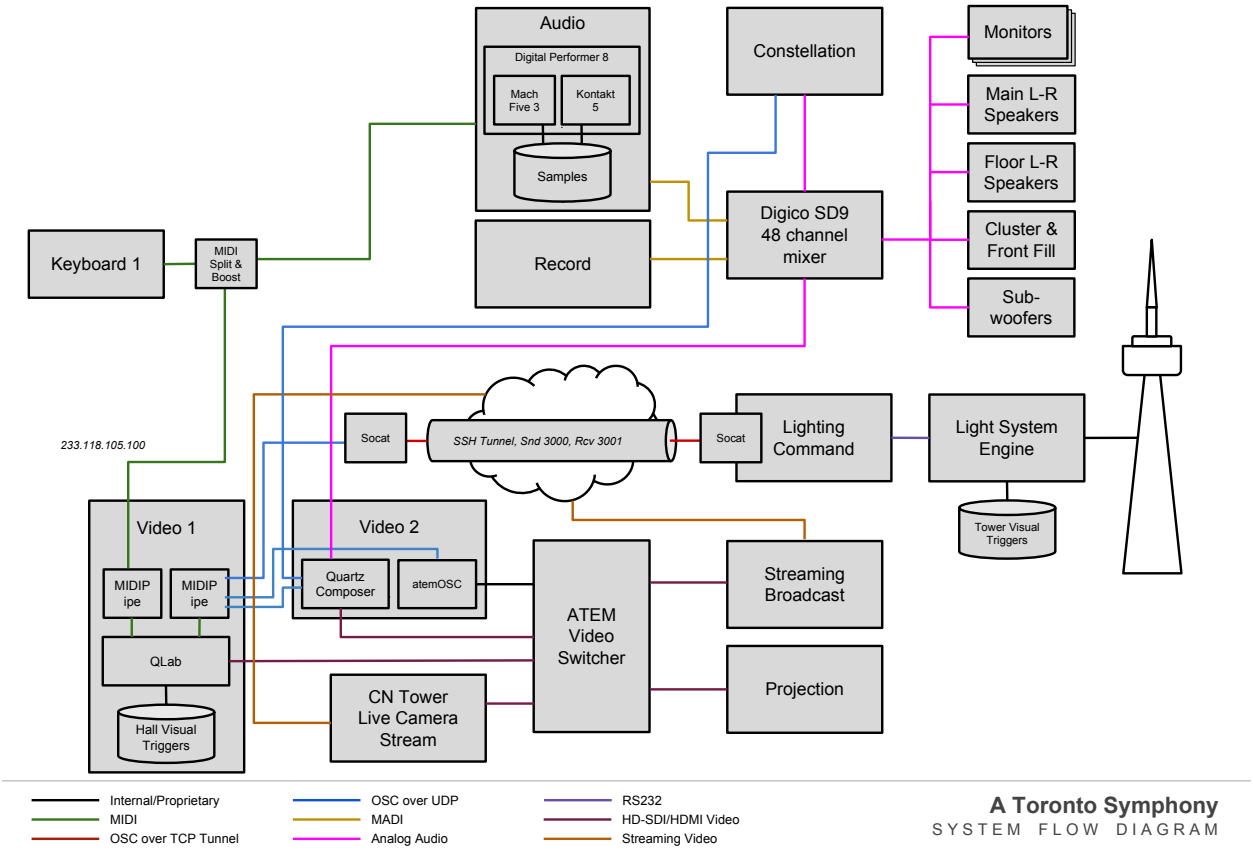

Figure 4.2: System flow for *A Toronto Symphony*

sembled in different ways in different places. By publishing the data used to create or augment a performance and all the component pieces of the performance we can "render" a single experience for many different output mediums.

4.3 Death and the Powers

Death and the Powers was a project originally commissioned for Prince Albert, which premiered in Monte-Carlo in 2010. This production encompasses all of the building blocks outlined in the previous section.

4.3.1 The story

Powers is an opera about a man, Simon Powers, who wants to live forever. He designs and builds "The System" with his research assistant Nicholas, which allows him to upload his essence and cease his existence in the world of the flesh. Once in The System, he slowly becomes his surroundings until he is an all-inhabiting, all-controlling presence that no longer values what it means to be human.

The world descends into chaos as he tries to convince his family to join him in the digital realm. World leaders make an attempt to appeal to Simon to fix the problems created by his companies and holdings, but he does not listen to them. At the end of the story he tries without success to convince his daughter to join him in The System.[37]

4.3.2 The Production Systems

When Simon dies and goes into the system at the end of Scene 1, in reality the singer goes down to the orchestra pit, enters a sound isolation booth, and puts on several sets of sensors that measure aspects of his movement and breathing. At this point he continues the rest of the piece hidden from the audience, but

the sensors and microphone he uses translate his performance to all aspects of the production technology; lighting, sound, visuals, robots, scenic automation, and video are controlled based on mappings from his sensor and audio data.[27]

The Keyboard automation system has 47 patches which recall parameters across six synthesizers and samplers. The production has 8 master modes that set up different rendering presets, banks of samples for audio triggers, and visuals presets. Each audio trigger optionally recalls a visuals mode as well. The visual system has well over 300 modes and mappings that change by triggers from the orchestra. In the case of this production, a patch change in the keyboard automation system, a note from another player in the orchestra, or manual cues can be used to trigger audio, video or mode changes on any of the systems. The score represents a set of synchronization points that are used to alter the treatment of real-time information such as audio or sensor data, or to trigger the start of a set of authored content that is altered in real time. In a sense, the entire production was a large hyperinstrument.

Components of the production system are distributed across several networks that allow the various systems to exchange data with OSC[5]:

- The audio system receives tracking data from a UWB RFID system to control tracking, effects and panning.
- Audio data is sent to the mapping system where it is scaled and passed to the robot system to modulate on-board lighting.
- Sensor and audio analysis from the mapping system is sent to the visuals rendering systems and robots to influence on-board lighting and articulation
- Triggers from the orchestra are relayed to the Keyboard Automation, Mapping and Visuals systems.

- Mode changes from the triggers system and keyboard are communicated to all visuals systems and keyboard automation system.

- Tracking data is communicated to the robot control system to influence the movement of the walls.

4.3.3 Audio Systems

The full audio system for *Powers* consists of a proscenium spaced array of approximately 20 loudspeakers which provide a traditional frontal image for sources on stage. An additional 60-80 loudspeakers surround the audience driven by a 3rd Order Ambisonic system, which has 52 sources that are spatialized around the audience throughout the production. A 23 foot speaker on the lip of the stage has 64 four inch drivers each individually powered and controlled. The system uses a Wave Field Synthesis algorithm based on Evert Start's work at Delft University[39].4.3

The audio system for *Death and the Powers* relies heavily on real-time data. Much of this data is provided by the UWB RFID tracking system, but an even greater portion is provided by the mixing engineer, in particular, me.

4.3.4 Mixing Powers

From a mixing standpoint, Powers is *operated* much like a Broadway production. We avoid the "typical sound" of Broadway by very carefully choosing our equipment, speaker locations and mixing aesthetics. That said, each performer and instrument in the orchestra has a dedicated microphone much like a Broadway production. Along with digital inputs from computer playback, surround and monitoring, the mixing engineer is responsible for 350 audio inputs and 250 audio outputs in total. Since I only have 10 fingers, we use VCA automation to keep track of the inputs and outputs in a manageable way. The mixing console has 9 modes which pull necessary groups of faders to the right place on the console at each point in the production. The modes also control

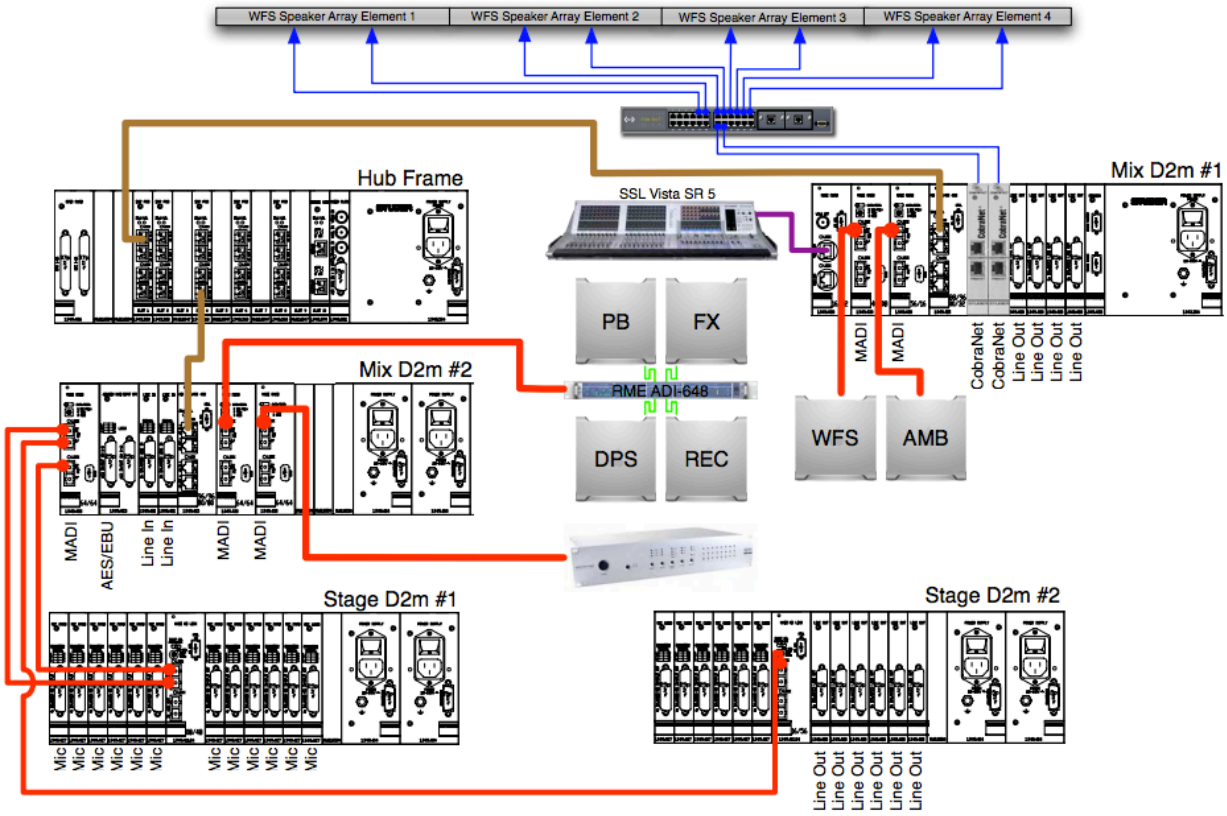

Figure 4.3: Death and the Powers Audio System

the panning automation in the surround system.

For each mode, the active microphones are placed on the surface under my fingers. At any given time, only a single microphone is turned on on stage. This is because opera singers are so loud that having any other nearby microphones turned on causes bad phasing and comb filtering caused when one source is picked up by multiple microphones.

As well as singer microphones, there are also faders for the orchestra, surround sound, playback and effects. These faders group many microphones together.

Given that there are many lines and the production moves relatively quickly, as the mixing engineer, I must have the entire production memorized and I know exactly how loud each performer performs each line of the piece. I bring up a fader for each line to the correct position and make sure the electronics and orchestra are at the proper level. Like the engineer in a hyperinstrument context, this could be thought of as a musical performance almost as much as the players in the orchestra. Each night the piece is played differently and I must adapt to the changes in performance the same way the players and singers do. As I improve, I get to understand nuances like lines that always change dynamics (i.e. the beginning of a line might be very quiet and the end much louder). The more nuanced I am, the more effective and transparent the audio sounds in the hall.

With this system (and almost every system I design and mix on) the goal is transparency and natural sound. We aim to have quiet sections of the piece sound as if they are unamplified; this does not mean there is no amplification. Rather, we strive to create an environment where the audience member forgets that the sound system exists, every word is crystal clear and the system does not add unnatural timbre to voices or instruments unless it is intentional.

As with the hyperinstrument system, an engineer provides an advantage that is nearly un-reproducible by mapping, machine learning, or other technological system in this case. It allows for almost infinite flexibility; although a system could be devised to "automix" the production taking into account the

score and lines and orchestra, it would be impossible for such a system to be as nuanced as a person listening. When the goal is nuance and emotion, a person is a critical part of the process. For simpler scenarios– offline processing, basic multi-source speech– it is possible to use an automated system.

Many systems[13][24] have been created to accomplish basic mixing like this. None have been demonstrated to correctly handle the demands of a live stage production.

4.3.5 Listening, Imagining, and Reacting

But what exactly does the mixing engineer do to create such a nuanced mix that an automixer cannot achieve? In many cases, he or she uses a significant amount of imagination! Often this involves conceiving a mental model of the location and treatment of sources, elements in the mix, to help determine how and where every input should be placed for the audience to experience. For example: we could imagine that each element of the mix— each instrument— reacts with the others and with some artificial space, maybe very large or very small. Changing the relative size and location of elements causes our attention to be drawn to them differently.

If a background element (ex. a soft violin) suddenly becomes very loud, we have the option (as the mix engineer) to let it rise to the front of of the audience's attention or to reduce its focus, either by making it smaller or pushing it further away.

All of these abstract qualities (size, space, location and distance) contribute a great amount to the feel of the mix and the engineer's ability to keep many sources discretely placed for the audience to appreciate. The engineer modulates them with great nuance like a player manipulates an instrument.

4.3.6 Hyperproduction

If we could take this extraordinarily nuanced modulation of these abstract qualities, the information that the mixing engineer is encoding already, and make it available the way performance data is used in a hyperinstrument system, it is possible to imagine a true mixture of human and computer collaboration where the level of detail and nuance is much greater.

Rather than having the computer create relatively naive responses to the performance itself and have an engineer to make sure the output is appropriate, we should give all the systems in the production access to the engineer's intuition first. Using this topology of interconnection, it might be possible to have many systems following a piece the same way a mixing engineer does.

4.4 Sleep No More

Sleep No More is an immersive theater experience based on Shakespeare's Macbeth and Hitchcock's Rebecca. Typically, audience members experience the performance by walking through a 90,000 square foot space on 6 stories of an extremely detailed set, while the cast members have interactions and play out scenes, sometimes involving the audience. The audience members wear masks and do not speak for the duration of the 3 hour event. There are 20 hours of content in the performance. Users cannot see everything in one night and are encouraged to make their own version of the performance each viewing by exploring and following characters they find interesting.

Our research group teamed up with the creators of Sleep No More to try and understand what it would be like to recreate the experience of Sleep No More for online participants at home. The goal of our collaboration was to build the required media distribution channels so that it would be possible to experiment by linking those at home to the world of the show with very low-latency video and audio broadcast over the internet.

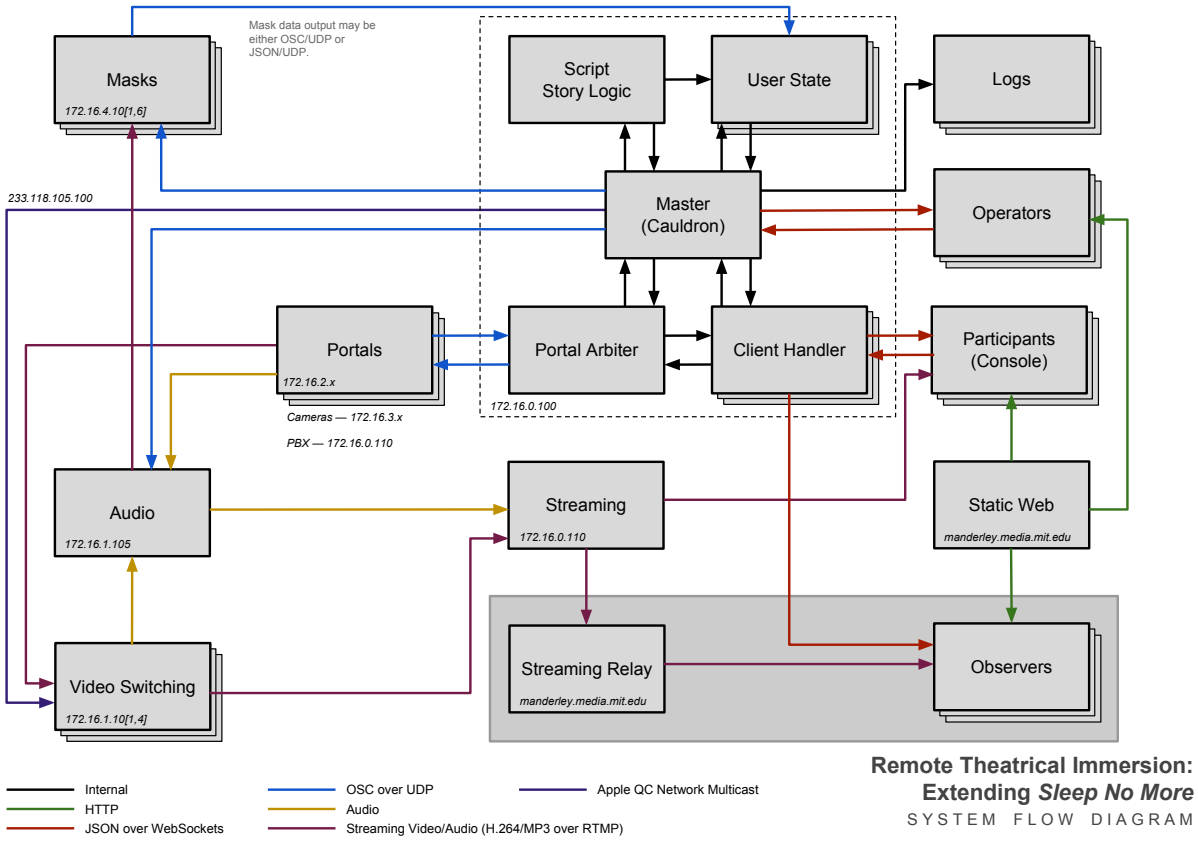

Figure 4.4: Sleep No More Internet Extension System Diagram

This could not be achieved with continuous high-bandwidth connection; this approach was deemed impractical and ineffective from an artistic standpoint. With a carefully curated selection of experiences using text, image, video and sound, we can create the illusion of a much larger, more vast and limitless experience by revealing less to the online user initially.

As a result, it was necessary to be able to orchestrate video, sound, text and imagery very precisely and with low latency for the online users and at times, for audience members seeing the performance on-site as well. A system overview is shown in figure 4.4.

4.4.1 Story Logic and User Console

A master story logic system controls all aspects of the performance experience for both online and on-site participants. It sends control messages to all systems to ensure visual, aural, physical and virtual experiences stay synchronized with the script. The engine is flexible so that the story unfolds in a non-linear fashion.

Online, the user interacts with a web-based console which provides text, images, video and sound depending on the story and how online and on-site participants interact with the system.

4.4.2 Portals and Masks

On-site, the participants wear a mask that has been instrumented with an Android device to deliver sound through bone-conduction speakers and to collect biometric data.[45] These masks lead the participants to various portals throughout the space. Portals allow online and on-site participants to communicate. The bandwidth of this communication varies greatly; some portals are simply a book flying off a shelf with specific words circled. Some send video or audio to the participant on-line.

4.4.3 Distributed Systems

Sleep No More represents an entirely different type of production system. Instead of the audience sharing a performance venue together, everyone was separated and experiencing the piece in their own homes on a computer by themselves. In an attempt to recreate the feeling of individual agency— is a key characteristic of the original production— those at home and those in the space were able to wander relatively freely around the their respective worlds. In the case of the online experience this world was virtual, meaning that both time and space could be altered with little consequence.

As a result, the production systems for this project had to take into account

the real time-line of the participants in the live production while constructing a world and series of events for the people in the online experience. In certain ways, this is equivalent to creating a full production that is completely customized for each audience member.

To achieve this, all the systems for this production were designed to be multi-input, multi-output. The mixing console was software based and could generate separate mixes for every participant whether online or wearing bone-conduction headsets in the physical space. A video system was also capable of generating output separately for each audience member. Both audio and video systems could take live input from a number of sources around the live space, from telephones, and from pre-recorded material as well. The material could be altered separately for each participant in real time even if it was pre-recorded playback.

In certain cases, having many audience members with individualized experiences created certain constraints that needed to be satisfied, such as live spaces that could only be used by one person at a time. These sorts of constraints were very much based in real time and real space and they also imposed constraints on the the virtual experience.

At the heart of this complex orchestration of real and virtual experiences was a control system, the Story Logic, which kept track of what was essentially an individualized score or script for each audience participant. As the audience made decisions, especially in the virtual realm, the score was able to rewrite itself.

The Story Logic also had a knowledge of participants' locations in the physical space as well. This allowed special behavior when two

4.4.4 Idempotent Triggers

Like many of our other projects, the "score" or Story Logic had a number of triggers associated. These triggers were slightly different from those of other

systems such as *Powers*, in that they were more like switches, enabling and disabling parts of the experience. When called, a trigger was assigned to an audience member. Because of this arrangement, triggers could be applied in almost any order. When the audience entered a new part of the piece, special triggers could disable all other running triggers to ensure it was possible to reset to a known state.

4.4.5 Operator as Performer

The online experience for Sleep No More was loosely based on classic text-based adventures. A goal of the experience was to ensure that participants never felt like they were interacting with a computer, so the system would never respond "command not found" or "please try again". Instead, at any unknown input, the system would involve a human operator to intervene and send back appropriate responses.

This is another case where it would have been possible to devise a very complex natural language processing system to try and account for all possible user input, but the goal of the project was to create an incredible user experience, not to design a bullet-proof NLP engine. In our case, having a person in the loop provided far greater nuance, emotion, and satisfaction. Like the mixing engineer in *Powers* and for Hyperinstruments, the Sleep No More operator was a uniquely performative role. As someone interacting with audience and shaping the experience, the operator had to know the story and the physical and the virtual spaces completely so that they could improvise creative and appropriate responses to any sort of interaction.

Chapter 5

Modeling Advanced Performance

In this chapter we'll discuss Sleep No More and the most recent production of *Death and the Powers* in detail. Both productions took a unique approach to production control which served as a testing ground for basic Hyperproduction systems. Rather than focus only on the nuts and bolts of each production system, we will look at the conceptual model of the production— a model that the production systems were able to replicate in the digital domain— understand how operators were included in the model, and how this ultimately resulted in a more powerful experience for the audience.

The nature of production often makes it difficult to fully test a new system in all the ways that one plans originally. There are so many moving parts to a production that unexpected interactions between all the parts often cause systems to change drastically from their initial intention. This makes live production an extraordinarily hard environment for conducting research, because it is quite difficult to propose a project and see it through exactly the way it is proposed. Ultimately the artistic nature of the production and the experience of the audience must take precedence over any technology. Artistic vision often evolves quite organically and more often than not technology must similarly evolve for the result to be satisfying.

5.1 Sleep No More

The structure of *Sleep No More* is unique from a traditional stage production in several ways. First and foremost, the production has no stage; instead the audience members wander around a set of 100 rooms. Because of this, it is impossible for an audience member to see all the action happening in the production at any given moment. As a trade-off, however, audience members are given some agency in what they would like to view during the evening. One can decide what to do and where to go. Significant events happen more than once to give more audience the chance to be part of special moments. This works particularly well because certain portions of the experience happen only for a limited number of people at a time.

A conceptual model of the performance must keep track of all of these pieces:

- what is happening at various points in the production in each location
- which experiences happen only for limited numbers of people
- when rooms are available to enter or are locked up

Another unique aspect of Sleep No More is that despite all of the agency afforded to those in the experience, the entire production is constrained quite heavily by time. In essence the entire production is on time-code and it is impossible to linger in a moment or be fluid with timing. An incredibly nuanced 19 channel sound track gives performers all their cues for the entire duration of the performance.

In summary, the production can be understood fully by referencing a large spreadsheet of time-slots, locations, scenes and characters, a point in the sound track, and a cue stack on the light console. These items, used along with the current time, specify an exact state of the performance for every human and technical production element every second of the show. Of course within the time-slots, ranging from seconds to minutes, it is possible for the live perform-

ers to improvise. At times they must make decisions about which audience members to include in smaller segments of the performance as well.

Creating an online version of the production affords more flexibility and agency to the participants, who are not limited by the physical constraints of a real-life space. In the online world, it is possible to teleport, travel back and forward in time, and even alter the flow of time. One of the biggest challenges in creating production systems for the online version of the show was to determine the best means of modeling this new kind of world. It poses a unique challenge in keeping track of the state of a performance because, in the most basic way, each online viewer could be in a completely different state depending on his or her decisions and actions in what was ultimately a fluid environment.

The online production systems had to be able to deliver a completely tailored experience based on the same world, but fractured to different states of existence as soon as a participant enters it. This sort of model and interaction is common in computer games, but the main difference between a game experience and this one is the notion that all decisions and actions a user takes must be sent to the production to ensure they are possible in the context of the live performance and the space. For example, two users could not exist in certain virtual rooms if their partners in the real world could not as well. At moments in the live performance, real-time video and audio is sent to participants online.

So given this, online production was a single production but the very definition of a single production— what it means for participants to be sharing an experience together— is not so clearly defined.

How do we create a model that encompasses these types of productions; where the world is multi-dimensional and people explore at their own pace, where certain interactions must be constrained but others may be completely unrestricted and the systems creating the experience must be able to create many similar but customized versions of the same production?

In the following sections I will describe the approach taken in detail from a

modeling and operation perspective. Once the concept of Hyperproduction, the system created for this thesis, is clearly defined in following chapters, I will return to this production to explain the advantages hyperproduction has over the implementation used here.

5.1.1 A Markup Language for Multi-dimensional Production

The first challenge is to describe the world of the production explicitly, defining all paths a user might take through it— what is possible to access when and how. The story is intricately tied to this description, almost like a script that contains scenic descriptions. This approach is the reverse, a vast scenic description instead contains the story that exists in each part of the described world. To achieve this level of detail an XML markup language was developed and employed (JEML, affectionately named are its creators, Jason and Elly) to contain and describe all the required elements. Here it is possible to see a description for a world that rebuilds and reworks itself based on the actions of the user. The main elements here are items, actions, and events. Events may cause items to appear and disappear, actions to enable or disable, and media assets whether live or pre-recorded in any medium to play.

An example representation of a room is shown in figure 5.1.

5.1.2 Production Elements in a Virtual Show

Video and audio systems are triggered based on the asset and event tags. In the case of the audio system, a pool of players and effects is allocated, started and routed to users when needed for a specific part of the experience, then resources are returned to the pool. Video, unlike audio, uses dedicated and reserved streaming and processing for each online participant. Ultimately for these experiences to work at scale, all systems must be able to function like the audio system, in pools. This provides a method to dynamically scale the system while keeping requirements for computational power as efficient as possible.

```
<room id="veggie_patch_grave" room_number="3-3" locked="false" name="a graveyard">
  <short_description>a small graveyard</short_description>
  <long_description>
    A cobblestone path winds its way through a small graveyard.
    Headstones emerge from the ground in rows like a vegetable patch. Something here is familiar.
  </long_description>
  <exit id="veggie_garden_3" direction="north" target="walled_garden_3" locked="false">
    <short_description>the walled garden</short_description>
  </exit>
  <exit id="veggie_living_room" direction="west" target="macduff_living_room" locked="false">
    <short_description>a small cottage</short_description>
  </exit>
  <item id="headstone_1" type="furniture" name="a headstone" does_reset="true">
    <short_description>a rounded headstone</short_description>
    <long_description>
      You can make out the word "NAISMITH" on the grave. The dates seem to have been eroded away.
    </long_description>
  </item>
  <item id="headstone_2" type="furniture" name="a headstone" does_reset="true">
    <short_description>a crossed headstone</short_description>
    <long_description>
      It says, "Catherine Campbell." Apparently, Catherine died quite young.
    </long_description>
  </item>
  <item id="headstone_3" type="furniture" name="a headstone" does_reset="true">
    <short_description>a headstone topped with an angel</short_description>
    <long_description>
      This is a large onyx marker for the graves of the Macbeth family.
    </long_description>
  </item>
  <item id="buried_bone" magical="true" visible="false" name="a broken bone" does_reset="true">
    <short_description>a leg bone, fractured in the middle</short_description>
    <long_description>
      It's a human legbone. You can't imagine why you dug this up.
      When you gaze on it though, you can't take your eyes off of it.
    </long_description>
  </item>
  <action id="dig_in_graveyard" command="use" override="true">
    <event type="TextDisplayEvent">You know you're not supposed to dig in graveyards, but here you are.</event>
    <event type="VisibilityChangeEvent" thatThing="buried_bone" visible="true" silent="true"></event>
    <event type="ItemMoveEvent" thisThing="self" thatThing="buried_bone"></event>
    <event type="TextDisplayEvent">…Thiswill be enough. You should go.</event>
    <event type="RemoveActionEvent" id="dig_in_graveyard"></event>
  </action>
  <item id="dirt_mound" type="furniture" name="a mound of dirt" does_reset="true">
    <short_description>a mound of dirt, piled on top on a grave</short_description>
    <long_description>
      You might expect there to be grass or rocks in this dirt, but you see none.
      Everything appears to be the same consistency as cottage cheese, thanks to the rain.
    </long_description>
  </item>
  <item id="graveyard_dirt" magical="true" visible="false" name="dirt from a grave" does_reset="true">
    <short_description>dark grey mud, from a grave</short_description>
    <long_description>
      The mud oozes between your fingers, cold and slimy. There is power here.
    </long_description>
  </item>
  <action command="pickup" condition="equals(thatThing, &amp;quot;dirt_mound&amp;quot;)" override="true">
      <event type="VisibilityChangeEvent" thatThing="graveyard_dirt" visible="true"></event>
      <event type="ItemMoveEvent" thisThing="self" thatThing="graveyard_dirt"></event>
  </action>
  <action command="move" name="leave grounds for hotel" condition="equals(thatThing, &amp;quot;macduff_living_room&amp;quot;)">
      <event type="AssetEvent">
          <asset type="audio">all-off-zones</asset>
      </event>
      <event type="AssetEvent">
          <asset type="audio">hotel-zone-on</asset>
      </event>
      <event type="AssetEvent">
          <asset type="audio_loop">hotel</asset>
      </event>
  </action>
</room>
```

Figure 5.1: An excerpt of JEML used to define items and locations in the virtual world of Sleep No More.

It is worth discussing the audio system in more detail since it has some unique properties discussed briefly in previous sections. Rather than using a physical mixing console for routing inputs and output, the audio system was completely software based, running on top of the popular digial audio workstation, Reaper. Inputs and outputs of the DAW were created programmatically and initialized at the beginning of the performance. Connections were created to sample playback software (Kontakt 5), hardware inputs from traditional microphones, IP sources from the local network or the internet, or telephones linked via an Asterisk PBX. Using these capabilities, the system was able to dynamically route and mix many different sources to many outputs. The most nuanced mixes were created for users on their headphones at home. For these mixes, a combination of all the sources were spatialized binaurally and streamed in real-time via a Wowza and IceCast server to the web interface used by online users. The latency on the live mix delivered to clients was generally less than 4 seconds. For this reason most of the content was designed to cross fade in a subtle way. Certain sounds needed abruptly were triggered on the client end in the online participant's web-browser. Latency for these triggers could be much lower because no streaming buffer is required to ensure smooth playback of audio.

The control mechanism and the model for this type of production control is quite unique. Again, the approach is half video game, and half show control. To keep track of possibilities we defined a second markup language (BLEML, Ben-Luke-Elly Markup Language) mapping routing and assets to specific control commands. Unlike JEML, BLEML defined an idempotent set of 'cues' which could be applied over and over again. Each cue puts the system into a known independent state for some set of inputs and outputs, meaning that applying the same cue over and over again to the same IO has no result. This allows flexibility in the JEML system to make calls to BLEML without needing to keep track of the state audio system— the audio system will always become completely the state implied by the sent cue. By specifying inputs and outputs separately, the system is effectively running a cue stack for each set of

IO. Again, the ultimate result here is giving the online user agency. It is possible to enter and exit a room 50 times and the production systems will apply the correct sounds and video without needing to understand how many assets should be overlapping or whether fades have finished.

A sample of BLEML is shown in figure 5.2. Note that these cues define a set of inputs and are called from the JEML for a specific output. The JEML system triggers operations on all systems, including the Asterisk PBX, via OSC.

5.1.3 Operator in the Loop

What does it mean to operate a virtual show? In the case of Sleep no more, a team of 8 people operated various aspects of the production to make sure it was operating smoothly.

- A story logic operator was responsible for the overall interaction of all components of the production. This includes synchronization with live aspects of the show, all on time code.
- A location and tracking operator oversaw systems to understand where on-site participants were located throughout the production. This information was kept up to date within the JEML system.
- A portals operator was in charge of systems for high-bandwidth communication between remote and on-site participants. Portals were embedded in the set, allowing participants to communicate discreetly.
- Story operators were the remainder of the team, monitoring and manipulating interactions between those online and the JEML system.

Each member of the operations team was a human interface between technology and the audience. Rather than using the hyperinstrument-live-mixing model of letting systems do what they will and constraining the output, this arrangement actually allowed the operators to exactly determine both input

```
<show>
  <cue id="owls">
    <midi note="18"/>
  </cue>
  <cue id="vertigo-main">
    <midi note="19"/>
  </cue>
  <cue id="ballroom">
    <midi note="20"/>
  </cue>
  <cue id="thunder">
    <midi note="21"/>
  </cue>
  <cue id="all-off-zones">
    <midi startnote="0" endnote="11" hold="1" vel="0"/>
  </cue>
  <cue id="all-off-console">
    <midi startnote="0" endnote="63" hold="1" vel="0"/>
  </cue>
  <cue id="all-off-mask">
    <midi startnote="63" endnote="127" hold="1" vel="0"/>
  </cue>
  <cue id="all-off">
    <midi startnote="0" endnote="127" hold="1" vel="0"/>
  </cue>
  <cue id="kitchen-radio-stream-on">
    <channel id="cKit" vol="0" time="3"/>
  </cue>
  <cue id="kitchen-radio-stream-off">
    <channel id="cKit" vol="-60" time="3"/>
  </cue>
  <cue id="agnes-mirror-stream-on">
    <channel id="cAgnes" vol="0" time="4"/>
  </cue>
  <cue id="agnes-closet-stream-on">
    <channel id="cAC" vol="0" time="4"/>
  </cue>
  <cue id="grace-mask-on">
    <channel id="mGrace" vol="0" time="4"/>
  </cue>
  <cue id="grace-mask-off">
    <channel id="mGrace" vol="-60" time="4"/>
  </cue>
  <cue id="dark-closet-on">
    <channel id="ClstToPh" vol="0" time="2"/>
    <channel id="PhToHP" vol="0" time="2"/>
    <channel id="mGrace" vol="0" time="2"/>
  </cue>
  <cue id="dark-closet-off">
    <channel id="ClstToPh" vol="-60" time="2"/>
    <channel id="PhToHP" vol="-60" time="2"/>
    <channel id="mGrace" vol="-60" time="2"/>
  </cue>
  <cue id="phone-call-on">
    <channel id="GraceToPh" vol="0" time="0.5"/>
    <channel id="PhToHP" vol="0" time="0.5"/>
  </cue>
  <cue id="phone-call-off">
    <channel id="GraceToPh" vol="-60" time="0.5"/>
    <channel id="PhToHP" vol="-60" time="0.5"/>
  </cue>
  <cue id="phone-booth-triangle-on">
    <channel id="SmpToPh" vol="0" time="0.5"/>
    <channel id="PhToHP" vol="0" time="0.5"/>
    <midi note="2" hold="1"/>
  </cue>
  <cue id="phone-booth-triangle-off">
    <channel id="SmpToPh" vol="-60" time="0.5"/>
    <channel id="PhToHP" vol="-60" time="0.5"/>
    <midi note="2" hold="1" vel="0"/>
  </cue>
</show>
```

Figure 5.2: An excerpt of BLEML used to define several idempotent triggers in the Sleep No More Internet Extension audio system.

and output of many systems. In the case of the story operators, control would be transferred to a human if the computer was unsure of what to do. This allowed the system to behave with incredible nuance— a human with knowledge of story and the world can more naturally respond and react to unknown input.

This is the most basic form of hyperproduction. In this case of this production, it was the operators behind the scenes, not performers, that were able to shape the world and the presentation of the piece. Of course performers had a huge role in that presentation as well, but the operators also played a performative role, interacting with the technological systems.

We learned from this experience that a performative production role can be much more complex and nuanced than a simple sound mix or a visuals operator. Operators were responsible for aggregating all sorts of information from many sources and using it in a creative way to affect the experience of the participants. The production was a complex interplay of elements that could not have been managed any other way.

This production motivated some of the more advanced features of the hyperproduction system that will be described in the next chapter. The final implementation is able to handle the control aspects of these systems with additional benefits.

5.2 Death and the Powers Live

For the most recent 2014 performance of *Death and the Powers*, produced by the Dallas Opera, there was considerable talk about mounting performances in many different venues around the world. Instead we chose to do a MET-style international simulcast of a single production in Dallas. Traditional simulcasts bring up a unique challenge in that ultimately, it does not matter whether or not it is a live broadcast or a screening of an previously recorded piece. To address this, we developed an interactive mobile application designed to take

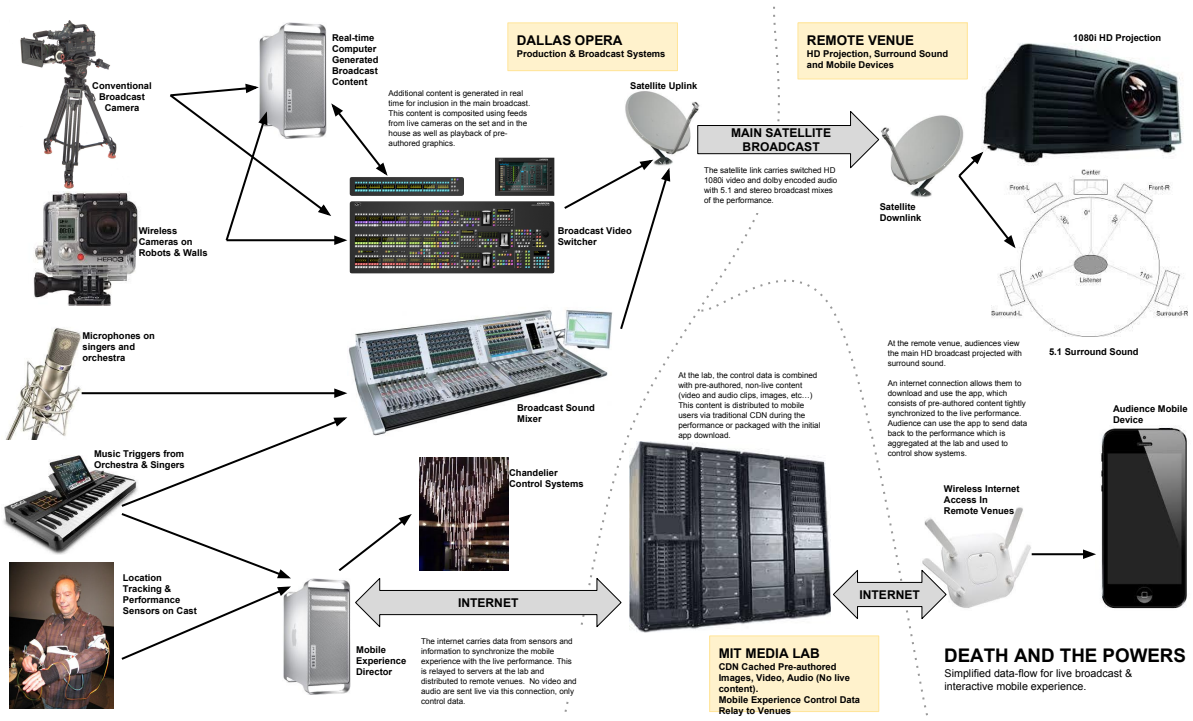

Figure 5.3: Death and the Powers Live System Flow

advantage of all the same show control infrastructure used to drive the performance systems. Visuals, audio, video and other elements were rendered in real-time on the devices, essentially extending the production systems into the hands of the remote audiences. Figure 5.3 gives an overview of the additional systems created for the simulcast.

All touch and movement events from the remote device were relayed back to the production in Dallas and incorporated into visuals control mappings for the Moody Foundation Chandelier, a 50ft LED-based lighting element in the house of the theater.

5.2.1 A Model of Emotion

The model for *Powers* was much different from Sleep No More simply because of the nature of the story. Instead of modeling the state and world of the entire

performance, the model consisted of an abstract representation of the main character, Simon Powers. In many ways this is effectually a model of the world and the entire performance because he becomes everything in the production and the story as the piece progresses.

Sensors attached to the body of the performer, Robert Orth, measured aspects of his movement, breath and voice. The sensor data was mapped to a set of abstract qualities using Peter Torpey's Disembodied Performance System.[44] These abstract qualities— anger, intensity, tension, fracture, and weight were used to control production systems. Meaningful mappings from these qualities to production elements were much easier to design (ex. the visuals should be more frenetic and red as Simon becomes more angry).

A visuals operator provided more nuanced shaping with continuous parameters connected to the mapping system as well. As with hyper-instruments, it would have been possible to do this mapping based on sensors, but given a fixed amount of time, the perception that a human operator offered was vastly more capable and nuanced. The operator was following the same sorts of parameters calculated from sensors, but was able to do it more precisely.

5.2.2 A Limited Representation

This type of organization of qualities with a single operator worked particularly well for *Powers*, an opera about an all encompassing omnipresence. Only Simon's emotions and performance were ultimately translated to the production systems. This is fortunate because the DPS system was only capable of measuring a single set of inputs and outputs. If there had been multiple characters which needed to be monitored, calibrated and sent to separate control systems, the mapping would have very quickly become unwieldy. A possibility might be to use a separate instance of the system for each character, but it quickly becomes difficult to track changes and updates between the two instances. In fact, this was necessary once the mobile aspects of the piece were added for the Dallas production. A secondary mapping system with a sepa-

rate set of cues and connections was responsible for managing interactions to and from mobile devices. While this was fine in practice, it does not diminish that fact that for an ideal system, the mapping instances would also be able to influence each other.

Ultimately, to be able to model a piece with all its entities and parts— not just based on a single character— we must have a way of organizing more complex mappings. There are a number of useful constructs which could help this sort of model to be much more widely applicable:

- Nesting of mappings
- Ability to group mappings and duplicate them together
- Separate cue stacks for separate groups of mappings
- Interactions and/or connections between disparate groups of mappings

Hyperproduction provides these constructs in a programmatic way so that systems can be constructed by hand or by script and components can be abstracted and isolated in meaningful ways. We will learn about the tools and organization of the hyperproduction framework in the next chapter.

5.2.3 Javascipt based Show Control

The control system for Powers Live was the first Opera of the Future Production system to be implemented in Javascript. NodeJS support for WebSockets is well tested and robust and the asynchronous nature of the V8 engine and Javascript made it possible to handle large numbers of concurrent connections quite effectively.

The system consisted of a load-balancer and two virtual machines each running 4 NodeJS processes with the Node Cluster module load-balancing TCP connections to the processes. VMs were chosen to increase reliability. If one of

the VMs went down, it was a simple matter to reconfigure the load-balancer and point connections to another instance.

Separate infrastructure served content delivery network accelerated assets to users, and a caching and content versioning system was created to ensure mobile devices would only download necessary components. Much care had to be taken to ensure that users would not overload the venue internet connections.

The system used a websocket connection opened to the device and with-in a webview renderer to send both triggers and live performance data at 12fps to users' devices. Playback of pre-cached media assets could be triggered, as well as real-time generative graphics, audio, and device vibration. Because triggers were sent via websocket, latency was negligible, compared with satellite broadcast latencies.

As a blue-print for the hyperproduction system, we paid close attention to performance metrics on the Powers live system and spent a significant amount of time load-testing and bench-marking before the broadcast. For a system and production of this magnitude (with a significant amount of publicity), it was imperative for the system to behave without issue on the day of the broadcast. This presented quite a conundrum, because there wasn't a way to truly test with real devices at scale— the first complete operation of the system was for the performance. Great care was taken to work with venues to ensure internet connectivity, bandwidth to the application servers.

We will discuss the performance and capability of this system and the hyperproduction system in Chapter 8.

Chapter 6

The Hyperproduction System

Given all of these production control methods used in a variety of ways over the last 7 years, we can imagine and design the next generation of a hyperinstrument system— one where the engineer is as integrated into the performance and the production control as the performers. Rather than using the engineer as a safety net to contrain the output of all systems, we should imagine the engineer as a node in a mapping system, capable of providing complex analysis and mapping that might require emotional intuition.

To facilitate this arrangement of people and technology, the Hyperproduction system is an advanced mapping and analysis tool for live data input, similar to the "Disembodied Performance System" by Peter Torpey and the "Extended Performance Extension System" by Elly Jessop. It is intended to be used in combination with an engineer, taking his or her input to production systems and using it to influence other production elements in a hyperintrument-like fashion.

The system is based on NodeJS which provides a performant, asynchronous platform that is optimized at runtime. The system has no sense of time or framerate; instead it is event-based and uses a push architecture. Any change of input causes an evaluation of all affected system outputs. Previously this sort of architecture would be too slow for use in performance critical environments. The wide adoptance of i series Intel processors and Google's V8 Javascript engine proved in initial testing that this arrangement could be achieved. In the case that an input changes too quickly, timing nodes are able to re-time incom-

ing data so the rest of the system evaluates at a reasonable interval.

6.1 Data Architecture

The system is separated into nodes that process data, devices that connect the system to the outside world, and containers that contain nodes and devices and have some interesting properties. Connections and ports facilitate the passing of data between nodes, device and containers. The system uses a "push" model to ensure that outputs are updated as their inputs change. Let's look in detail at each component.

All objects— ports, connnections, nodes, devices and containers- have a unique ID. This ID is used to associate objects with each other and create connections that pass data. When hand-writing mappings, it is possible to specify these IDs. Future versions of the system will have a descriptive name associated with nodes and containers that does not need to be unique.

All objects have a JSON representation as well, which provides a simple means of integration with frontend or monitoring systems. Internally, the JSON represenation is used as a template to create deep copies of nodes and containers.

6.1.1 Nodes

Nodes are the smallest building block of the system. A node takes a definition: a number of inputs and outputs, a text description and a processing function. When created, the node instantiates ports for each input and output, and connects them to the processing function from the definition. When inputs are updated, the processing function is called to process data to outputs. The processing function has a state that can be maintained across calls. This allows the function to implement processing that takes previous input or output into account.

Keeping the definition of the node's behavior separate from the node itself

```
1  //Definition of a node
2  AddOperation2i1o = {
3    nodetype: "AddOperation2i1o",
4    descr: "Two input adder",
5    procfn : function(ports, state) {
6      ports.o1.set(ports.i1.get()+ports.i2.get());
7      return state; // for a simple adder state is not used
8    },
9    inputs: {
10     i1 : { type : Type.INT, defaultValue : 0 },
11     i2 : { type : Type.INT, defaultValue : 0 }
12   },
13   outputs : {
14     o1: { type : Type.INT, defaultValue : 0 }
15   }
16 }
17
18 //Creating the node from the definition with the id "myAdder"
19 myMapping.createNode(AddOperation2i1o, "myAdder");
20
21 //Creating a second adder
22 myMapping.createNode(AddOperation2i1o, "anotherAdder");
```

Figure 6.1: A basic node definition and instantiation

```
1 //Connect the output of myAdder to the first input of anotherAdder
2 myMapping.connectByNodeIdPortName("myAdder","o1",
3                                    "anotherAdder","i1");
```

Figure 6.2: Connecting the two ports on nodes using the node-id, port name method.

facilitates simple copying of nodes. The node keeps the defintion used to create it and can therefore be duplicated very easily.6.1

Nodes are intended to be incredibly simple to write. During production, a node could be added to the system dynamically and replicated as many times as needed, wherever the node's functionality is required. The system does not need to be restarted to add nodes; this is a benefit of using a scripted language. The intention is that a library of nodes could be accumulated over time to provide many types of processing.

Data entering and exiting the node on a port can take any form. This provides support for complex data types such as multi-dimensional input and output, strings, ENUM, etc. In the current implementation it is up to nodes to do their own type checking. This functionality is left to the node and is not executed in ports or connections to prevent overhead. A map may have thousands of connections, ports and nodes. By checking type only in nodes that require it, it is possible to have greater control of the overall performance of the system.

6.1.2 Connections

A connection is a unary object that connects a single output and input. An output port may have many connections, but an input port may only have a single connection. When an output port's value is updated, the port object updates all its connections, which in turn update respective input ports.

Connections can be created between any two port IDs as long as one port is an input and one is an output. Methods exist to create connections between ports by their ID directly or also with two node IDs and port names.6.2

6.1.3 Devices

Devices are special nodes which connect the system to the outside world. A device node takes a definition that maps inputs or outputs to some control-data for a production system. Currently there are OSC Sender and OSC Receiver nodes. OSC is an increasingly common protocol that is found on many production systems— mixers, video switchers, lighting consoles, playback systems, DSP, etc. The definition for an OSC device node lists possible OSC addresses that the sender and receiver can use to communicate. The arguments for each address can be associated to ports on that node.

The OSC nodes are based on NodeJS OSC-min library and work in several modes:

Receiver nodes may

- work in a subscription mode, where a message is sent to suscribe to data sent to the system at periodic intervals. When data is received, the corresponding output ports on the device node are updated. Ports may be defined for each argument of every input message.
- work in a polling mode, where a message is sent for each OSC address at a specified timeout when that message is received. An update command sends messages to all defined OSC addresses for each address initially.
- work in a passive mode, where no attempt is made to subscribe or poll.

Sender nodes may

- work in an all-update mode, where any input causes all defined addresses to send their current state.
- work in single update mode, where input to the node causes only the associated OSC address to update.

- work in message send mode, where input on a port causes a specific message and arguments to be sent.

We have created basic OSC maps to prove interoperability with Behringer's x32 mixer, a 48 input 24 output digital mixing console that is OSC controllable. The x32 OSC Device node can read and send values of faders and read values of meters. There is also a basic OSC map for Blackmagic's ATEM HD-SDI video switcher, which is currently able to take a camera on the program output.6.3

Future implementation might involve the creation of DMX, ArtNet, MSC, MIDI and Time code Device nodes. The goal of the system is to have as much interface capability as possible with as many existing control systems as possible. This facilitates the connection of many different types of systems together using this platform.

6.1.4 Containers

Containers hold internal nodes and connections which can be duplicated or used together as a logical grouping of functionality. Since these containers are mappings themselves and can be represented as JSON, it becomes possible to store groups of nodes that can all be instantiated together in a textual representation.

Inlet and outlet nodes are special nodes whose connections appear on the outside of the container and on the Inlet or Outlet object inside the container. These make it possible for nodes outside a container to connect to nodes inside a container. This is achieved with a special port type, EXT, that when instantiated is added to the container's port list instead of the node's. An Inlet node has a standard output Port and an EXT input port, and Outlet node has a standard input port and an EXT output port.

Containers themselves behave like nodes and can therefore be embedded like a node inside other containers. A root container is instantiated automatically

```
1  //OSC Definition for first 3 channels of X32 mixer
2  X32ReceiveNode = {
3    nodetype: "Receive",
4    descr : "Receives␣data␣from␣the␣X32␣module",
5    outputs : {
6      o1: { type : Type.INT, defaultValue : 0},
7      o2: { type : Type.INT, defaultValue : 0},
8      o3: { type : Type.INT, defaultValue : 0}
9    },
10   addresses : {
11     "/ch/01/mix/fader": ["o1"], //map of arguments
12     "/ch/02/mix/fader": ["o2"], //to ports defined
13     "/ch/03/mix/fader": ["o3"]  //above
14   }
15 };
16
17 //Instantiating an OSC Device node with X32 definition and IP
18 var myReceiveNode = new OscReceiveNode("receiver",'18.85.52.46' \
19                                        , 10023, X32ReceiveNode);
20
21 //Add receive node created with X32 definition to mapping
22 myMapping.createDeviceNode(myReceiveNode);
23
24 //Create an adder in the mapping
25 myMapping.createNode(AddOperation2i1o,"adder");
26
27 //Create connect outputs of two faders to an adder
28 myMapping.connectByNodeIdPortName("receiver", "o1", "adder", "i1");
29 myMapping.connectByNodeIdPortName("receiver", "o2", "adder", "i2");
```

Figure 6.3: Instantiating and connecting OSC Nodes.

when the system is started to contain all devices, nodes and containers.

```
//Here we see the JSON result of embedding a container
//within a copy of itself

var ContainerMapNode = require("./ContainerMapNode.js");
var ComputeModules = require("./ComputeModules.js");
var l = require("./log.js");
var bb = require("./benb_utils.js");

var myMapping = new ContainerMapNode();

//Create all nodes, including inlets and outlets for the container
var add1 = myMapping.createNode(ComputeModules.AddOperation2i1o);
var inlet1 = myMapping.createNode(ComputeModules.Inlet);
var inlet2 = myMapping.createNode(ComputeModules.Inlet);
var outlet1 = myMapping.createNode(ComputeModules.Outlet);

//connect nodes together within our mapping
myMapping.connectByNodeIdPortName(inlet1, "o1", add1, "i1");
myMapping.connectByNodeIdPortName(inlet2, "o1", add1, "i2");
myMapping.connectByNodeIdPortName(add1, "o1", outlet1, "i1");

//create a copy of the container and its contents
//embedd the copy within our first mapping
myMapping.createContainerMapNode(myMapping.getFullMapping());

//Print resulting JSON
l.debug(JSON.stringify(newMapping.getFullMapping(), undefined, 2));
```

```

////
//
// Resulting JSON
//

{
  "nodes": {
    "48c41ed6-64f6-414d-809f-5f66be998f15": {
      "nodetype": "AddOperation2i1o",
      "descr": "Two input adder",
      "ports": {
        "inputs": {
          "i1": "145b2021-4c89-42f6-aa6d-3d0635da77e0",
          "i2": "7bd63c9d-ccfd-47c3-9040-429bf99a2322"
        },
        "outputs": {
          "o1": "b86ee1cf-a64e-4538-8b7b-29d3e5861f4d"
        }
      }
    },
```

```
    "8a8ea6c9-c8dd-4a78-9cf2-92e46602e386": {
      "nodetype": "Inlet",
      "descr": "Allows publishing inputs of a ContainerMapNode.",
      "ports": {
        "inputs": {
          "i1": "26546e23-11cd-4107-9d3d-24dd57007ad6"
        },
        "outputs": {
          "o1": "b7a00c01-6e52-4da5-a7cb-23542ea1ac47"
        }
      }
    },
    "16021595-e14e-4d14-8585-df98a5f3c804": {
      "nodetype": "Inlet",
      "descr": "Allows publishing inputs of a ContainerMapNode.",
      "ports": {
        "inputs": {
          "i1": "0540fa4c-5a65-49ef-b654-b7ed31992710"
        },
        "outputs": {
          "o1": "09a91baf-7df0-4d33-94c8-30d107fb99cd"
        }
      }
    },
    "4815305b-0aaa-4472-bb10-3eb0efc45cd2": {
      "nodetype": "Outlet",
      "descr": "Allows publishing outputs of a ContainerMapNode.",
      "ports": {
        "inputs": {
          "i1": "18fbdc58-3bfb-4e22-8f4f-2f02371fd873"
        },
        "outputs": {
          "o1": "4a0768cb-2cc7-4361-9863-671ad39a10d1"
        }
      }
    },
    "d9b0b6f7-a615-406d-85db-a30a2baebd71": {
      "nodes": {
        "620c5257-bede-485b-a0df-580f95155dc0": {
          "nodetype": "AddOperation2i1o",
          "descr": "Two input adder",
          "ports": {
            "inputs": {
              "i1": "59363ee1-834a-4c91-841a-4fd6a1330741",
              "i2": "5c9f5ab6-8f1d-410f-bf12-2e87374456de"
            },
            "outputs": {
              "o1": "6dea271a-505c-4011-988c-41c6dd1bd296"
            }
          }
        },
        "81292832-7a04-4054-829c-9692c685e39d": {
          "nodetype": "Inlet",
          "descr": "Allows publishing inputs of a ContainerMapNode.",
          "ports": {
            "inputs": {
              "i1": "f8d80287-abf6-4894-99fb-47f10b36d914"
            },
            "outputs": {
              "o1": "766409c1-b12a-4809-a128-940a1aa6a6a3"
            }
          }
        },
        "b8d48301-ca56-48b6-834b-3626508f7c70": {
          "nodetype": "Inlet",
          "descr": "Allows publishing inputs of a ContainerMapNode.",
          "ports": {
            "inputs": {
              "i1": "70e80717-4763-445e-90dd-662c23d09231"
            },
            "outputs": {
              "o1": "84c40f16-418a-4f18-947a-062d098258bf"
            }
          }
        },
        "b507585e-41eb-40b3-855a-31c4d37e6d33": {
          "nodetype": "Outlet",
          "descr": "Allows publishing outputs of a ContainerMapNode.",
          "ports": {
            "inputs": {
              "i1": "e60166f6-b081-4863-a5b7-c7cace8ca8fc"
```

```
            },
            "outputs": {
              "o1": "7ead8a71-d57e-499d-857a-a7917101f29d"
            }
          }
        }
      },
      "connections": {
        "62702f9c-6752-4a0a-ad0d-b6c4bfe30fe7": {
          "o": "766409c1-b12a-4809-a128-940a1aa6a6a3",
          "i": "59363ee1-834a-4c91-841a-4fd6a1330741"
        },
        "0eb3e627-3da4-4be5-a4f2-385f2599266e": {
          "o": "84c40f16-418a-4f18-947a-062d098258bf",
          "i": "5c9f5ab6-8f1d-410f-bf12-2e87374456de"
        },
        "b23a2e5d-c686-44d6-ac7e-d454b4c351ca": {
          "o": "6dea271a-505c-4011-988c-41c6dd1bd296",
          "i": "e60166f6-b081-4863-a5b7-c7cace8ca8fc"
        }
      },
      "nodetype": "ContainerMapNode",
      "descr": "A container for a sub mapping.",
      "ports": {
        "inputs": {
          "in-81292832-7a04-4054-829c-9692c685e39d": "f8d80287-abf6-4894-99fb-47f10b36d914",
          "in-b8d48301-ca56-48b6-834b-3626508f7c70": "70e80717-4763-445e-90dd-662c23d09231"
        },
        "outputs": {
          "out-b507585e-41eb-40b3-855a-31c4d37e6d33": "7ead8a71-d57e-499d-857a-a7917101f29d"
        }
      }
    }
  },
  "connections": {
    "7f7d1c20-bf07-405c-91ba-7195754058bb": {
      "o": "b7a00c01-6e52-4da5-a7cb-23542ea1ac47",
      "i": "145b2021-4c89-42f6-aa6d-3d0635da77e0"
    },
    "04dbc24c-e6cd-477a-bd67-1bd9173be56a": {
      "o": "09a91baf-7df0-4d33-94c8-30d107fb99cd",
      "i": "7bd63c9d-ccfd-47c3-9040-429bf99a2322"
    },
    "2f7a43ab-caad-4795-b0a0-30e4ea60ce29": {
      "o": "b86ee1cf-a64e-4538-8b7b-29d3e5861f4d",
      "i": "18fbdc58-3bfb-4e22-8f4f-2f02371fd873"
    }
  },
  "nodetype": "ContainerMapNode",
  "descr": "A container for a sub mapping.",
  "ports": {
    "inputs": {
      "in-8a8ea6c9-c8dd-4a78-9cf2-92e46602e386": "26546e23-11cd-4107-9d3d-24dd57007ad6",
      "in-16021595-e14e-4d14-8585-df98a5f3c804": "0540fa4c-5a65-49ef-b654-b7ed31992710"
    },
    "outputs": {
      "out-4815305b-0aaa-4472-bb10-3eb0efc45cd2": "4a0768cb-2cc7-4361-9863-671ad39a10d1"
    }
  }
}
```


It is important to note that while containers may have Inlets and Outlets to facilitate external connections, it is possible to connect any input and output across the entire system through many containers, as long as both port IDs or node IDs and port names are known.

6.2 Advanced Features

Several more advanced features are planed for this framework and have not been completely implemented and tested. These are described in this section.

6.2.1 Cue-Stack Containers

An enhanced container is planned with the following additions:

- The container has many sub-containers.
- Inlets and outlets are kept in sync between the master container and all sub containers so that all the sub-containers have the same inputs and outputs.
- A special interpolator node has all inlets and outlets of the master container.
- The connections to these inlets and outlets are switched to directly send and receive from a single sub-container based on input to the interpolator node.
- The interpolator node takes input to specify which sub-container should be used and a value to interpolate outputs to the newly selected sub-container.
- The interpolator node decides how to interpolate based on the data-type.

Given this implementation it would be possible to create a sequence of mappings and cross-fade between them. This effectively creates a structure where a list of cued interactions and be embedded inside a group-able, copy-able object. With the whole object represented in JSON, it is possible to progamatically design and instantiate cue lists. Cue-stack containers might contain specific input-output mappings or even just simple constants.

For the remainder of the document, when I use the terms cue, cue lists, and cue stacks in reference to this framework, I am referring to this arrangement of sub-containers.

6.2.2 Threading

By default, NodeJS operates on a single thread. This provides good performance and in practice works well for very large mappings. It would be possible to run multiple versions of the system and have them communicate. This communication could be using standard show control protocols (OSC, MSC, etc...) or a more tailored protocol. A device or node could be designed to manage inter-process communication.

6.2.3 Timed Nodes

In the current implementation, nodes pass data to their outputs on any change of input. This causes the system to run "as fast as possible." While this is desirable in many cases, one can easily imagine a scenario where systems should be pipelined and a desired frame-rate must be carefully maintained.

A timing node passes input to output at a specific rate rather than on any change. Both external and internal clocking mechanisms are planned. Because timing is not guaranteed in NodeJS, this sort of clocking is really only suitable for control data, not for media.

6.2.4 Feedback

When using a timed node, it will be possible to have nodes feed their output back to control inputs. Without imposed limits on timing, the system would get itself into an accelerating loop. Allowing feedback enables the possibility for cue lists to run themselves. An action or input can be mapped and determine the next cue or sub-container, on a cue-by-cue basis.

6.3 A Backend System

The system was designed such that it provides a flexible and scriptable backend that may be integrated with a frontend later on. It is not intended as a user-friendly tool for making these kinds of maps useful to beginners or even novices, rather it is for power users to facilitate experimentation with different models of performance and integration of operators and engineers into the analysis of live production.

The system provides a JSON representation. We additionally created a rudimentary UI to visualize containers, nodes and devices. It is suggested that frontends for this system keep a representation of the current container. All create, destroy, connect, and disconnect operations return appropriate JSON to keep a representation updated with the state of the backend system.

6.4 A Conceptual Framework for Creating Hyperproductions

Given the productions described earlier and the lessons learned building and producing pieces for the last several years, the following is a general guide to creating a system with this new framework:

Identify Inputs and Outputs

It is important to understand the types of sensor input, analysis, production output and control that will be possible for production. Often these resources are limited for reasons that are not artistic. Furthermore, it is important to understand how outputs are enumerated with respect to the audience. If there is a single speaker for each audience member, and each audience member has some specific control input (the case for Sleep No More), then a mapping and cuing system may need to be slightly different in arrangement than a single PA system for the entire audience with no input (the case for Death and the Powers)

Identify Model-worthy Elements within the Piece

We must determine which components of the piece require a model to be fully represented conceptually by the system. This part of the design process takes imagination and often it helps to understand the qualities of elements in the piece that may need to be altered. For example, in a television studio with a discussion panel, we may wish to model each panelist and keep track of qualities such as 'degree of engagement' or 'jealousy.' Using available input data, audio mix, volume, camera positions, etc... we might determine a value for each of these qualities for each panelist and use those values to determine a proper way to light the scene— for example with the most engaging panelist the brightest.

In Sleep No More, each location was modeled in detail. The entire world was augmented with properties to allow participants to interact with them in the physical and virtual worlds.

Death and the Powers Live uses a single character as one model and an aggregate of all remote audiences as a second model. In the implementation for Dallas, the two models were not able to interact.

In this framework, we use containers for modeled elements. Each container can have mappings which analyze incoming data, whether from production systems and operators (cameras, mixing console), performers (sensors on Simon Powers), or audience (user input from participants of Sleep No More). Since each container can have a cue-stack, it is possible to have many variations of states of analysis and output for any given element. Unlike existing systems, it is possible for different modeled elements to interact with each other.

Containers may even contain other containers, so it is possible to have nested sets of cue-lists. Adding feedback control of cues and input, this creates quite an unlimited set of control capability.

Mapping to Production Elements

Mappings for taking abstract qualities and rendering to production elements

may also be implemented in containers. These containers have inputs for abstract qualities coming from modeled elements and create the necessary transformations to apply the qualities to production elements. Often production control for basic elements does not have the flexibility to map directly from the model. Sometimes there is secondary mapping infrastructure whose only function is to scale continuous data, or quantize data to a specific frame rate which can be accepted by the control system for a particular element. A container can fulfill this role without needing an additional system. Timing, scaling, and even different modes of control (through cue stacks) may be integrated into a single container for this purpose.

Passing Non-traditional Data-types

Because the framework does not have restrictions on the data passed over ports and connections, it is possible to make nodes that process more complex inputs, such as text input from users online. This makes it possible to imagine and create interactions that are not only based on continuous input (floats, integers, etc..), but also text based, or other inputs. For implementing a system similar to the JEML structure used in the Sleep No More online experience, this capability is useful. Elements, cues and triggers may be named.

Chapter 7

Example Mappings and Applications

In this we'll detail several live production scenarios and possible representations of them within this framework. We will start with our two case study production, Powers and Sleep No More and look at a few other hypothetical scenarios as well.

7.1 Death and the Powers

An implementation of the Powers mapping systems in this framework is relatively straight forward. The following containers would be created:

- Simon container— with mappings for all of Simon's emotional state and outputs containing higher quality abstractions of his performance.
- Mobile systems container— with mappings for aggregated mobile interactions from remote audiences in other cities. This container outputs the state of the other souls in the system.
- Stage systems mappings— this container takes abstract qualities from the Simon container and scales them for use in the production systems on stage. This includes lighting, video and LED interactions.
- Chandelier container— which takes abstract data from Simon, and the

remote users to generate control data for systems rendering the visuals displayed on the Chandelier.

- Operator input container— receiving data from operators, sound and visuals and routing as input data to the Simon and Remote Audience containers.

Each container has nodes which implement the mappings required to take input to output in a meaningful way.

7.2 Sleep No More

Sleep No More has complex requirements that can only broadly be considered "mapping." To build a system with the hyperproduction framework that recreates the functionality of the story logic and JEML systems is not trivial.

To do this, we should imagine cue-stack containers to operate like state-machines. Each cue (or sub-container) representing a location in the world has an internal mapping which contains constants for text delivered to the user, items in the room, available actions, media assets, etc... Actions in the room may utilize their own cue-stack container if there is logic to their availability. Output from the internal nodes in a cue feeds the cue selection input for the main stack of locations. On a cue change or cross-fade, the entire contents of the sub-container change to new constants, sending new data back to the user. Input and output can be numbers, text, vectors or just about any other data type. This makes it possible to imagine a node handling user input in text form.

Creating that number of cues would be incredibly complicated to do by hand writing definitions. Instead, it would be possible to write a JEML parser which could read the file and create all the necessary infrastructure to support the production.

The advantage of designing the system this way: it is possible to generatively build a world for each user when they log in to the system. Not only is it

possible to build these worlds, duplicating state, functionality and everything else, but it is also possible to create links between the created worlds quite easily. Again, a connection can be created between any input and output ports on the system, regardless of how many sub-containers there are separating the ports from each-other.

We look forward to the next production requiring this type of complexity. Although this framework has not yet been tested in this type of use case, it was designed with these capabilities in mind to be able to handle the most advanced type of multifaceted production. It should be possible to run a cue-stack container or sub containers for each member of the audience, dynamically build and destroy mappings based on templates, and create links across the system to or from any component to any other component, regardless of how the components were created.

Chapter 8

Performance and Evaluation

It is not simple to come up with an evaluation for a system which has not yet been battle tested in a large-scale production. Had timing and scheduling worked out, this system would have been employed for Death and the Powers and several productions slated for summer of 2014. Unfortunately, the summer productions did not happen and Powers had production challenges which meant time for testing new and un-proven systems was very short.

That said, the Powers Live system is a good example of a completely fresh approach on a new platform (NodeJS) with some of the architecture and technical implementation that was later incorporated into the Hyperproduction framework.

The Powers Live system also gave a good sense of how to work with NodeJS at scale and in high-pressure production situations. The content versioning and remote control aspects of the system, allowing us to centrally disable and enable functionality, roll-back or push new assets and content, allow or disallow network connectivity, etc... were heavily utilized when, in the 15 minutes before the production, telephone calls started arriving saying that a few users were experiencing crashes of the mobile application.

Ultimately the system worked incredibly well. Our simulations on the system supported about 7500 clients connected across 8 processes. The bandwidth on that system 200Mbit with high CPU usage on the VM guests and host. During the performance, we sustained about 1000 simultaneous connections across 9

cities for the 90 minute production.

The performance of NodeJS on Powers Live made it a good candidate for the Hyperproduction framework. As well as being extraordinarily easy to integrate with websockets, communications protocols like OSC, and other show related IO (MIDI, etc...), the asynchronous nature of the platform lent itself well to the challenge of eschewing traditional frame-rate limitations. Existing systems in the Opera of the Future group (DPS[44] and all derivatives) use a set frame-rate for the entire system. Not only do they require a single specific frame rate, but they use a "pull" architecture, where outputs are evaluated each frame by traversing a tree all the way to the inputs. While this integrates well with video and visuals based systems, it is unsurprisingly quite complex to use connected to many systems at running different rates.

Since the goal of this framework is to do exactly that, it became necessary to choose a new architecture. The holy grail of these systems is a "push" evaluation method where outputs are re-evaluated when input changes. This sort of system is often complicated to make work well because evaluations may need to happen incredibly fast. Initially, several tests were conducted to determine whether this sort of approach was feasible with NodeJS and what overhead was incurred by having the structure of Nodes, Containers, and connections.

The following performance test was conducted:

- Programmatically create a large number of two input adder nodes
- Connect nodes together in a large tree, with the inputs of each adder connected to the outputs of another adder.
- On the bottom of the pyramid (adders with unconnected inputs), loop through the inputs of each adder and change the value as quickly as possible.
- Count and time the number of changes of the output of the last adder.

This test is shown in figure 8.1.

```
1
2  var ContainerMapNode = require("./ContainerMapNode.js");
3  var ComputeModules = require("./ComputeModules.js");
4
5  var l = require("./log.js");
6  var bb = require("./benb_utils.js");
7
8  var myMapping = new ContainerMapNode();
9
10 var triangle_height = 3, total_nodes = Math.pow(2,triangle_height);
11 var unconnectedInputs = []
12
13 for (var i = 0; i < total_nodes-1; i++) {
14   var n = myMapping.createNode(ComputeModules.AddOperation2i1o,i);
15   bb.forEachObjKey(myMapping.getPortsByMapNodeId(n).inputs, function(portName,portId) {
16     unconnectedInputs.push(portId);
17   });
18   if ((unconnectedInputs.length > 0) && (i > 0)) {
19     myMapping.connectByPortId(myMapping.getPortsByMapNodeId(n).outputs.o1, unconnectedInputs.shift());
20   }
21 }
22
23 myMapping.createNode(ComputeModules.ProcessCounter,"counter");
24 myMapping.connectByNodeIdPortName(0,"o1","counter","i1");
25
26 console.log(JSON.stringify(myMapping.getFullMapping(), undefined, 2));
27
28 //console.log(JSON.stringify(myMapping.getCytoscapeElements()));
29
30 l.profile("1000 adds");
31 for (var c = 0; c < 1000; c++) {
32
33   //l.profile("single add")
34   for (var i = total_nodes-1; i >= total_nodes/2; i--) {
35     myMapping.setValueByNodeIdPortName(i-1,"i1",c);
36     myMapping.setValueByNodeIdPortName(i-1,"i2",c);
37   }
38   var total = myMapping.getValueByNodeIdPortName(0,"o1");
39
40
41   //console.log(total);
42   if ( total != total_nodes*c ) {
43     console.log("Ack something went wrong! "+total)
44     break;
45   }
46
47   //l.profile("single add")
48
49   //l.info(myMapping.getValueByNodeIdPortName("counter","o1"));
50
51
52 }
53 l.profile("1000 adds");
```

Figure 8.1: Hyperproduction framework adder performance test

Setting the tree height gives the number of total adders. For 1000 loops, each unconnected input is set to a new value. This causes evaluations of each branch in the tree for every leaf. Table 8.1 shows the results of this test and it is possible to get an idea of how many nodes are realistic for a given process. These results are in the best case, the V8 Javascript runtime is able to optimize the adder node efficiently. For more complex nodes, performance would most likely be worse.

However from these results, we can see that it is possible to have an output fed by 128 inputs changing as quickly as possible while maintaining well over 30 frames-per-second at the output node. A single branch evaluation is extraordinarily fast. Typically trees of nodes in mappings are much deeper than they are wide, so this is also encouraging— we can see that evaluating 10 nodes in a branch is well under a millisecond.

This is sufficiently performant to run all mappings that have been created by the Opera of the Future Group in the last 6 years. Most mappings are on the order of 25-40 total nodes, which would run at 300-600 frames per second in this framework. Given these results and the new organizational features added, we look forward to creating much, much larger and more complex mappings. For extremely large systems that cannot fit on a single process, it would be possible to run different components in separate processes as well.

Additional work will be conducted in the following months:

- Running a similar test with two isolated trees with separate outputs to understand whether performance of the evaluations are correlated to the number of total nodes in the mapping or only the nodes in the tree
- Quantifying the run time of very long branches
- Testing the performance more complicated data types— such as arrays and strings
- Testing inter-process communication to see how isolated the computa-

Height	No. Adders	Inputs	Run Time (ms)	All Eval. (ms)	Branch Eval. (ms)	FPS All	FPS Branch
1	2	1	34	0.034	0.034	29411.7	29411.7
2	4	2	106	0.106	0.053	9433.9	18867.9
3	8	4	282	0.282	0.070	3546.0	14184.3
4	16	8	702	0.702	0.088	1424.5	11396.0
5	32	16	1680	1.68	0.105	595.2	9523.8
6	64	32	3953	3.953	0.123	252.9	8095.1
7	128	64	9176	9.176	0.143	108.9	6974.7
8	256	128	21031	21.031	0.164	47.5	6086.2
9	512	256	47649	47.649	0.186	20.9	5372.6
10	1024	512	110083	110.083	0.215	9.08	4651.0

Table 8.1: Results of Adder Test conducted on a 2.4GHz Intel quad-core i7 laptop

tion running across processes must be

Even given all of these tests, the best feedback will be obtained by using the system for a production in a typical scenario where mapping and show control are required. Plans are in progress to see this happen throughout the next academic year.

Chapter 9

Conclusions and Future Work

Production systems are being innovated at incredible rates. The speed at which new sensing methods, new production elements, and new signal processing are introduced is staggering. As these new systems are growing popular, more and more complex control technology is necessary to manage and connect them together. Timecode and traditonal cuing cannot give us meaningful, emotional, human experiences. It is my hope to develop the advanced features of the hyperproduction system and turn the platform into a tool that can be adapted and extended as new technology and systems are released. By creating a system capable of replacing every existing control arrangement we have used in thus far in Opera of the Future, I hope that we can push the boundaries of what is possible with live experiences.

The next step in this work will be to test this system on large-scale high stakes production. There are many upcoming opportunities to see this take shape and understand what modifications need to be made to support flexible rehearsing, programming and performing with the system. Our hope to support an entirely new type of performance, one that specifies new and different involvement of space, time, interaction and emotion. Scoring in these realms opens the door to many interesting possibilities for experience. Among upcoming opportunities are a pavilion for the World Expo 2020 in Dubai, two operas in Boston and the United Arab Emirates, new hyperensemble pieces, and many other exciting commissions.

It is my intention to continue development on this project as long as it remains

a technically groundbreaking complement to our production process. Since it provides extraordinary new capabilities that have been impossible until now, I look forward to seeing it grow and become an integral part of our future work.

References

[1] *A Toronto Symphony*. URL: http://toronto.media.mit.edu (visited on 07/21/2014).

[2] *Ableton Live*. URL: https://www.ableton.com/en/live/new-in-9/ (visited on 07/21/2014).

[3] *Avid*. URL: http://www.avid.com (visited on 07/21/2014).

[4] *BlackTrax*. URL: http://www.cast-soft.com/blacktrax (visited on 07/21/2014).

[5] Benjamin Bloomberg. *Death and the Powers Systems Detail Workbook*. URL: http://web.media.mit.edu/~benb/statics/POWERS%20Majestic%20Documentation%20v2.1.pdf (visited on 07/21/2014).

[6] Antonio Camurri et al. "EyesWeb: Toward Gesture and Affect Recognition in Interactive Dance and Music Systems". In: *Comput. Music J.* 24.1 (Apr. 2000), pp. 57–69. ISSN: 0148-9267. DOI: 10.1162/014892600559182. URL: http://dx.doi.org/10.1162/014892600559182.

[7] *Chamsys MagicQ*. URL: https://secure.chamsys.co.uk/magicq (visited on 07/21/2014).

[8] *Chuck Language Documentation*. URL: http://chuck.cs.princeton.edu/doc/language/ (visited on 07/21/2014).

[9] J. Rogers D. Dixon and P. Eggleston. *Between Worlds: Report for NESTA on MIT/Punchdrunk Theatre Sleep No More Digital R&D Project*. University of Dundee, University of West England Bristol, 2012.

[10] *D'Mitri Digital Audio Platform*. URL: http://www.meyersound.com/products/d-mitri/ (visited on 07/21/2014).

[11] *Digico*. URL: http://www.digico.biz (visited on 07/21/2014).

[12] Marc Downie. "Field— a New Environment for Making Digital Art". In: *Comput. Entertain.* 6.4 (Dec. 2008), 54:1–54:34. ISSN: 1544-3574. DOI: 10.1145/1461999.1462006. URL: http://doi.acm.org/10.1145/1461999.1462006.

[13] *Dugan Automixing*. URL: http://www.protechaudio.com/products/PDFFiles/DuganMixing.pdf (visited on 07/21/2014).

[14] *Einstein on the Beach: LA Opera Program Notes*. URL: http://www.laopera.org/documentslao/press/1314/lao.einstein-iw-f-web.pdf (visited on 07/21/2014).

[15] *Electronic Theater Controls*. URL: http://www.etcconnect.com (visited on 07/21/2014).

[16] *Enterprise Location Intelligence*. URL: http://www.ubisense.net/en/ (visited on 07/21/2014).

[17] *Faster than Sound 2010: Spheres and Splinters*. URL: http://bot23.com/2010/11/15/faster-than-sound-2010/ (visited on 07/21/2014).

[18] Rebecca Fiebrink, Perry R. Cook, and Dan Trueman. "Human Model Evaluation in Interactive Supervised Learning". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pp. 147–156. ISBN: 978-1-4503-0228-9. DOI: 10.1145/1978942.1978965. URL: http://doi.acm.org/10.1145/1978942.1978965.

[19] *Figure 53: QLab*. URL: http://figure53.com/qlab/ (visited on 07/21/2014).

[20] Mary Hart. *The art of ancient Greek theater*. Los Angeles, Calif: J. Paul Getty Museum, 2010. ISBN: 1606060376.

[21] *High End Systems*. URL: http://www.highend.com/ (visited on 07/21/2014).

[22] Andy Hunt, Marcelo M. Wanderley, and Matthew Paradis. "The Importance of Parameter Mapping in Electronic Instrument Design". In: *Proceedings of the 2002 Conference on New Interfaces for Musical Expression*. NIME '02. Dublin, Ireland: National University of Singapore, 2002, pp. 1–6. ISBN: 1-87465365-8. URL: http://dl.acm.org/citation.cfm?id=1085171.1085207.

[23] John Huntington. *Show networks and control systems : formerly control systems for live entertainment*. Brooklyn, N.Y: Zircon Designs Press, 2012. ISBN: 0615655904.

[24] *Intelligent Audio Mixing and Mastering Technology - Mix Genius*. URL: http://mixgenius.com/ (visited on 07/21/2014).

[25] *JR Clancy Scene Control 5600*. URL: http://www.jrclancy.com/scenecontrol5600.asp (visited on 07/21/2014).

[26] Elena Naomi Jessop. *A gestural media framework : tools for expressive gesture recognition and mapping in rehearsal and performance*. 2010.

[27] Elena Jessop, Peter A. Torpey, and Benjamin Bloomberg. "Music and technology in death and the powers". In: *Proc. NIME '11*. 2011, pp. 349–354.

[28] *LCS CueConsole Modular Control Surface User Guide*. URL: https://www.meyersound.com/pdf/products/lcs_series/CueConsole_Brochure.pdf (visited on 07/21/2014).

[29] R Laban. *Mastery of Movement*. 4th ed. Northcote House, 1980.

[30] *MIDAS*. URL: http://www.midasconsoles.com (visited on 07/21/2014).

[31] *MIT Media Laboratory, Opera of the Future: Death and the Powers*. URL: http://powers.media.mit.edu (visited on 07/21/2014).

[32] Tod Machover. "Hyperinstruments: A Composer's Approach to the Evolution of Intelligent Musical Instruments". In: *Cyberarts: Exploring the Arts and Technology* (1992). Ed. by L. Jacobson, pp. 67–76.

[33] Tod Machover. *Hyperinstruments: A Progress Report*. MIT Media Laboratory, 1992.

[34] *Madam Butterfly Stalked By TiMax Track The Actors*. URL: http://livedesignonline.com/theatre/madam-butterfly-stalked-timax-track-actors (visited on 07/21/2014).

[35] Joseph A. Paradiso and Neil Gershenfeld. "Musical Applications of Electric Field Sensing". In: *Computer Music Journal* 21 (1997), pp. 69–89.

[36] *Philip Glass Ensemble: Artists*. URL: http://www.pomegranatearts.com/project-philip_glass/artists.html (visited on 07/21/2014).

[37] Robert Pinsky. "Death and the Powers: A Robot Pageant". In: *Poetry* (2010), pp. 285–327.

[38] *SFX*. URL: https://www.stageresearch.com/products/SFX6/SFX6.aspx (visited on 07/21/2014).

[39] Evert Start. "Direct sound enhancement by Wave Field Synthesis". PhD thesis. Delft University, 1997.

[40] Joshua Strickon. "Smoke and Mirrors to Modern Computers: Rethinking the Design and Implementation of Interactive, Location-Based Entertainment Experiences". PhD thesis. Massachusetts Institute of Technology, 2003.

[41] *Studer by Harman*. URL: http://www.studer.ch (visited on 07/21/2014).

[42] *SuperCollider*. URL: http://supercollider.sourceforge.net/ (visited on 07/21/2014).

[43] *Timax Tracker*. URL: http://www.outboard.co.uk/pages/timaxtracker.htm (visited on 07/21/2014).

[44] Peter Alexander Torpey. *Disembodied Performance: Abstraction of Representation in Live Theater*. 2009.

[45] Akito van Troyer. "Enhancing Site-specific Theatre Experience with Remote Partners in Sleep No More". In: *Proceedings of the 2013 ACM International Workshop on Immersive Media Experiences*. ImmersiveMe '13. Barcelona, Spain: ACM, 2013, pp. 17–20. ISBN: 978-1-4503-2402-1. DOI: 10.1145/2512142.2512150. URL: http://doi.acm.org/10.1145/2512142.2512150.

[46] *Yamaha Commercial Audio Systems*. URL: http://www.yamahaca.com/ (visited on 07/21/2014).