California Institute of the Arts

Collaborative Applications in Computational Creativity and the Arts

by

Dimitri Diakopoulos

A thesis submitted in partial fulfillment for the
degree of Master of Fine Arts

in the
Herb Alpert School of Music
Music Technology: Interaction, Intelligence & Design

November 2012

Supervisory Committee

Dr. Ajay Kapur, Mentor

Owen Vallis, Reviewer

Jordan Hochenbaum, Reviewer

"An idea or product that deserves the label 'creative' arises from the synergy of many sources and not only from the mind of a single person."

Mihaly Csikszentmihalyi

California Institute of the Arts

Abstract

Herb Alpert School of Music
Music Technology: Interaction, Intelligence & Design

Master of Fine Arts

by Dimitri Diakopoulos

This thesis presents several practical applications of networking techniques to support real-time collaborative creativity. Firmly situated in musical and physical computing, the following work synthesizes foundational work from a number of domains to produce knowledge that can be applied by other developers exploring applications in computational creativity. One of the main themes of this work is wrangling complexity for both developers and users of collaborative systems, as seen in four research-oriented projects. The first chapter presents HIDUINO, a foundational utility to proxy the physical input and output of networked systems. The second chapter details Signal, a client-server application built on the premise of engendering creativity in the collaborative composition of music for robotic instruments. The third chapter steps back from the application-level and looks at the development process of synchronous systems by introducing the Netpixl toolkit. Chapter four delves into the design principles of GR1D, a project conceptually marrying the design philosophies of Netpixl with an audience-interactive installation.

Contents

List of Figures

Abbreviations

API	Application Programming Interface
CSCW	Computer Supported Cooperative Work
DIY	Do It Yourself
DME	Digital Music Ensemble
HCI	Human Computer Interaction
HID	Human Interface Device
JS	JavaScript
JSON	JavaScript Object Notation
MTIID	Music Technology: Interaction, Intelligence & Design
MVC	Model, View, Controller
OS	Operating System
OSC	OpenSoundControl
RGB	Red Green Blue
TMO	The Machine Orchestra
TTL	Transistor-Transistor Logic
UI	User Interface
WIMP	Windows, Icons, Menus, Pointing

Chapter 1

Introduction

This thesis documents several case studies showcasing the design of projects within the sphere of *computational creativity*. In particular, this work emphasizes the use of local-area networks to afford synchronous artistic collaboration. As the tools, programming languages, and theory of creative coding has evolved in the past decade, this work extols the virtue that new media artists are uniquely positioned to design new systems with greater social and artistic significance.

Computation and the Arts

I believe a compelling area of research is in the methods by which systems can be designed to mediate real-time creative collaboration. A broad definition for this might include the the term computational creativity to denote the research, design, development, and evaluation of new tools that can re-synthesize and organize the input of multiple creative minds to produce meaningful work. Unified by a goal to demonstrate new ideas in creative and collaborative synchronicity, this thesis can be divided up into two guiding research questions: i) How is it possible to artistically leverage the real-time expressivity of networks for collaborators? and ii) How can we technically support new work in this area?

This thesis looks at these questions through several tools and applications. These projects are framed in a wider context of physical computing and creative coding, recent terms describing meta-techniques of 21st century media artists. More deeply,

synchronous creative collaboration is a similarly a youthful field of research, having its roots in creativity and cognition theory, social systems theory, and a wide array of computationally-related disciplines including ubiquitous computing, human-computer interaction, and computer supported cooperative work. Identifying this, networked collaboration in the arts is a relatively small niche benefitting from applications of theoretical and practical principles in these domains. In the following work, I try to synthesize major concepts across these areas to explore the connection between networks, collaboration, music, installation art, and creativity. By producing work within this space, I feel innovation not only advances our understanding and appreciation of a single field such as musical computing, but a wide spectrum of digital arts as well.

Domain Definitions

Already, there are a great number of terms that require clarification in understanding the focus and direction of this work. The first concerns creativity. Csikszentmihalyi defines creativity as *"any act, idea, or product that changes and existing domain, or that transforms an existing domain into a new one [1]."* Within this definition, Csikszentmihalyi also defines the notion of creativity with a Big C and creativity with a little c. Big C is concerned with groundbreaking ideas that fundamentally change a domain. Little c is associated with the creativity of everyday, and the type most musicians and artists experience regularly. Beyond this, computational creativity denotes the act of technological assistance to the human creative process, a focus of the projects presented here.

Human computer interaction (HCI) is a field describing user interfaces and interaction techniques. HCI is important within computational creativity because designing tools to support creativity often takes Big C innovation. User interfaces work at a high level of abstraction in a system. Tools cannot always cope with the immediacy of human creative desire to express [2], and users (or developers) are often constrained by the substantial work needed to translate idea to output. Research in Computer Supported Cooperative Work helps mitigate such constraints, in general, by iterating on our understanding of creative collaboration. Synchronicity, as used in this thesis, is another term for *perceptual real-time* (a departure of the definition of real-time as viewed by engineers, for example). The last term warranting a definition is *abstraction*, a common concept in software

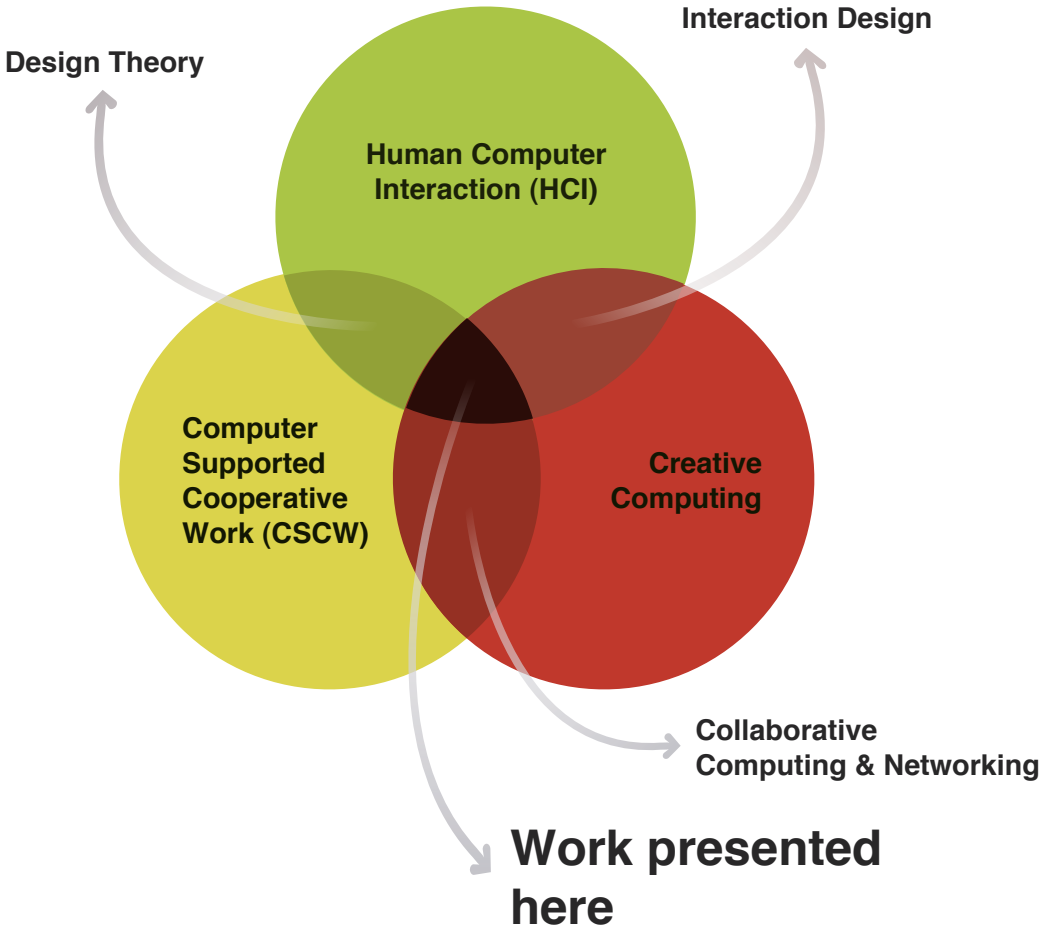

FIGURE 1.1: Three-part Venn Diagram illustrating where this thesis exists within several research fields. To be specific, this thesis has a dual focus on musical computing systems and abstractions that I believe can be useful in other domains such as computational art or interactive installations.

engineering. Abstracted ideas and patterns are those which hide implementation details in order to reduce the mental complexity of understanding and implementing an idea. A quick reference to many of these terms and others found within Appendix A.

Motivations

Firstly, this work is founded on the premise of enhancing the expressivity and creativity of artists who create – or who are interested in creating – through hardware and software. Within this model, I am motivated to showcase how social components for visual and musical expression can be ideated. Working towards this, one goal is to demonstrate the rich availability of easily-graspable abstractions for networking and sensing in the

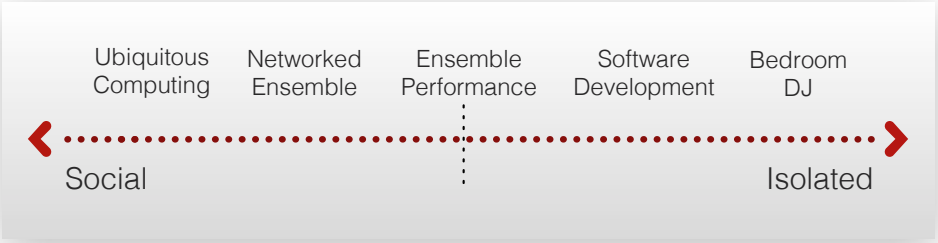

FIGURE 1.2: Subjective placement of creative activities across a social spectrum. The projects introduced in this thesis are designed to leverage real-time networking to shift this spectrum toward social.

creation of new applications, pieces, and installations. Another goal is to inspire beyond the keyboard and mouse interaction paradigm. In this work, I attempt to build on the recent idea that embodied interaction via sensors and networks is a natural extension of artistic expression. I believe such embodiment is key in generating a sense of physicality among geospatially diverse collaborators, a factor of success in new interactive systems.

Secondly, another core motivation of this work is to expound methods for managing complexity. Underlying all four projects presented here, the theme that software developers should remain unburdened in their creative endeavors is a common thread. Managing complexity relates directly to the process by which the architects of frameworks, toolkits, libraries, applications, systems, and environments can focus on the *design* task at hand, rather than an encumbrance of features and low-level code concerns. My goal in this is in demonstrating how code-level tools may help developers fluidly navigate the creative process while remaining sufficiently flexible. Unspecific but related to this goal, this thesis broadly encourages the use of an agile software development process: design, build, and iterate.

Thirdly, this work is influenced by the principles of the Critical Engineering Manifesto. Principal Zero states:

> The Critical Engineer considers Engineering to be the most transformative language of our time, shaping the way we move, communicate and think. It is the work of the Critical Engineer to study and exploit this language, exposing its influence[3].

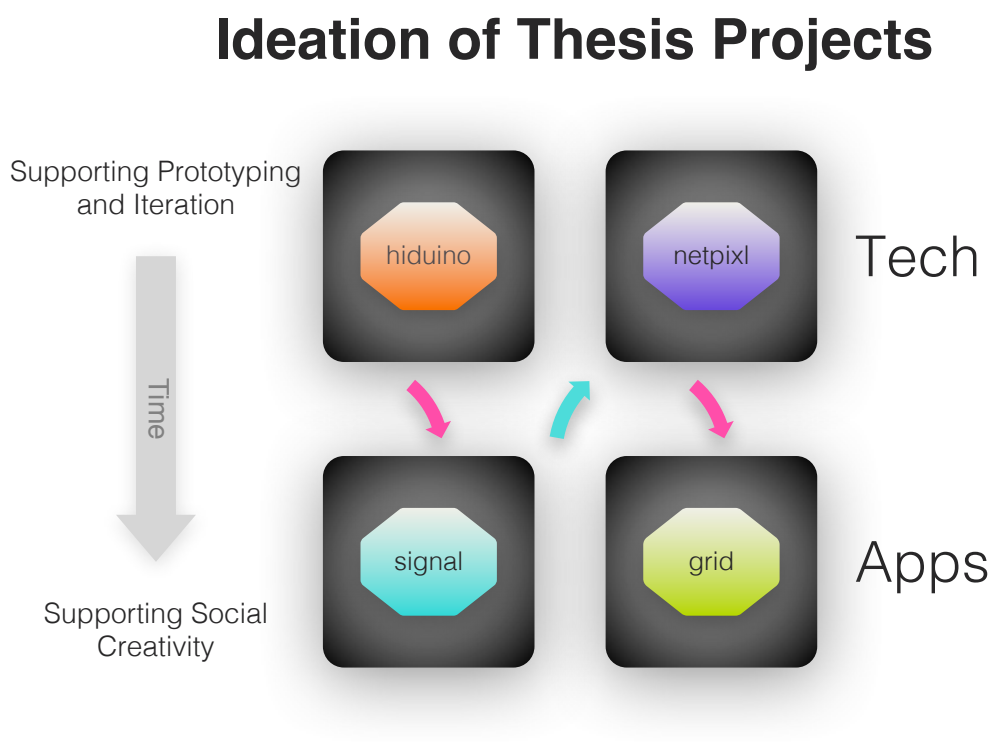

FIGURE 1.3: A visual explorer of the projects presented in this thesis and their technological and ideological relationships

It is without a doubt the future of creative work is moving in the digital direction. I believe it is our responsibility as (software) engineers to embrace this and inspire as many artists as possible.

Outline

As shown in Figure 1.3, this thesis presents four projects that may be grouped as two tools and two applications. These projects are discussed in order of their conception, demonstrating their relative influence on one another while illustrating the evolution of ideas and technology. Chapter 2 presents HIDUINO, a toolkit that helps students build physical computing systems. Chapter 3 introduces Signal, software that defines a domain-specific application of a model for embodied interaction within the context of musical robotics. Chapter 4 introduces Netpixl, a generic toolkit for building browser-based collaboration network influenced by recent advances in networking technology and incorporating research gleaned from Signal. Lastly, Chapter 5 expounds the design

strategies of an interactive media facade entitled *GR1D*. GR1D is an architecturally integrated display exploring diffuse audience-installation mobile interaction. Netpixl was developed as the software layer for GR1D and is a continual area of research beyond the work presented here.

Chapter 2

HIDUINO: Managing Complexity in Physical Computing

This chapter presents HIDUINO, a series of open-source firmwares for the latest iterations of the popular Arduino microcontroller platform. A portmanteau of Human Interface Device and Arduino, the HIDUINO project tackles a major problem in designing NIMEs: easily and reliably communicating with a host computer using standard MIDI over USB. In many systems, this process is typically fraught with a number of communication difficulties and protocol translations that occur in order to transform sensor data into standardized and/or sharable formats. A 2010 update to several Arduino models included a hackable and re-programmable USB controller chip which permitted a great simplification of this process. The HIDUINO firmwares were developed to enable USB MIDI communication, the protocol standard to most commerical MIDI controllers but uncommon in the DIY community. The primary use cases for HIDUINO were were developed in conjunction with a class at the California Institute of the Arts that introduces physical and virtual interaction design within the context of musical controllers.

2.1 Overview

In isolation, HIDUINO is not a collaborative system, but rather a building block for higher level processes supporting computational projects in the arts. Managing complexity, as a recurring theme in this thesis, is the underlying thread which connects this

work to the other systems detailed in the rest of this thesis. As most collaborative systems have fundamental input and output components, HIDUINO is a software process for message transport within larger and more complex musical and artistic projects. Here, managing complexity manifested itself as a process simplification derived from experiences teaching introductory physical computing. From this, the project aims to drive user's creative concepts further without overburdening the already lengthy process of developing new artistic systems and projects. HIDUINO is also briefly discussed in Chapter 3, where the software component plays a mixed role in the data sharing and transport scheme of The Machine Orchestra.

A Little Background

With new systems techniques regularly presented, it is without doubt that many within the New Interfaces for Musical Expression conference community are experts at finding creative and ingenious uses of new sensing technology. This shared knowledge coupled with the rapidly growing open-source and open-hardware maker community enables both experienced and inexperienced artists to build expressive interfaces for musical control [4–6]. Inspired by this recent surge in open-source and open-hardware, a core goal of the Music Technology program at the California Institute of the Arts is to teach students how to connect the physical and virtual world. Interface Design for Music and Media Applications is a yearlong class that introduces students to artistic interactivity through microcontrollers and sensors/actuators[1]. Modeled after the template presented at CCRMA by Bill Verplank et al in 2001, "A Course on Controllers [7], and Gideon DArcangelos course at ITP [8] the class required a modern microcontroller platform that was neither too high-level nor too complex.

The Arduino turned out to be the ideal solution for the class, although there were significant usability issues related to the platforms main built-in communication protocol, serial. HIDUINO is the response to these usability issues, permitting standardized driverless USB MIDI messaging between Arduino and host computer. One of the primary motivations of HIDUINO was to lessen time and skill needed to translate sensor

[1] http://mtiid.calarts.edu/class/interface-design-music-and-media

data into a musically useful format. In turn, we hoped to engage students with higher-level topics in interaction design, namely embodied interaction, tangible and wearable computing, and networked performance.

The class follows the basic ideas Perry Cook outlines in "Designing Principles for Computer Music Controllers as well several requirements described by Wanderly and Orio in [9], including *learnability*, *exportability*, and *feature compatibility* [10, 11]. Our selection of a suitable microcontroller for the class held these concepts in mind, applying them not only to the properties of a musical controller but also to the elements core to their design. Based on the research presented by Scott Wilson et al in [12], the pedagogical results of using Atmels AVR microcontroller appeared to meet many of these design criteria.

Since the Wilson publication, many AVR-based microcontroller platforms have been released, including the prominent Arduino. Our choice to move to this platform was motivated by the large community of support and open-source nature of many Arduino-based projects. Alicia Gibb describes the Arduino's growing reputation as an extensible platform for interactive media and goes on to say, "The design of the Arduino microcontroller caters to a non- technical audience by focusing on usability to achieve its intended goal as a platform for designers and artists," supporting one of our core criteria of learnability [13]. As the Arduino language is simply an abstracted form of C, we found that our exportability and feature-compatibility requirements were sufficiently satisfied by the ability to write and use low-level C libraries for more advanced projects.

2.2 On Protocols and Usability

Hans-Christoph Steiners paper, "Firmata: Towards making microcontrollers act like extensions of the computer, reveals a general problem in existing microcontroller platforms: communication protocols [14]. Arduino, and other similar platforms like Wiring and Gainer, implement virtual COM ports via USB for basic serial I/O between microcontroller and host. Since no major music application outside of Max/MSP supports reading serial directly, there is a significant disconnect between controller and application. For us, this limitation created a large usability gap in the platform which turned into the primary motivating factor behind HIDUINO.

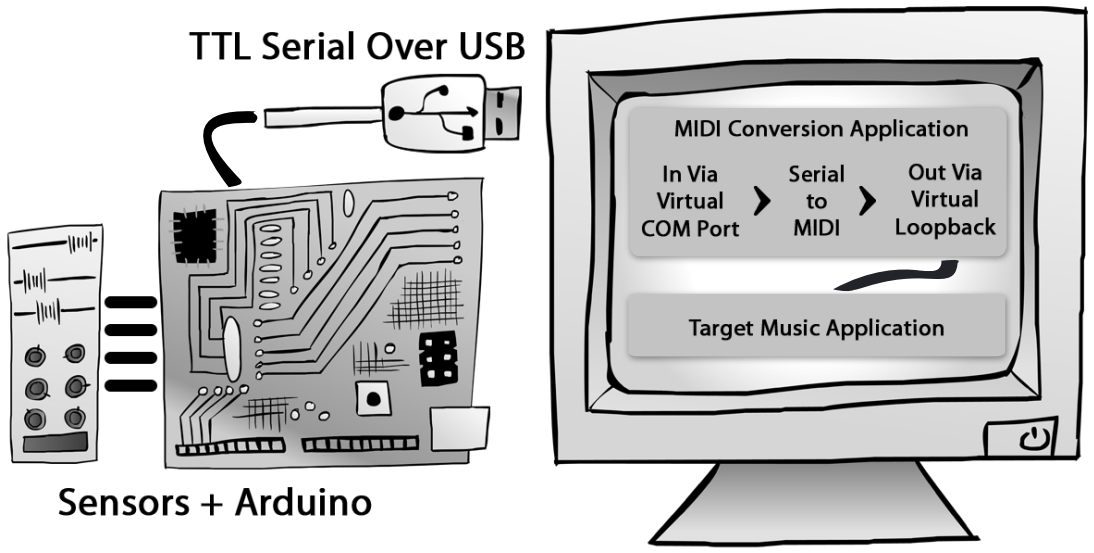

FIGURE 2.1: The pre-HIDUINO problematic middleware protocol translation process for the average student project.

We felt the major usability problem, for the purposes of an introductory physical computing class, was an over reliance on middleware for protocol translation. In the first 2009 offering of the class, each students project demanded of individualized instruction on account of the lengthy process of reading sensor data, packing it in a standard format, sending it to a computer, and finally decoding it into OSC or MIDI. While OSC is now seen as a more modern and desirable protocol for musical uses, MIDI is still the de-facto standard for commercial music applications and the format most students desire their data. In this situation, MIDI is particularly problematic since it also requires the use of virtual MIDI loopback drivers (or proprietary loopback software, in the case of Windows). In general, considerable time was added to the development process of many projects on account of this middleware translation software. Furthermore, we observed middleware decreasing system stability, introducing additional latency, and often creating a single-point of failure for many performance-based projects. In short, middleware-based solutions were too complex to teach and prototype in a single-semester introductory course.

After identifying the problem, our natural reaction was to research various existing projects and cases where people felt similarly burdened by the basic serial I/O of many microcontroller platforms.The first was look to the commercial market. The vast majority of commercial MIDI controllers on the market implement a standard protocol known as the universal serial bus human interface device (USB-HID). This protocol is often

viewed negatively by developers, citing its implementation complexity and bloat [14]. On account of these difficulties, few have been able to implement the protocol in a working form suitable for musical controller development. However, one of the more recent and successful microcontroller projects using USB-HID is the Create USB Controller (CUI) developed by Dan Overholt at UC Santa Barbara [15].

Prior to the 2009 Interface Design class, the CUI was the main controller platform on account of its native USB-HID support. Built on top of a Microchip PIC (a competitor to the Atmel AVR), the CUI comes bundled with a generic MIDI-HID firmware which sends out 12-channels of pitch-bend data, one for each ADC pin present on the board. In previous years, students in the class noted a few frustrations with the CUI, claiming the propriety development chain and necessity of low-level C required by Microchip complicated the process of making changes to the firmware. In short, it did not meet our expectation of usability.

Another project, uOSC by the CNMAT team, used the SLIP protocol to encode OSC packets as ethernet frames over serial [16, 17]. Unfortunately, packaging techniques like this also require middleware or driver-level software to support its use. Sidestepping the need for serial, we found several potential solutions in the form of hardware add-ons known in the Arduino community as shields. To begin, shields already add complexity in terms of assembly, wiring, and additional embedded software. The first shield tested, a MIDI breakout board from electronics supplier Sparkfun, still required that computers have a separate MIDI interface for the older-style 5-pin connector. The second, revision 2 of the official Arduino Ethernet shield, added significant cost and extra cabling as USB was still required for power. Moreover, students wishing to use their controller with commercial software still needed an OSC to MIDI conversion app.

In summary, neither existing hardware nor software solutions satisfied the need presented in the class. What other approaches were there? Luckily, the 2010 Arduino redesign opened a new avenue to explore.

2.3 Implementation

The implementation of HIDUINO was catalyzed by two recent events within the Arduino community: a redesign of the Arduino microcontroller and the support of an existing

USB-HID library by the Arduino team.

The 2010 revision to the Arduino platform introduces the UNO and the Mega2560, using the Atmel ATmega328 and the ATmega2560 chips respectively. Earlier revisions were equipped with an FTDI chip permitting users to interface with the Arduino via USB. The FTDI chip presented a few challenges to familiar users, namely that it required proprietary drivers on all platforms and could only act as a virtual serial port. The 2010 redesign omitted this chip in favor of the ATmega 8U2, the tiniest chip in Atmels lineup that included native support for a USB stack. The new Arduino includes a pre-loaded firmware which emulates the functionality of the older FTDI chip, but more importantly exposes the pins necessary to re-flash the 8U2 with the user's own custom firmware. This change opened up the possibility of writing a firmware that could conform to the USB-HID protocol specification and still communicate with the primary microcontroller on the Arduino.

LUFA Library

In 2008 Dean Camera started the MyUSB library, an open-source Human Interface Device (HID) library for the USB-compatible line of AVR microcontrollers. Later renamed LUFA, *Lightweight USB Framework for AVRs*, the project set out to create an elegantly-written library to demystify the USB-HID protocol. In contrast with a number of libraries written for the same purpose (detailed in the LUFA documentation), the API is incredibly straightforward. Indeed, even NXP Semiconductors ported LUFA to their ARM architecture on account of the well thought-out implementation. In addition, the library comes preloaded with descriptors for generic HID devices, including MIDI. Descriptors, as a core part of the USB protocol, instruct which drivers a host computer should use to interface with the device. Most operating systems provide built-in drivers for these USB class-compliant devices.

The Firmwares

The LUFA library already provides appropriate descriptors for USB-MIDI, thus the majority of HIDUINO code is targeted at structuring the communication between the

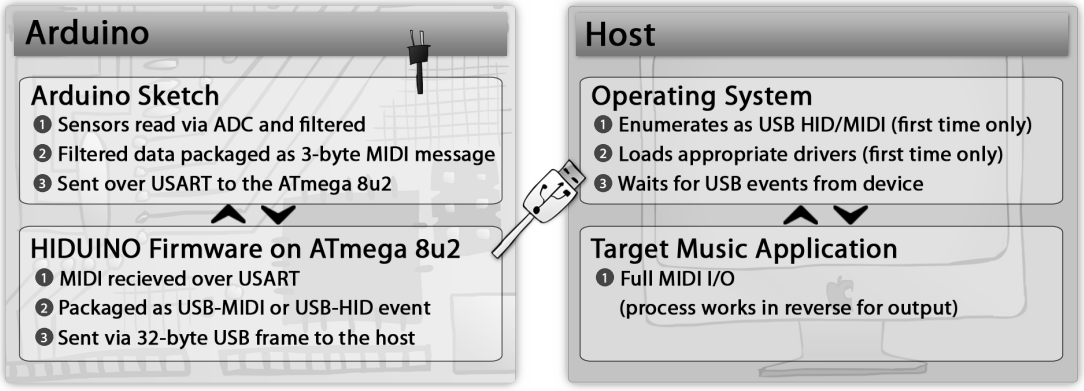

FIGURE 2.2: Overview of the hardware and software components with interconnecting protocols

main ATmega chip and the new 8U2. Figure 2.2 illustrates a full-system overview of a controller using the HIDUINO, showing the main functions of the firmware.

The main Arduino ATmega and 8U2 chip communicate over shared transmit/receive USART pins. Although the structure of the serial sent from the main chip does not matter on account of its later re-packaging as 32-byte USB-MIDI event, we decided implement a standard 3-byte MIDI protocol to standardize and leverage the existing MIDI library on the Arduino. Once the HIDUINO firmware receives a complete 3-byte message, it is checked for validity, placed into a USB-MIDI event container, and finally pushed over USB. The entire firmware was developed for bi-directional input and output.

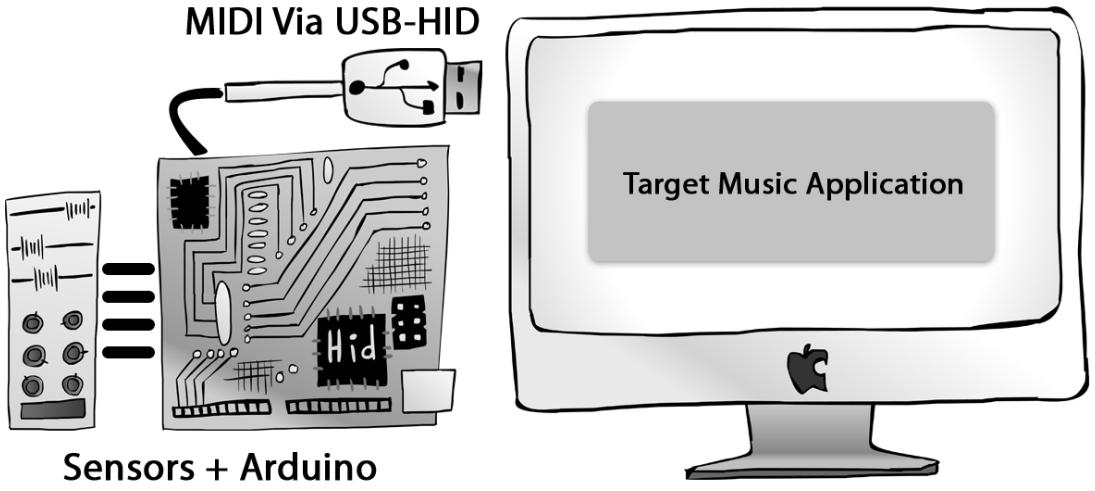

FIGURE 2.3: An improved model of the process presented in Figure 1, now with HIDUINO.

2.3.1 Process & Prototyping

For the class, the process for most projects was broken down into four steps. Although the following steps can be considered discretely, most projects iterate over the process a number of times between concept and prototype. In contrast to former methods using the CUI and middleware, the production process for the user has been simplified on two main fronts: (i) overall speed, and (ii) reduced system complexity.

- Design
- Signal Conditioning
- Flashing
- Testing & Debugging

The design phase exists to consider aspects of overall form, function, and effectiveness as musical controller. Students are encouraged to test and experiment with various sensors including potentiometers, soft potentiometers, force sensing resistors, buttons, accelerometers and gyros, photocells, resistive touch surfaces, proximity sensors, hall-effect sensors, and flex sensors. The output of these experiments often use a closed-loop LED for visual feedback or a string printed to the serial console in the Arduino IDE.

The next phase involves wrangling noisy sensor data into something artistically useful. In our experience with the class, most sensors connected to the Arduino require some level of conditioning and scaling before being used artistically. For example, oftentimes accelerometers have a low-pass filter applied to them or read soft-potentiometers are read with sample-and-hold logic. This conditioning is performed on the Arduino itself and printed to the serial monitor. After a user is content with the sensor readings, the Arduino sketch is ready to implement MIDI. Using a simple Arduino library for reading and writing MIDI over serial, a conditioned reading can be packaged as a note on, continuous control, or pitch bend message.

The flashing process can be accomplished one of two ways depending on whether a user has access to an in system programmer (ISP). Both ATmega chips on the redesigned Arduino have exposed in circuit serial programming (ICSP) headers. In our own testing and within the class, an Atmel AVR-ISP MKII was utilized to flash firmwares directly onto the 8U2. A second option is through the use of a bootloader. The DFU bootloader written by Atmel can piggyback on top of any firmware granted there is enough free flash memory. Our own personal preference is the ISP method as there are no pins that need to be set in order to enter the bootloading mode.

Lastly, the testing phase ensures that each sensor is correctly scaled and addressed by the right MIDI identifier.

2.4 Summary

The major achievement of HIDUINO was reducing system complexity by heralding a simplified, more usable process. Underlying several ideas presented by Owen Vallis et al in his 2010 NIME presentation, “A Shift toward Iterative and Open-Source Design for Musical Interfaces [18] , this project attacked complexity in artistic physical computing projects by iterating a problematic workflow and identifying and removing pieces of the protocol puzzle. The overall result was less time spent writing custom code and troubleshooting, which then opened up more flexibility in teaching philosphical techniques and practical method of interaction design for musical and creative applications.

Driven by a desire to ease the technical burden on students and teachers, the primary goal of HIDUINO was to dismiss the need for custom software and remove Arduino’s

dependence on the serial protocol. Harkening back to the mention of critical engineering in the introduction of this thesis, principle 5 states, *The Critical Engineer recognises that each work of engineering engineers its user, proportional to that user's dependency upon it* [3]. This principle sums up the HIDUINO project well: it suitably re-engineered and re-balanced the prototyping process such that a middleware dependency was resolved, helping both users and developers manage complexity in prototyping new musical interfaces. A secondary goal was in the realization of a utility that can participate as an input and output proxy in the scheme of a larger collaborative system, a principal focus of this thesis. The next chapter expounds the importance of HIDUINO as an enabler for custom input and output interaction modalities as part of The Machine Orchestra.

Future Work

With respect to future work, ongoing development includes the research and implementation of HIDUINO-like functionality on the latest generation of embedded architectures, namely ARM. Several popular microcontroller platforms such as the BeagleBone[2], mbed[3], Android ADK[4], and RaspberryPi[5] are approaching Arduinos level of beginner friendliness and usability. These next-generation boards have much to offer to students in physical computing, such as integrated graphics, ethernet, and full Linux-based operating systems. While the tools may already exist, these platforms will require similarly standardized processes for establishing a teaching-friendly workflow for computing projects with USB-MIDI and OSC components.

Code Repository

The HIDUINO project page is located online and includes a tutorial-style guide to prototyping interfaces with the firmware. Additionally, a guide is also available to optionally alter and compile the firmware from scratch. The current GIT code repository and project page is located on Github at `http://www.github.com/ddiakopoulos/hiduino`.

[2] `http://www.beagleboard.org/bone/`
[3] `http://www.mbed.org/`
[4] `http://developer.android.com/tools/adk/adk2.html`
[5] `http://www.raspberrypi.org/`

Chapter 3

Signal: An Application for Networked Performance

"The Critical Engineer expands 'machine' to describe interrelationships encompassing devices, bodies, agents, forces and networks."

Principle 6 of the Critical Engineering Manifesto

This chapter presents Signal, a client-server application demonstrating a collection of solutions to facilitate networked performance with musical robotics. A primary motivating factor for the development of this software was the wealth of challenges in collaborative performance introduced through the networking of The Machine Orchestra. Although Signal is specialized framework for a specific set of use cases – the control of a shared social instrument of musical robotics – many implementations-specific tools were designed to be portable and applied to similar collaborative systems. This chapter details the influence of other musical networking platforms on Signal, followed by a comprehensive documentation on the process and principles used in constructing the software.

3.1 Overview

Signal wraps a variety of tools with the intent to mediate creative experiences on a local-area network. As part of this mediation, Signal assists in defining a type of embodied

performance where multiple human collaborators can simultaneously control a shared social instrument of musical robotics. Historical systems for musical networking have focused on the network itself as a creative instrument. The Machine Orchestra toys with this idea by adding a component of real-time input coupled with actuated acoustic output. The input techniques of ensemble performers are often inspired by the curriculum in physical computing at CalArts. The specific type of musical interaction engaged by the performers is distinguished through use of custom, sensor-driven instruments. On the robotic side, the instruments are actuated by embedded microcontrollers listening for relevant control messages.

Chapter 2 presented HIDUINO in the context of teaching physical computing. In the Machine Orchestra, it plays a new role addressing the same protocol-translation process in reverse: the firmware is used as an output method for robotic control. The evolution of Signal has been informed by the use of HIDUINO and other software components and utilities, but aesthetic considerations have also been equally important in defining the behavior of software developed for the ensemble.

Many pieces written for networked ensembles employ a software as a piece approach to composition. The Machine Orchestra differs in that many ensemble members are performative electronic musicians who use a variety of hardware and software tools in their workflow. With the exception of Signal, no hardware or software standards exist between ensemble players – the choices are entirely personal. Similarly, members and guest performers bring a vast array of compositional knowledge across a great number of electronic genres. In turn, these details have commanded the architecture of the software to be maximally flexible in order to adapt the differing tools and techniques of ensemble members.

Signal evolved through these requirements to offer a core set of features and an extensible platform for future ones. The compositional process of the ensemble taught us that the interaction with software needs to be as minimal as possible to maximize creative output. We felt time should be spent on the music, not on debugging and troubleshooting. Consequently, this minimalist approach to software design also necessitated a focus on performance and recoverable runtime exception handling: the goal was not to have any war stories about crashes in the middle of performance. Moreover, we found that the performant aspects of local-area networks have a great effect on the cohesive playability

of music on a whole, especially given the rhythmic affinities of many Machine Orchestra members. On account of this, perceptual real-time synchronicity is a major technological challenge handled by Signal and sub-components and tools such as HIDUINO.

This chapter begins with an introduction to several pertinent problems in network music. Section A details how several artists and ensembles, both old and new, have considered the aesthetic influence of networks on music. Section A frames the iterative development cycle on which Signal has evolved from, taking into consideration the theoretical and real challenges of networked musical collaboration and the response of a shared social instrument. Section B, The Network Foundation, navigates several of the issues mentioned previously and is principally centered on explaining our software development processes and design solutions for dealing with an ensemble of networked robotics.

The work presented in Section A was performed in association with Owen Vallis whose ideas and language are jointly reflected with the authors.

3.2 Background

The Machine Orchestra stands on the shoulders of historic network ensembles such as The League of Automatic Composers and The Hub, as well as contemporary ensembles such as PLork, and SLork [19–21]. Informed by the aesthetic and technical contributions of each of these ensembles, we have undergone several iterations of an ad-hoc cycle to inform the requirements of our software. From an ongoing process of music composition, performance, evaluation, and development, our software reflects a number of insights to this process, jointly informed by two major contributions to the field of network music: Gil Weinbergs seminal work on network interactions [22], and lvaro Barboas concept of shared social instruments [23].

While these recent works have brought a lexicon to musical networking and allowed new theoretical discussion, the basis of music in this genre is rooted, like many other digital arts, in harnessing the creative potential of new technology. Founding artists in networked music demonstrated a primary interest in the potential of networks to simply share arbitrary and oftentimes chaotic data. Early experiments by The League of Automatic Composers [19] involved three networked microcomputers (KIM-1), each with its own custom software instrument, all sharing control data. This process of allowing the

performers to control each others instruments created an ensemble that had never before been possible, and led to the creation of a shared and social instrument, diffusing the absolute control a musician traditionally had over his or her own instrument. Members of The League named this new style of music *Network Computer Music*, and continued to experiment with new interactions well into the next decade.

The open-ended nature of the type and purpose of shared musical data has become a hallmark feature of networked computer music, with Brian Kane going so far as to say "Any aesthetics of Net music would, correspondingly, imply a set of musical practices that exploit these (and other) specific affordances of networks [24]. Additionally, while developing a classification framework for describing the multitude of possible network ensemble interconnections, Gil Weinberg states that he attempted " to map the field based on what [he sees] as the central innovative concept of the medium: the level of interconnectivity among players and the role of the computer in enhancing the interdependent social relations [22]. In developing his framework, Weinberg re-christened the idea of *Network Computer Music* as *Interconnected Music Networks*, a term embracing the availability of new types of musical data and the hybrid social and musical implications of its use in ensemble-style performance.

3.2.1 Embodying Performance

While these new methods of interconnectivity create experimental ways for performers to socially produce and consume musical data, they have also introduced new challenges. Complex networking schemes allow for the sharing of performance data between musicians, however, the connection between the music and the listeners – both ensemble participants and audience members – can be diffuse and unfocused. Oftentimes interactions are unclear and shielded by the hidden design of networked software, however simple or complex. Weinberg describes this as one of "the fields main drawbacks, in [his] opinion, stem[ing] from the focus that was put on complex interdependent connections which forced participants and audiences to concentrate on low-level analytical elements in order to follow the interaction [25]. He goes on to say in a later article, "these networks posed high entrance barriers for players by requiring specialized musical skills and theoretical knowledge in order to take part in and follow the interaction in a meaningful manner [22]. One of the original members of The Hub also expresses a similar idea

stating, "the audience was often mystified by what they heard in relation to what they saw the performers doing [20].

Weinberg suggests that "the design of expressive gesture-based interconnected instruments [would provide] participants with an expressive as well as coherent access to complex interdependent network topologies, which will allow them to focus on the artistic aspects of the experiences [25]. The instruments proposed by Weinberg would provide a strongly embodied link between the performers actions and the music created. The Machine Orchestras manifestation of this idea was in the conception of a shared, social instrument comprising an array of custom built electro-mechanical instruments, alongside the use of gestural interfaces used by the performers.

A shared instrument allows for multiple performers to express themselves independently within a social context [23]. TMO has created this kind of shared, social instrument using musical robotics which are accessible by every member of the ensemble, serving to directly embody the actions of the group through the physical movements of the robotic actuators [26]. While this is similar to acoustic ensembles, with each musician acting independently but cohesively, it is acutely different. Network ensembles afford musicians the ability to exist in or transition between multiple interaction contexts in a way that is arguably difficult or impossible to realize in music outside electronic media [27]. Graphically, the ensemble is represented in Figure 3.1.

3.2.2 Interaction Contexts

Modern software packages and graphical programming languages have increasingly afforded performers the ability to fluidly shift between different interaction contexts. These affordances have created an expressive space for musical exploration. Wanderley and Orio describe seven of primary contexts for interaction within interactive media, with the first three having direct applicability to sharing arbitrary symbolic musical events [**?**]:

1. Note-level control, or musical instrument manipulation (performer-instrument interaction), i.e., the real-time gestural control of sound synthesis parameters, which may affect basic sound features as pitch, loudness and timbre.

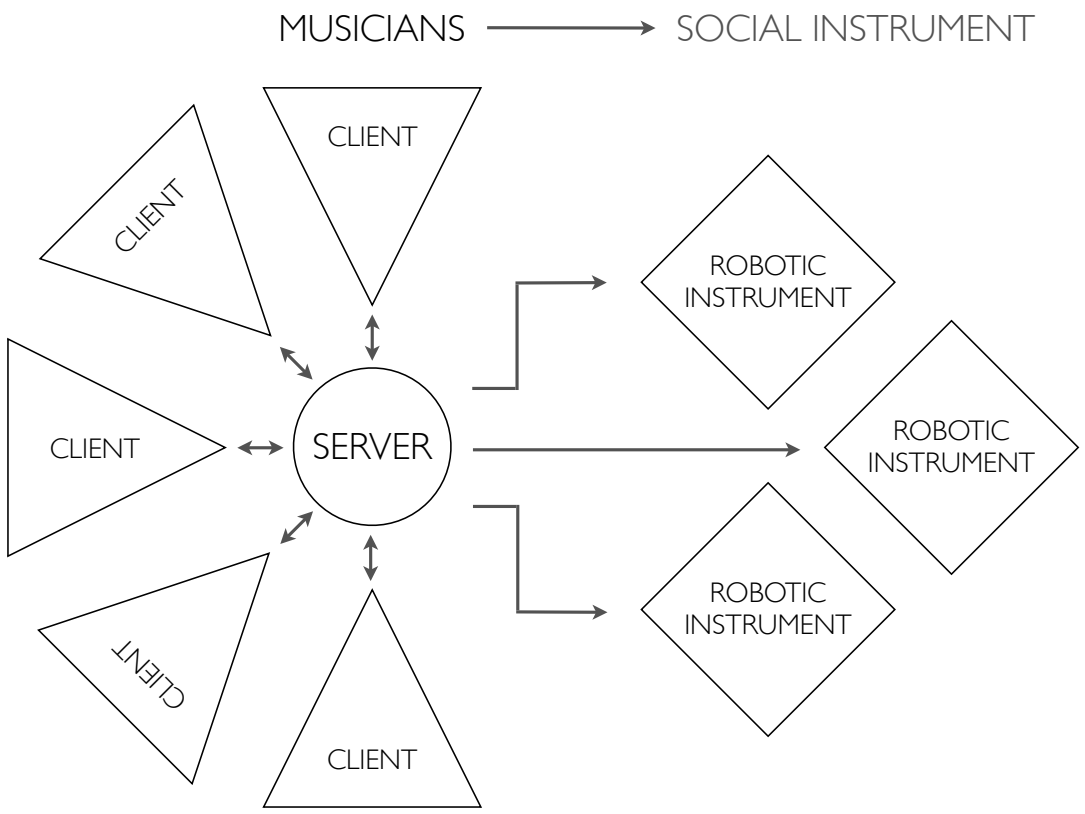

FIGURE 3.1: Graphical representation of The Machine Orchestra

2. Score-level control, for instance, a conductors baton used to control features to be applied to a previously definedpossibly computer generatedsequence.

3. Sound processing control, or post production activities, where digital audio effects or sound spatialization of a live performance are controlled in real time, typically with live-electronics

However, in a live setting, synchronism is a prerequisite of these contexts. Traditionally, musicians in an ensemble constantly adjust their playing to stay in time and keep the piece musically cohesive (unless a composition explicitly aims to subverts this); however, software is not able to do this conveniently. In situations where a very loose coupling of musical tempo is acceptable, approaches such as periodic sync signals or similar may prove adequate [28]. However, it is widely known that clocks in the average laptop are unstable sources and unpredictably drift, leading to the need for an external sync source or specialized algorithm when a high volume of time-critical messages need to be transferred – especially in the case of transporting audio [29, 30]. While it is possible to reliably sync for short periods of time, the major technical issue at hand is the need

for a centralized synchronization subsystem that can operate within these interaction contexts.

To functionally achieve these contexts in performance, a major point of this work is the functional specification by which a software platform can support shared synchronization among all nodes on a network, both human and robotic. In general, the requirements were empirically derived from specific musical ideas rather than generalized use-cases. As development began on the first platform in 2008, the major functional requirement was a common note-level framework for sharing data between performers and among robots. This was the first step in beginning our iterative cycle; with this first prototype, we were able to evaluate the definite musical needs and begin incrementally adding new features to the platform as the use-cases evolved on a per-piece basis.

With each release of Signal, we evaluate previous compositional sessions, rehearsals, and performances to identify musical traits that are not readily supported by the software. In many cases, these traits manifest through the addition of new performers or robotic instruments. For example, the inclusion of a robotic instrument capable of extended actuation instead of short on/off messages; or the necessity of a new ensemble member who desires to send an extremely high volume of messages to many actuators. The iterative development of the software introduced many new or optimized methods handling use cases such as these. In analysing cases where the problem could be abstracted or isolated to the network level (for instance, message timing problems caused by network jitter), we found that implementing known solutions within the academic community provided satisfactory resolution.

The following section details the most recent ideas behind the Signal software, offering core implementation details essential to the day-to-day musical operation of The Machine Orchestra. While the needs of this platform are understandably unique, we describe several key problems and solutions that we hope will be of further use to designers and musicians working with collaborative musical systems. Weinbergs works suggest a pattern language for musical networking – a collection of concrete assumptions and knowledge about both common and obscure design problems [31]. While this chapter does not describe the expansive scope of networking in robots, the primary purpose is to evolve the the process of designing collaborative systems that interface with musical robotics.

3.3 The Network Foundation

Signal is the latest iteration of an ongoing client-server platform for The Machine Orchestra. Prior TMO research [26] used the ChucK [32] programming language, and addressed two primary needs of the ensemble: networked performer-performer and networked performer-robot communication within a note-level context. The codebase developed for this initial customized architecture drew inspiration from the framework written by PLork [21]. Signal is a redesign of many message-routing tools developed in that codebase, now written in C++ using the open-source JUCE library. The transition to a lower-level language was motivated by TMOs perceived need for increased speed, routing, and network monitoring facilities. While the analysis of Signal as a research tool in networked composition remains ongoing, in part because of a quick cycle between musical idea and implementation, the contributions described in this section target three areas: network synchronization, protocols, and overall system design.

It is important to note that the tools and methods presented in this section are described without referencing the piece or performance where a problem was originally identified. In reality, Signal is a reflection of the rapid and incremental process between idea and functional specification. This decoupling should permit readers to envision similar contexts in their own musical networks.

Topology and Performance

The physical network topology on which Signal operates is a departure from several contemporary laptop ensembles. The design of a low-latency, low-jitter network was necessitated by the inclusion of a) musical robotics b) the need for tight synchronization between performers. We considered these prerequisites to building the performant note-level tools of the Signal software. Electromechanical instruments possess an intrinsic layer of mechanical latency that cannot be compensated for in real-time performance. Based on tests done by Mark Cerqueira for PLork [33], a decision was made to avoid wireless due to the added latency and jitter of consumer wireless equipment. In order to maintain temporal musicality across these physically dislocated instruments, and to minimize the potential need for jitter suppression algorithms, the ensemble is currently wired. At present, all networked data travels through an Extreme Networks 200E

unit, an enterprise-grade Ethernet switch. This topology compares favorably against work done by [34], which maintain that networked ensembles often work best within the 10-20 millisecond range. TMO latency metrics between the switch and individual performers indicate a round-trip overhead of no more than 1.20 ms under nominal network conditions (approximately 12 performers). All performers are wired using shielded CAT 5E in runs of 3, 7, 15, and 25 meters.

Server

Signal/Server is the central exchange for all data across TMOs network. Similar to Weinbergs idea of *flower topology* as part of what he calls a *synchronous centralized network*, the server application does not typically act as the sole source of data, but rather a collection of routing and exchange tools to facilitate data-sharing between networked nodes. Since the ensemble has a primary focus on local-area networked performance, the server application only handles symbolic music data, leaving high-bandwidth digital audio in the analog realm if it needs to be passed among performers. This decision not to transport audio was done in an effort to reduce possible latency degradation of the network due to congestion.

The server communicates with performers and robotic instruments via two standard protocols, OpenSoundControl (OSC) [35] and USB-MIDI, respectively. This redesigned Signal/Server prototype was implemented in C++ for the following reasons: increased stability as it no longer runs in a virtual machine; improved application speeds as C++ is one of the fastest compiled languages; and for GUI support through JUCE, which allows easy use of tools for data collection and visualization (See Figure 3.2 for an example of the client logging interface) . Speed is particularly critical in message-handling applications. Since OSC operates on the UDP networking protocol where packets of information are not guaranteed to arrive (unlike TCP), a C++ messaging stack essentially allows sufficient processing performance such that UDP packets are rarely, if ever, lost. In combination with a wired topology, empirical results indicate that message-handling through the server is fairly robust even without any packet-arrival guarantees.

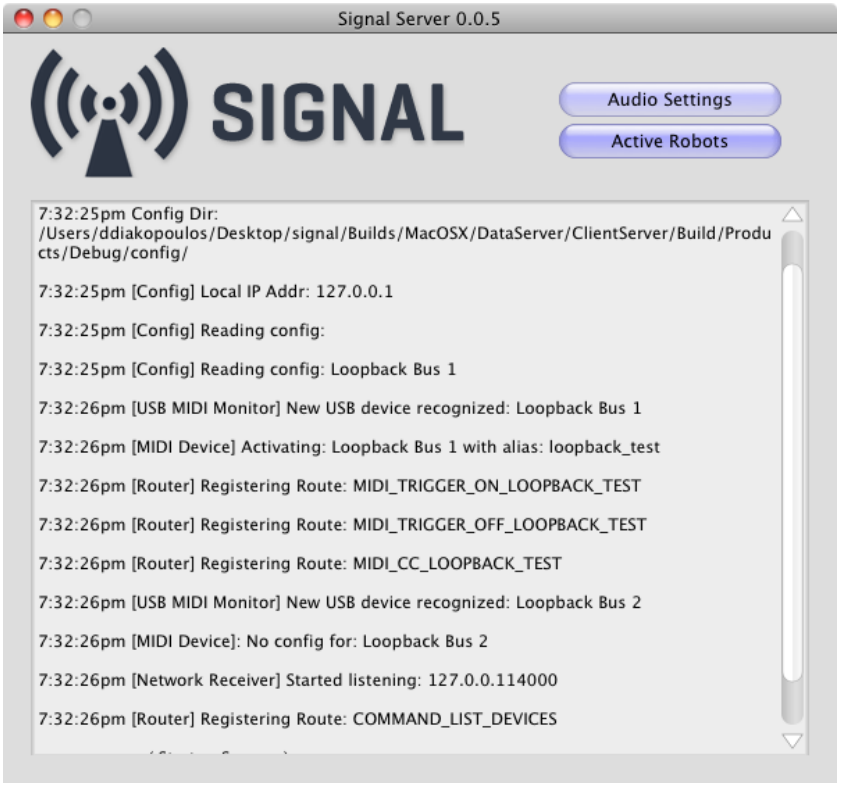

FIGURE 3.2: The Signal/Server logging window

Client

The range of software used for audio synthesis and robotic control in the ensemble is varied: ChucK, Max/MSP, SuperCollider, and Ableton Live are all commonly used in performance. However, not all applications can open network sockets and receive or transmit OSC data as required by the server. To mitigate this problem, the Signal/Client application was developed as a bi-directional MIDI/OSC translator. Early versions of the client were written in ChucK and provided methods to relay incoming sync data and map MIDI notes from host software to robotic actuators. In order for host software such as Live to communicate with this client, a virtual MIDI loopback port was used. The rewritten C++ client includes these methods with additional facility to remove the necessity of a loopback port, i.e. the Signal/Client application exposes itself to the operating system and host software as a standard MIDI device.

Going back to the idea that network problems should not interfere with the musical process, the client interface has a number of widgets and indicators to troubleshoot common problems such as network connectivity and actuator selection. A central window collects and aggregates relevant log messages between the host software and the

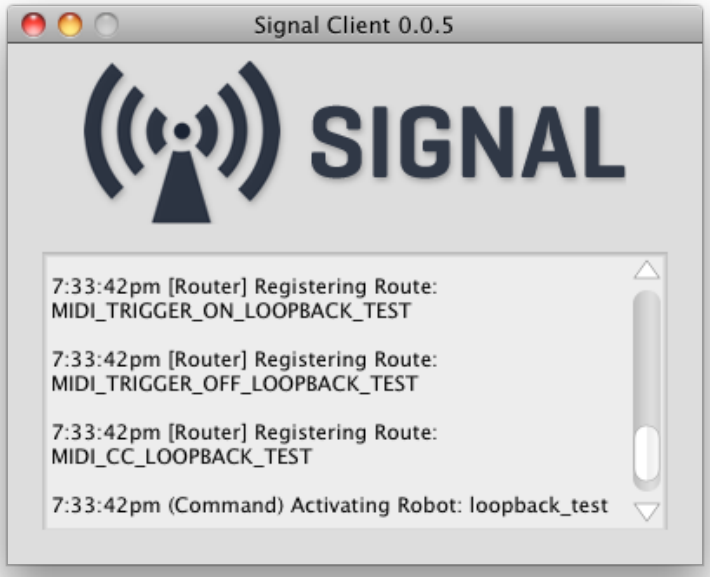

FIGURE 3.3: The Signal/Client logging window

Signal/Server. Indicator LEDs monitor the following conditions: server connectivity and network connectivity. In situations where advanced routing is needed in the host software, the logging window keeps keeps track of address, directionality, and payload of each message that passes through. Figure 3.3 shows the client logging window, one of the more polished elements at the time of this writing.

Synchronization

The first of our three principal areas of investigation was designing a system that could adequately handle tightly-synchronized rhythmic pieces. The Machine Orchestras music is highly dependent on a stable sync source for a number of performance contexts: rhythmic sequences between performers, tempo-synchronized audio effects, and note-level improvisatory playing. This aesthetic is derived both from the musical preferences of its members, but also by the affordances of musical robotics. While TMOs wired network affords sufficiently low latency, the issue of jitter, however minor, is still problematic when distributing a clock source with dynamic tempo. In real-world use, this jitter translates into minor (1-4 BPM) clock drift above and below the desired tempo. Finding the source of jitter can be difficult [33], and even though musicians can adapt

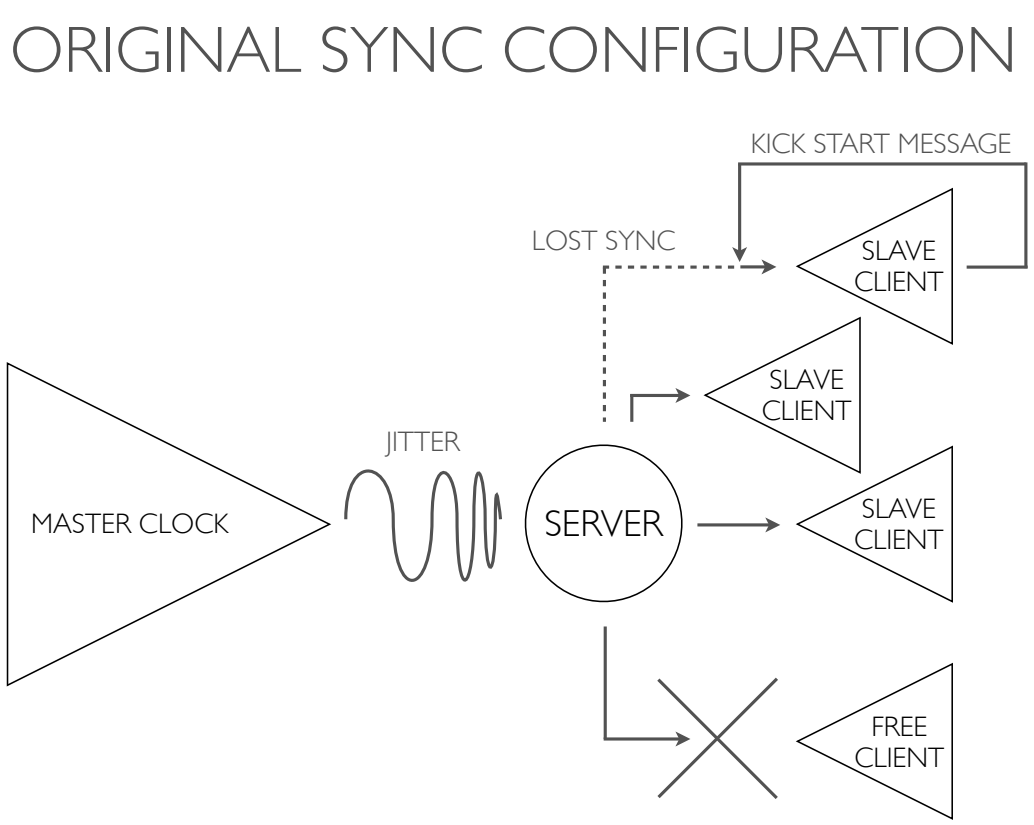

FIGURE 3.4: An illustration of the original synchronization scheme in the ChucK prototype

and perform with a fixed latency up to 40 MS [28, 34], tempo jitter as low as 1 MS is enough to significantly affect real-time performance [16].

An earlier iteration of the ChucK prototype (see Figure 3.4) distributed a clock signal through the server from a conductor client via MIDI at a standard 96 pulse-per-quarter (PPQ) resolution. The jitter from this conductor-server to server-clients relay was far than ideal, though we found it predictable enough to use in performance. Early efforts at combating jitter in the ChucK prototype were focused on implementing basic suppression algorithms such as low-pass and Kalman filters. By design, these filters are not particularly responsive to the rapid tempo changes which we found were very common musical devices in our compositions. In an effort to further address the problem, we were motivated to seek alternate solutions not reliant on a relay-based architecture.

The cycle between the ChucK prototype and Signal/Server & Signal/Client allowed us to rethink the way we distributed sync messages, as in Figure 3.5 Instead of a conductor relaying messages through the server, a master Signal/Client has control over

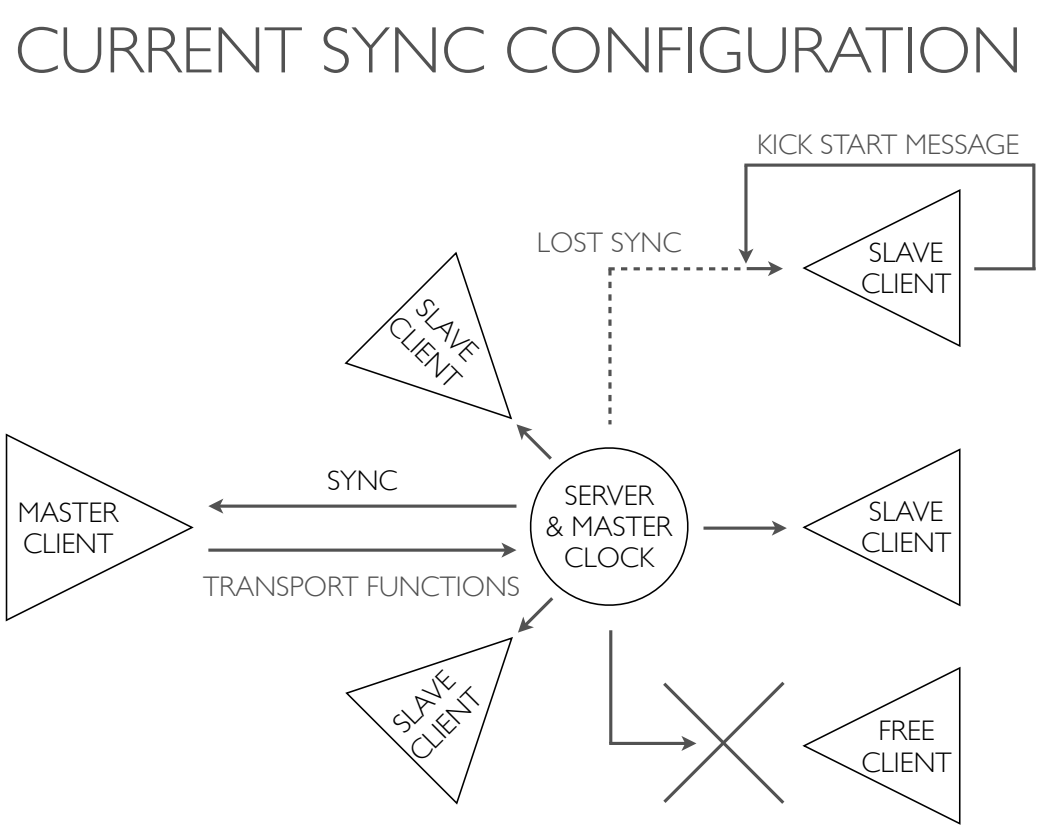

FIGURE 3.5: An illustration of the improved sychronization scheme in the rewritten client/server

tempo but leaves the physical generation of the clock to a high-priority C++ thread of Signal/Server. This method minimizes the amount of code necessary to distribute the messages and the number of times it passes through the switch. The move to compiled language with a full OSC library also led us to begin actively investigating the effectiveness of OSCs built in time-stamping functionality [16, 35]. The use of OSC timestamps would permit clients to compensate for the jitter present in incoming sync messages, while simultaneously allowing a fast adaptation to global tempo changes from the server. Although using the timestamps would require a buffer to allow for late timing messages, causing a fixed-latency delay, the delay should be well within human reaction times [36]. Current research is exploring this functionality and, while no current quantitative statistics exist to evaluate performance, our empirical observations suggest this approach sufficiently addresses this issue of clock drift.

Protocols

The second area of focus was standardizing and optimizing the necessary protocols for a functional musical network. Historically, the client and robots have communicated through a mixture of protocols including OSC, MIDI, and serial. Several problems were apparent with the previous ChucK prototype. Firstly, synchronization support on the client side mandates that we transport MIDI sync over OSC, and secondly, the robotic instruments were controlled via serial, a protocol not supported by ChucK. The resultant system was a convoluted mix of protocols and conversions on both client and server, which negatively affected software performance and *debuggability*. Although we satisficed with these tools for several performances, these attributes negatively affected the compositional process of the ensemble on the whole.

The response to this protocol was to standardize all network communication using OSC, considered by many to be the de facto protocol for musical networking. Since a number of musical applications only permit MIDI synchronization, we began with designing a standardized messaging scheme to pass MIDI sync messages from the server to client over OSC. The resultant performance penalty incurred by ChucKs virtual machine in processing the onslaught of messages considerably increased jitter and would sometimes cause a client to fall out of the clock stream entirely. The MIDI specification requires that the master clock send stop/start messages for slave devices to reconnect properly. This would have disastrous results in live performance, so we introduced a kick-start feature into the client. This feature produced a start message, allowing any software applications to be immediately reinserted into the clock stream (FIGURE). The transition to Signal alleviated many of the jitter problems; however, we decided to keep the kick-start feature as a matter of convenience in the event of any unforeseen network hiccups.

The musical robotics built for TMO previously communicated via TTL serial over USB to an Arduino microcontroller. In order for a performer to communicate with a robot, a note-on message would undergo a complex path to its destination. The server would receive a message as OSC, locally relay it over OSC to a Java application, and then finally re-package and send as serial. This convoluted man-in-the-middle software made pinpointing messaging errors and connection problems difficult. The design of Signal coincided with HIDUINO, allowing the robots to speak true MIDI over USB and negating the use of serial and this extra application [37]. This project was instrumental to the

architecture of Signal/Server and our goal of standardizing on only two protocols, OSC and USB-MIDI.

Design Patterns and System Usability

Our final focus was analyzing and improving the usability of the development process. First attempts with the ChucK platform were not concentrated on developmental simplicity nor designed with user ease-of-use in mind; many features were architected quickly as a proof-of-concept for the iterative model of development. In the users domain, only a simple Java GUI was available for manually triggering the robotic instruments directly on the server. While the rapid prototyping process between development and musical composition was able to accommodate the ensemble, the lack of debuggability and increasing number of interrelated messaging tools started becoming a hindrance. To address these issues in the new Signal platform, we were compelled to research patterns to other open-source networking applications.

The first design principle of the software concentrated on breaking out a lot of the variables and logging statements nested deep within the codebase to a user interface. One of the most familiar patterns for structuring interfaces is known as Model-View-Controller (MVC), where the View and the Controller represent the visual and interactive components of a model, a collection of data [38] The premise of this pattern was to give order to the myriad of data that could be of use to composers and performers: active clients, connection status, message traces, and sync status, among others. From a developers standpoint, structuring GUI code in an MVC permits a lot of boilerplate display code from being duplicated, often a cause of an anti-pattern known as spaghetti-code.

While MVC is a popular for GUI applications, it does not work as a design pattern to structure client-server messaging architectures, the vast majority of functionality in Signal. In software engineering, the terminology and methods for describing the architecture of networked applications can be varied. In his PhD thesis on the design of network-based software architectures, Roy Fielding presents a fairly consistent vocabulary for describing the specific elements of networked software /citepfielding. Fielding describes i) *components* ii) *connectors*, and iii) *data* as the primary building blocks of networked applications. When abstract ideas like *flexibility* have been previously mentioned, these are the elements on which most of the messaging system in Signal has

been built, where independent modules are built with elements similar to i, ii, and iii. Another pattern, iv) *configuration*, is used to describe the relationships between modules and data that needs to be persisted or manually edited. An example of the robot notifier is given next.

A good example of the architecture within a single module in Signal is the robot notifier, a function of the server to inform clients of the robots currently available on the network. The primary *component* of the module is collection of all the functional code to poll the USB module for changes. *Data* keeps track of the presently active robots, essentially any arbitrary data structure. *Connectors* are small functions visible to other modules used to describe the methods for getting or setting data. In actually notifying performers of the robots, the notifier module has a callback for a connector in the primary OSC module, which separately maintains the infrastructure for sending and receiving messages. The strength of this approach in development also allows for significant re-use between modules. As most data through Signal is transient, most modules share a common data definition known as a *Signal* (and now you know where the name comes from!), an agnostic structure for describing the origin, destination, and payload of a message – very similar in concept to OSC, although used as an interapplication protocol.

Signal uses the *configuration* pattern extensively through JUCE as a way to ease development and user-facing client interactions. Two key configuration interfaces were developed for use with Signal: one for robotic instruments for use on the server, and another for performers clients. For robotic instruments, these definitions enable Signal/Server to understand basic properties like name, number of actuators, velocity requirements, base note-to-actuator mappings, and permissions in the case of restricted or broken actuators. The Signal/Client configuration holds a mixture of user-specified and server-supplied data, including unique identifier, IP address, permissions, and master-client status. Both configurations exist as JSON files (Java Script Object Notation), an easily parsable format for storing key-value data.

3.4 Summary

Our discussion of interaction contexts represent the unique ability of live computer musicians to extend the performance capability of a single individual through the help of

networked software. We have demonstrated how Signal, the result of iteratively constructed software designed to accommodate our own musical needs, addresses interconnectivity and musical embodiment while only being explicitly designed to handle ad-hoc uses cases. In concentrating on the musical demands of the compositions and their relationships to the performers and robots, we have actualized a platform that mediates networked performance without being tied to any one musical idea. One of the main themes in this work is that the success of Signal is dependent on the effectiveness of tightly-integrated relationships with specific compositions, its own implementation, musical practice, and resultant performance. Through our process of incremental design, dictated by ever-changing compositional requirements, performers now have access to a musically expressive platform from which they can move through different interaction contexts seamlessly and transparently.

By carefully examining our needs as they arise from our compositions and software development process, we hope to offer many technical observations that potentially have great influence on future musical networks: Programming languages matter – the lower-level, the better; Timing is critical: balance the affordances of high-level clocks with their machine-level equivalents; Protocols are important – OSC is the linga-franca of musical networking; An easily-extensible GUI is important to quickly and painlessly debug issues, both musical and technical; networking frameworks should be flexible and able to accommodate the development of new interaction contexts without fundamental architecture changes; and lastly: that the music should dictate the technology, and not the other way around.

Moving into the future, we hope the ideas presented by our integration of musical robotics into a networked ensemble may influence the direction of new and existing networked ensembles. The lessons derived from the conception and realization of Signal should should be relevant and applicable to a wide variety of interaction contexts outside The Machine Orchestras own needs. While the novelty of a cycle between musical idea and programmatic implementation may be limited, following such a pattern in a loose but structured way has a great potential to illuminate and reveal many potential avenues for musical exploration that may not have been apparent had the system been fully designed and implemented from the start. We stress that iterative development is key for networked ensembles and hope other ensembles may be able to adopt similar paradigms in the realization of their own musical ideas.

Chapter 4

Netpixl: Designing a Toolkit for Synchronous Mobile Systems

Netpixl is a new micro-toolkit facilitating the development of artistically-minded mobile and social applications. The toolkit is a collection of server-side components for handling groupware-level problems commonly encountered in ubiquitous computing applications, expressly real-time message routing. Netpixl, in part, takes advantage of recently adopted HTML5 standards for real-time web applications through the new Websocket protocol. While many UbiComp applications, toolkits, and frameworks are often limited in scope to a single platform such as a tabletop surface, Netpixl aims to reduce the development overhead of building multi-user networked applications that can be integrated across a diverse number of mobile devices and protocols.

4.1 Overview

So far, this thesis has presented a relatively linear progression of projects that demonstrate physical and digital abstractions in the design of collaborative systems. In contrast to applications like Signal that have been developed for narrow use cases, Netpixl introduces a toolkit-layer abstraction on which many multi-user applications can be authored, whether interactive installation or artistic experiment. What differentiates Netpixl from ideas presented earlier in this thesis is the notion of ubiquitous computing. Within the

creative coding community, there is a broad movement exploring the capabilities of mobile computing platforms like phones and tablets. Mark Weiser's vision of ubiquitous computing has not yet achieved adoption on the grand scale he originally envisioned [39], however the proliferation of sensing and I/O technology supported by new mobile devices suggest many creative and pervasive applications beyond e-mail and games.

With the advent of new standards for real-time browser-based networking, Netpixl is in a unique space to integrate well-described network topologies in new artistic contexts. Chapter 3 first introduced the work of Alvaro Barbosa and his classification schema for network music. Within the broader sense of CSCW, this schema works well to describe the varied approaches to projects that are both synchronous and multi-user. Specifically, Barbosa describes a space with two axis: synchronous and asynchronous in the time dimension, and remote and local in the space dimension [23]. Like projects presented earlier in this thesis, Netpixl has been motivated by applications firmly in the synchronous, local dimension. The reasons for this are twofold. One: there is a greater potential for an interplay between sociality and creativity where collaborators are located in the same physical space. And two: interaction has a lower barrier to embodiment in low-latency, high-bandwidth environments (e.g. mobile devices sharing a local WiFi hotspot). The primary high-level goal of Netpixl is to help developers build applications within this schema. Towards this, netpixl has been motivated by challenges from three interrelated facets: developer experience (DX), toolkit design, and multi-user interaction.

Netpixl identifies and offers a solution to the fragmentation of technologies and toolchains used in developing multi-user applications. Too often, ubiquitous systems are limited in scope to a specific platform or programming language. This is particularly relevant in the Apple iOS ecosystem. Oftentimes, binary-native software enables superior performance in graphics or DSP applications, however, this creates a situation of non-portability that is problematic to any effort toward ubiquity across platforms. Going back to a lesson learned from Signal: if development time is spent with (platform-specific) implementation challenges, creative intentions can suffer. Recognizing this, the conception of Netpixl was partially driven by the adoption of the HTML5 API standards by many mobile device manufacturers. This standardization has opened a number of doors that were previously only available to native application developers, paving the way for a new array of Javascript libraries and frameworks on which many useful abstractions can be

built to assist creative coders footnoteFor an overview of some of the recent and more useful Javascript libraries and toolkits, see `http://microjs.com/` .

Adopting researcher Saul Greenberg's conclusion when talking about how toolkits evolved groupware applications, Netpixl aims to provide a foundational structure for addressing integration-level problems in multi-device applications. Greenberg specifically noted:

> Toolkits make it easier for researchers to create new breakthroughs through rapid prototyping of many new ideas. They let others replicate and evolve ideas reported in the literature. They also let researchers move more easily into empiricism by making it easy to create different versions of testable systems [40]

The power of HTML5 in the mobile and ubiquitous networking space can be expressed through an acronym: CODE, *create once, deploy everywhere.* A true ubiquitous application *should* be capable of deployment across many devices while maintaining synchronous links to other users and parts of the environment. Similarly, a software developer should be allowed to focus on the artistic aspects of multi-user collaborative interaction while being spared concerns of low-level networking and synchronicity. Netpixl identifies several key design and implementation attributes of other frameworks for creative coding and ubiquitous computing and offers several useful abstractions that can assist a developers ability to CODE with less code. This type of ability is a function of a toolkits general developer experience. While the true premise of CODE lies in the structure of the end-application itself, Netpixls purpose is to support CODE by marrying browser-based applications (using the Websocket protocol) with a server-side structure for communicating with a host of other communication protocols commonly used in creative computing (including serial, OSC, and MIDI). The design of Netpixl was approached with a minimalist design philosophy following a literature review of several software design patterns and their impact on overall developer experience.

The last facet concerns a meta-challenge in collaborative artistic works. As ubiquity transpires through mobile devices, a question can be asked of the the intersection of embodied interaction and sociality: where is the line between audience and performer? The notion of audience participation is an emerging topic, as seen by the large number of new works dealing with this topic in the past two years [41]. In practice, this idea

is seen as a common interface, either hardware or software, socially consumed among participants to manipulate transformative aspects of a work. Mobile devices are readily positioned to engender these types of interactions. Netpixl means to approach this form of audience interaction design from a technical angle. One of the main functions of the toolkit is the ability to decouple interaction methods from an underlying networking layer. Doing so permits developers to design collaborative interactions across many types of devices (tablet, phone, desktop), without spending much time wrestling with communication.

The major thrust of this chapter describes the design challenges of the Netpixl toolkit. Greenberg has described how toolkits should encapsulate common design concepts described by the research community into programmable objects that can be reused in many projects with little implementation overhead [40]. True to this idea, a major design principle of Netpixl is the extensibility and reusability of server-side components. Challenging this principle was the process of researching and abstracting appropriate conceptual CSCW and HCI models [42] into an application structure that suggests, but does not enforce any specific models of creative groupware interaction.

Chapter Outline

This rest of this chapter consists of three primary parts. Section Two starts by outlining the background and motivations of the Netpixl toolkit, briefly introducing the GR1D project (Chapter 5) and discussing an influential precursor project, Argos. Section Two also summarizes some of the more notable and relevant technologies, techniques, and projects in the browser, networking, mobile media, music computing, and toolkit-design spaces. Section Three centers on describing the design principles behind the toolkit, firstly framing the envisioned use cases, and secondly illustrating how those cases translated into the functional requirements of the toolkit. The chapter concludes in Section Four with a short exposition of the technical implementation of Netpixl, followed by a small discussion about the future direction of the toolkit.

4.2 Background & Motivations

One of my first experiences at CalArts was experimenting with social interactions through a DIY multi-touch surface. Surfaces afford a unique and direct combination of audio/visual/gestural interaction between co-located participants. The notion of social creativity intrigued me: what are the implications of shared environments in the artistic process? I have been continually motivated to answer this question from a software engineering perspective through various levels of developmental abstraction, from low-level sensor integration in HIDUINO through Signals complex OSC networking schemas.

When it came to evaluating the software requirements for a new installation idea, I began looking at my previous work and finding the technological similarities between implementations that supported collaborative interactions. It brought me back to one of my first explorations in the domain of social creativity, Argos. Argos was a multi-user interface builder for touch surfaces. The main lesson I took away from Argos was that surfaces have a very particular interaction modality that works well for some creative tasks but not others, as is often the case with new technologies. I learned that Argos itself worked well as a collaborative sketching environment for user interfaces, but those same interfaces were not useful for more than two users when applied to an audio performance application. The implementation ended up being too tightly coupled to the technology easily iterate on the multi-user idea. However, in this case it was also critical to remember the goals of the application in the first place: as a catalyst for creativity, not an exposition of technical ability [43]. Sometimes an inflexible implementation leads to other interesting research avenues, however at a cost.

Flexi-modal is a befitting term to ascribe to projects like Argos, denoting an ability to flexibly and adaptively mix interaction modalities [44] such as multi-touch, audio, sensor, and gestural. Netpixl was originally conceived to be the software layer of an an architecturally integrated installation (presented next in Chapter 5), where the hope was achieve some level of flexi-modality based on ideas from previous projects while abstracting away some domain-specificities. The differing technology stacks, however – each with their own context-dependent abstractions and implementations – made little opportunity for code reusability from earlier projects, despite obvious similarities.

Netpixl attempts to synthesize some aspects that have been presented earlier into a minimal yet modular platform. Namely, these aspects can be summarized into three areas: message routing, support for multiple interaction contexts, and synchrony among users and devices. These ideas can all be framed within the general idea of extending performance attributes into the audience through UbiComp-style mobility. Of this artistic statement in particular, I wanted to harness the recent surge of mobile devices and use them in a situation where the line between audience and work was diffuse. But, there was no catalytic application, framework, or toolkit suitable to kickstart the development. Thusly, the reasoning behind building a toolkit like Netpixl became apparent.

This section continues with a literature overview of the various ideas that have defined the methodological, functional, and theoretical foundations of Netpixl, starting with Argos. While the spread of work may at first seem unfocused, all of the following projects have attributes or themes that have directly or indirectly impacted design and implementation decisions throughout Netpixls lifecycle.

4.2.1 Related Work

Argos

Along with Signal, Argos influenced the early conceptual prototype of Netpixl. One of the main motivations of Argos [45] was to create an open-source competitor to the JazzMutant Lemur, the first commercial multi-touch device for music control [1]. Since Argos was tied to the expensive and lo-fi sensing methods of DIY multi-touch surfaces, it was of limited practical use, even though collaboration was an intrinsic feature through its table-based design. Since the original iPad 1 and subsequent generations, a number of interface-building applications have been released. For one, the Lemur was re-released for iPad in 2011, although by then other apps had already risen to meet the demand, notably Hexler's TouchOSC, and Control.

Argos was also a reusable touch widget library, though it did not see widespread popularity likely on account of it's C++ lineage and limited documentation. C++ has a well known reputation for being an unfriendly language for rapid application development

[1] http://www.jazzmutant.com

(although this has been recently changing with projects like openFrameworks and Cinder), and other libraries such as PyMT [46], MT4j [47], and MrMr [2]. SurfaceEditor [48], and others [49] created an easier path to multi-touch development. A common detail throughout many of these projects is the use of OSC for inter-app communication. Back in 2009, there were few examples of multi-user interfaces sending OSC, although the Audicle was a promising development in that direction [50].

Artistic tabletop and groupware computing is a popular and well researched domain [51], with notable projects like reacTable [52] and AudioPad [53] being inspirational to many future projects [54–58]. Within the past two years, the trend in multi-touch computing has shifted away from the table and toward mobile, in no small part due to the popularity of Apple's iOS and the surge of low-cost Android tablets [59]. Alongside a vague artistic intention to involve audience participation, a recognition of this shift is was what compelled me to look at mobile devices for the GR1D project; while many possible paradigms of interaction shifted, a common multi-touch input modality reminded me of the positive group outcomes noted while researching Argos.

Mobile

The capabilities and proliferation of mobile devices continue to fulfill Mark Weiser's idea of ubiquitous computing as evidenced by their recurring use in an emerging number of collaborative task and creative-oriented environments [39, 60]. One of the most visible results of this in the music community has been the development of a number of mobile phone orchestras, the mobile analogy of a digital music ensemble or laptop orchestra [61, 62]. As these ensembles have matured, several open-source toolkits have been released to help popularize the idea.

In a recent survey on the input modalities of mobile music interfaces, respondents noted that the rich availability of library/framework resources were one of the biggest advantages of using mobile devices [59]. MoMu is one of the first toolkits to offer abstractions for audio applications and was developed by the Stanford MoPho group [63]. Developed for iOS, MoMu offers a means of unifying sensor access and routing messages both within an app and over the network. UrMus is another recent toolkit, with a general focus on

[2] `http://mrmr.noisepages.com/mrmr-control-protocol`

evented interaction for UI widgets [64]. Design-wise, UrMus encompasses several valuable strategies that have been previously identified as being of interest to Netpixl. First, UrMus expounds the idea of interaction neutrality, meaning that it is an environment capable of assuming many possible paradigms without enforcing any particular pattern (i.e. it is flexi-modal?). Second, the toolkit is designed to permit design-by-molding, in that UI patterns can be remixed and modified in an effort to allow exploration. Both MoMu and UrMus are similar in that their operational modes are on the devices themselves, not specifically within a client-server topology.

jQMultiTouch (jQMT) is a toolkit abstracting different multi-touch APIs across browsers [65]. One influential idea jQMT expounds is the position that browser toolkits should be implemented in a lightweight, non-coupled fashion. Ideally, developers should be able to easily mix focused toolkits over choosing a monolithic framework like UrMus or MoMu. With Javascript frameworks like this becoming the norm for browser development, the design of Netpixl was influenced to keep specific interaction modalities separated to third party libraries like jQMT. Another influential idea in this domain can be found in the Damask toolkit, which forces designers to think of user interfaces in terms of patterns that can be reused across multiple devices, rather than visual widgets or components [66]. Like jQMT, this paradigm can be valuable when developing multi-user environments, because it helps maintain a loose coupling between device platform and application. In general, loose couplings enhance developer experience by compartmentalizing features while remaining modular.

One recent project that goes beyond fully client-side interactions (either natively or in the browser) is Control, an application designed for programmatic interface consumption on mobile devices [67]. In Control, developers can define interface layouts in Javascript and push them to clients running the application on a local network. An interesting attribute is the coupling with message-routing backends in languages like SuperCollider and Max/MSP. Control interfaces are automatically bound to an OSC address namespace that can be easily be read by preconfigured objects in SC or Max. One of the design goals of this project is to create a 'black box' around OSC networking to simplify application development, but also exposing functions to extend the black box. The notion of a black box is a useful abstraction for toolkit functions, hiding some of the harrier parts of an application to keep a developer focused on their primary creative task.

Lastly, an attractive feature of mobile devices is their utility in environments where an audience may take a direct role in the output of the work or piece [68]. Jieun Oh presents both a broad review of audience participation with mobile devices in [41] while presenting the latest experiments with Stanfords MoPho. Oh documents three participatory prototypes: audio sampling the audience, leveraging social networking services, and utilizing interactive web applications. This last model, based on a client-server relationship, is particularly relevant to Netpixl since it demonstrates a high level of interactivity though presents scalability problems. Weitzner et al introduces a similar client-server model in [69], called massMobile. Real-time is supported by massMobile, though the framework also targets asynchronous communication for high-latency environments. The authors of massMobile acknowledge the challenge of balance in participation/coherence and features for mapping, processing, and audience feedback. massMobile is described as not addressing these challenges directly, but instead offering an agnostic set of tools to encourage the rapid and iterative development and evaluation cycle, very similar to the development process introduced in Chapter Three. Audience participation as potentially mediated by the Netpixl toolkit is discussed more thoroughly in the next chapter.

Browser Communication Techniques

Thus far, this section has reviewed several projects each with differing approaches to the development of creative applications for mobile devices. Here, we will shift the focus to projects demonstrating utility in networking, particularly those closely related to newer HTML5 APIs.

CSCW literature is filled with discussions on the theoretical and real-world constraints of developing and deploying groupware systems. Toolkits like [70–72] help structure distributed systems, though they all face the same issue of platform-specificity. While the web has been acknowledged for a long time as an emerging CSCW platform [73], synchronous real-time networking in the browser has been an unmet challenge until recently. Several projects have found workarounds for this type of functionality, namely using plugin technologies such as Flash. Platform support for plugins is varied, and design philosophies like CODE preclude their use in ubiquitous applications since availability can not be guaranteed [74]. Since early 2010, several methods of real-time communication have achieved widespread adoption, for varying degrees of real-time. Gutwin

et al. present a performance overview of these methods, finding the HTML5 Websocket protocol as being the most performant and reliable [74]. Websockets resemble standard TCP/IP networking sockets in functionality, however sandboxed to the browser for security reasons.

Platform-native applications can utilize TCP/IP sockets, where we see familiar protocols like OSC being implemented [35]. For historic security reasons, application code running in a browser has always been restricted from direct access to native sockets, precluding the use of protocols like OSC. Effectively, this has made real-time nigh impossible in the browser. Newer web applications use AJAX polling and XHR multipart streaming to simulate real-time updates [75], but their performance characteristics have excluded their use in some creative applications where reactivity is critical [74]. Fortunately, the HTML5 Websocket API has seen growing adoption among browser vendors, opening the proverbial flood-gates of real-time.

By its relative youth, there are few examples of websockets in published literature. The WebSoDa framework is an experimental data-binding framework for keeping UI elements in sync between a server model and a client. These types of UI sync tasks are common in modern web applications and are normally accomplished through AJAX requests [76]. Marion et al. note a cooperative advantage in scientific visualization and present a collaborative data viz system based on websockets and WebGL (another HTML5 API). One end-to-end system application with a structure similar to the conceptual direction of Netpixl can be found in Patchwerk. Patchwerk is a multi-user HTML5 UI that communicates with a control-voltage (CV) module in a modular synthesizer [77]. Outside of the literature, there are an emerging number of multiplayer games and music environments, as seen in work like Rumpetroll[3], Plink[4], BrowserQuest[5], and Technitone [6].

Javascript Frameworks

Several frameworks have recently popularized the use of websockets as the next-generation of web application development. Although not specifically designed for creative coding,

[3] `http://rumpetroll.com/`
[4] `labs.dinahmoe.com/plink`
[5] `browserquest.mozilla.org`
[6] `www.technitone.com`

these frameworks are useful in concert with specific aspects of micro-toolkits like Netpixl since they address a common feature in collaborative or multi-user systems: database persistence. Derby[7] is one such framework, also employing websockets to sync client properties via server-side database changes. Another similar framework is Meteor[8], which performs the same type of real-time database synchronization. Meteor also introduces and their new Distributed Data Protocol (DDP), a simple way of importing arbitrary data streams into Meteor databases. In both frameworks, a publication/subscription foundation is used to notify all (or some) clients when any particular model property is updated. Lastly, it is worth noting the rising popularity of the Processing[9] programming language as implemented in Javascript. ProcessingJS `processingjs.org/` is fairly monolithic in its function as a creative coding environment for graphics and audio, however its implementation makes it fully compatible with websockets, making a perfect companion to server-side toolkits such as Meteor, Derby, or Netpixl.

Role of the Toolkit

One final area of review relates to the design and architecture of software. Netpixl is a code-level toolkit for developers and is thusly impacted by many aspects of software design. These aspects are particularly critical since the API of a toolkit is indeed an interface and one of the only usability dimensions for a developer. This subsection entertains the review of several works in the area of toolkit design as they have influenced the design goals and implementation of Netpixl. For a deeper look, a lengthy survey of frameworks and toolkits for Ubiquitous computing can be found in [78].

One of the important questions in a project like Netpixl is the simple, *why a toolkit?* While we saw a definition-by-example of toolkits as presented in the mobile section above, answering this question with a concrete answer will help describe much of the related work in this section. Saul Greenberg in his work "Toolkits and interface creativity," gives a particularly succinct and useful definition:

> Toolkits in ordinary application areas let average programmers rapidly develop software resembling other standard applications. In contrast, toolkits

[7] `derbyjs.com/`
[8] `meteor.com/`
[9] `processing.org/`

> for novel and perhaps unfamiliar application areas enhance the creativity of these programmers. By removing low-level implementation burdens and supplying appropriate building blocks, toolkits give people a 'language' to think about these new interfaces [40].

Chapter 4 briefly talked about pattern languages. In this context, toolkits can be viewed as pattern languages for programmers that are tangible and reusable. They are the antithesis of one-off prototype systems and help encourage good software design. Greenberg, citing Csikszentmihalyi's definition of creativity, goes on to talk about the creative aspects of toolkits and the power they hold in prototyping, developing, deploying, and evaluating new systems:

> ... the creativity I am talking about is not the 'big C' creativity usually associated with breakthrough ideas, or the result of people explicitly learning exercises or formal steps to promote their own creativity [...] Rather, it is 'small c' creativity where people naturally develop their own ideas through copying and varying the ideas of others in interesting and unorthodox ways, and through testing their ideas via prototypes and learning from their experiences of what works and what does not. In essence, this is the type of creativity often described as part of iterative interface development. [40].

Previously discussed were toolkits for mobile devices, groupware environments, and web application development. In the meanwhile, the Websocket API has been functionally complete since early 2011 but there have been no toolkits facilitating the design and implementation of systems specific to the areas that creative coders work in. Developing a new toolkit in this area seemed a worthy task, especially given inspirational ideas through the work of Ben Shneiderman, who describes that tools supporting creativity are one of the grand challenges of HCI researchers [79].

The most complete work discussing toolkits for synchronous collaboration is Peter Tandler's thesis on collaboration within UbiComp environments [42]. Tandler's work encompasses and expounds many principles of good toolkit design, but was written before a time where browser-based groupware could be realized. In his own software, Tandler explained the need for a strong approach to design on the assertion that, "Different

forms of interaction have different characteristics requiring different abstractions and different handling of input and output [42]." Tandler emphasizes the use of his particular conceptual model to guide the development process of collaborative applications. He segments his model into three *design dimensions*:

The first dimension, separation of concerns, creates a distinction between five intentionally vague parts of UbiComp systems: data, application, user interface, environment, and interaction model. The second dimension, coupling and sharing, is concerned with the degree to which concerns are bound through their respective software layers. The third dimension is the level of abstraction. In this, four levels of abstraction are distinguished: task, generic, model, and core [42].

This conceptual model was immensely influential to the design of the Netpixl toolkit, but led to an important question, *what are the most important abstractions necessary in this model for a toolkit?* Claubia Iacob turned to a review of apps with synchronous collaboration to look for design patterns [80]. Of the dozen or so she found, the following were particularly relevant to the goals of Netpixl:

1) Eyes wide open addresses the problem of allowing each collaborator to be notified about and visualize what the others are contributing to the process at any time.

2) Customize collaboration points to providing the collaborators with the possibility of customizing the parameters of their collaborative process. These parameters could be simple visualization or editing options, or more complex options such as assigning roles and rights among the collaborators.

3) Resume collaboration suggests allowing the collaborators to pause their collaborative process, store its state, and restore it later without affecting any collaborator's contribution.

4) Adapt application to device suggests supporting the materialization of the application on various devices so that users are allowed to collaborate even if they use different devices for that.

These patterns clearly weave throughout Tandlers design dimensions. In understanding some of the technical ideas that may support these design patterns, one of the most popular methods is though evented interaction: low-level user events bubbling up to trigger higher-level tasks which in turn influence parameters of the environment, for

example. To quickly dive into some technical details, evented interaction might be described as an action through which a server notifies the client or vice-versa when an action has occurred. The evented model is particularly useful in groupware applications which use a WIMP or telepointer-type modality [81]. Tandler argues against using this approach so granularly, instead preferring to track application state via shared objects or models. Websockets are particularly suitable for this task because changes to the model properties on either side, client or server, can be pushed immediately across a network. Consequently, a higher degree of development usability can be achieved as a programmer does not need to handle separate channels and events for each user, instead relying on toolkit components to make and synchronize model updates. Recently, this shared-state approach – often used in videogames – has been adopted in musical applications as well, indicating its variable utility in completely different contexts [82].

4.3 Design Goals

From the related works, a number of themes can be derived from the projects discussed. One theme is that toolkit developers can approach solutions narrowly or widely, be it interface development, mobility, or ubiquity. Another theme is that well designed abstractions can help a developer code more effectively. Yet another theme is that the recognition of design patterns, and their implementations as toolkits, can help differentiate extensible software and hastily-assembled one-off prototypes. Lastly, the availability of browser tech for real-time communication can be seen as a ripe area for exploration, particularly for creative coders.

Expressing these themes in a new toolkit creates an interesting problem in specifying the scope, in characterizing the domain, and in defining abstractions within that scope to help reflect new solutions. Luckily, groupware and synchronous collaboration is a fairly well defined and researched domain within CSCW and HCI; Tandler has already provided a immensely useful conceptual model. However, with technologies like Websockets, the scope has changed significantly: the groupware is not limited to custom platforms but rather any that supports HTML5. As a result, common techniques of groupware development need to be re-evaluated as functions of reusability and portability. To restate an earlier idea, the overarching goal of Netpixl is to offer creative coders a set of reusable code-level abstractions for networking disparate systems, devices, and

ultimately people. While programmers have been tackling these tasks for decades, there is always momentum toward harnessing the latest languages, protocols, and techniques in order to advance the art.

The term flexi-modal is an important trait that was previously described. Other traits like this help identify the salient aspects of design that need to be considered in a toolkit. Several researchers have presented similar taxonomies in the music domain, but they can be seen as equally applicable to CSCW systems. The first dates back to 2003, where Tina Blaine and Sidney fels review many of the pertinent issues in collaborative musical systems in a review of approximately two dozen projects. The complete classification is available in [27], but several are listed here: Mapping and Control, Location, Media, Scalability, and Player Interaction. More recently, Ian Hattwick and Marcelo Wanderley present an abstract explorer for evaluating such collaborative systems. Their taxonomic structure includes a graph with three axes with the following attributes: Texture, Equality, Centralization, Physicality, Synchrony, and Dependence [83].

Several terms are considered below as they assisted in the definition of the functional requirements of the toolkit:

- Equality - how do we treat and represent the models of systems, devices, and people?
- Centralization - what are network topologies that can best fit distributed users and devices?
- Mapping and Control - to what extent does the toolkit support input and output mapping algorithms?
- Scalability - does the toolkit define methods for manage multi-user scaling, or does it leave it up to the application?
- Physicality - at what level of hardware abstraction does the toolkit operate?
- Synchrony - what methods of real-time are best suited for a reusable toolkit?
- Media - what platforms are included or excluded as interaction targets?
- Flexi-modality - on what level of abstraction does the toolkit operate to accommodate a diverse set of interaction modalities?

4.3.1 Use Case Analysis

The responses to these questions may be best served with a generous helping of context. Interaction contexts were first introduced in Chapter Three and presented within a very narrow application: musical robotics. Netpixl was envisioned with the ability to work within similarly defined contexts, however married to a flexible and platform-agnostic architecture. The best way to describe these contexts are with several theoretical use-cases derived from the related work, affordances of the Websocket protocol, and early ideas for the GR1D installation.

Case A

Many toolkits have been described that assist in the development of native applications on Apple's iOS. The first and most apparent use-case of a new Javascript toolkit would be to replicate some features in the browser. Such features might include reading sensors, rendering a UI, or communicating with a server. The main actors in this use-case would be developers who are interested in translating this functionality across disparate platforms such as desktop, tabletop, mobile, and embedded without interfacing with inconsistent or complex development toolchains.

Case B

A second use case is in the integration of networked systems. Many frameworks and toolkits exist to assist in the development of domain-specific applications, such as audio sequencers or multi-user instruments. Other domain-specific systems might be as varied as social networking tools or physical installations. The goal is to proxy input data from any input modality, location, or platform regardless of protocol. Support for protocols might initially include websockets, OSC, MIDI, or serial. Again, the main actors would be developers authoring real-time applications. A successful toolkit would allow coders fluidly exchange data across an environment with no low-level communication overhead.

Case C

A third use case is an exploitation of the surge of mobile devices. There is no shortage of innovation in the interactive arts, but the predominant model of exhibition is still performer-audience. The objective of a toolkit in this situation might be that works can easily integrate interactivity when audience members are equipped with compatible devices. Since a browser *is* the application platform, a WiFi connection is the only prerequisite; no native apps have to be downloaded. The actors in this case are both developers who need to design around a new model of participation, and the general public, who need to contextualize new methods of interactivity. The primary achievement would be a mechanism for rapid prototyping so new participative or democratic [58] models can be designed, deployed, and evaluated quickly.

4.3.2 Design Strategies and Principles

Toolkit development can be seen as a process of designing interactions for developers in the form of an API. To the unfamiliar reader, an API can be essentially described as a series of related functions which help an application programmer introduce new functionality in his or her software. In order to describe the Netpixl API, first the scope of Netpixls functionality needs to be concretely defined. Tandler presents a list of five categories that are frequently addressed in ubiquitous computing environments. These are: a) interaction with devices using different forms of interaction, b) collaboration of users supported by a wide range of devices, c) integration of devices in the environment, d) support for different tasks, and e) hardware interaction [84]. So, the question becomes: what areas of functionality should be left to the application developer, and what should be left to the toolkit? As we saw in the related work, many toolkits center around a narrow domain, such as interface design, input abstraction, or general audio/graphics. Inspired by the specificity of the jQMT library [65] and other functionally-delineated toolkits, a decision was made to keep the toolkit as minimally restricted to server-side protocol integration as possible. While Netpixl, through examples, could possibly *suggest* specific the use of particular libraries for input capture, database persistence, or client-side application structure, it was decided early on that usability of the core networking functions should take precedence over the availability of features for which other libraries are more appropriate.

An exemplar of this scope-limited functionality definition can be seen in the 'flexibility-usability tradeoff' design principle. This principle is explained as:

> The flexibility-usability tradeoff is a design principle maintaining that, as the flexibility of a system increases, its usability decreases. The tradeoff exists because accommodating flexibility requires satisfying a larger set of requirements, which results in complexity and usability compromises (Wikipedia's definition per [85]).

Complementary to this principle is the idea of *featuritis*, where designers or developers tend to "emphasize the number or novelty of features over core usability [86]." Don Norman also argues that as functionality increases, mental and physical clutter can severely hinder usability [87].

Relating to the terms above, Netpixl can smartly sidestep issues of physicality, flexi-modality, media, and scalability. Modeling these specific attributes can be left to the developer, while the toolkit maintains a focus on the core networking stack and how it relates to supporting equality, centralization, mapping, control, and synchrony in a system.

This exclusion can be seen as the manifestation of the interaction neutrality principle described by Essl previously [64]. This follows a strategy exposed by one of Tandlers design dimensions: a separation of concerns [42]. The technical scope of the Toolkit can be then distilled down to a single responsibility: *mediating messages through a simple API to facilitate communication between a browser-based app and other interactive elements.*

4.3.2.1 Five Design Goals

The considerations we have just discussed can be expressed as five design goals inspired by, and partially derived from, Resnick et. al's, "Design Principles for Tools to Support Creative Thinking [88]." The intent of each goal is to inform implementation-level solutions for our specific areas: equality, centralization, mapping and control, and synchrony.

1. **Support Iteration and Exploration.** Creativity support tools should emphasize the importance of designing through prototypes. For an application developer

using a toolkit, this means being able to easily reconfigure data models and mappings, change the direction of the concept, and creating new versions without rewriting large parts of the program. Resnick et al. call this goal *tinkerability*. In programmatic terms, the toolkit should offer an understandable API for adding and removing input and output targets on an environment-wide level while being agnostic to the application-defined interaction modalities. Data publishing and subscription should be entirely re-configurable without necessitating architectural changes to the networking layer.

2. **Keep Developers Unburdened.** An unburdened developer is able to achieve a creative idea more quickly and iterate faster without being faced with common groupware-level concerns, notably multi-user/device data distribution, concurrency, and protocol translation. The abstraction of this idea can be seen as a black-box development model, where certain aspects of the toolkit work as if by 'magic'. A good toolkit should provide the notion of 'hooks' for more flexibility in the black-boxes, but the constraints of added complexity should be well noted to the developer. Synchrony – the ability for shared model data to be distributed as close to real-time as possible – should be one of the prime 'magical' features.

3. **Encapsulate Domain Knowledge.** Domain knowledge can refer to a number of concepts with varying specificity. In this context, domain knowledge encapsulates well known models in synchronous collaboration, including data-sharing and concurrency. These features should be available with little implementation effort and support common client-server networking topologies. A developer should have access to ready-made abstractions for these concepts without having to know their underlying principles deeply.

4. **Speak a Common Language.** Toolkits, by design, can be implemented in a vast number of languages and thusly excludes users on other platforms or toolchains. Javascript is the primary language of the web, supported by all major browsers, and the target of an immense number of libraries abstracting everything from user-interface design to machine learning. A toolkit should be implemented in Javascript to leverage the availability of libraries and the domain knowledge of many exports. A common language means a standardized and equal means of representing task and model-level abstractions.

5. **Enhance Creativity.** By managing the complexity of groupware concepts through appropriate abstractions, a toolkit should enhance overall creativity. In providing the ability to remove the mental clutter of low-level implementation problems, and supply the fundamental building blocks of collaborative applications, a language should exist to describe novel and creative applications. This is firmly a meta-goal and a topic of further evaluation and research.

Evaluation

As previously noted, toolkits cannot be developed in a vacuum. The next section focuses on specific implementation aspects as they transpired while working on the GR1D application prototype; in reality, models are hard or impossible to verify without prototypes. GR1D is presented fully in the next chapter, offering a thorough but subjective look at the user-interactions with the the associated Netpixl application.

4.4 Implementation and API

This section is centered on providing preliminary documentation about the Netpixl server and API and its features for kickstarting multi-user application development. In general, the section is presented as a broad overview, leaving specific implementation questions to be answered by the source-code of the toolkit itself (although several code examples if the API are presented). We start by describing the technology stack on which Netpixl was built, reasoning through the decision to use an asynchronous programming environment. The section continues with an exposition of the APIs exposed to an application developer, offering several clues as to how higher-level application tasks might be assembled from lower-level API components.

Leveraging Asynchronous Programming Models

The asynchronous programming paradigm is uniquely suited for evented interactions, a cornerstone of many interactive systems. This paradigm can be leveraged on top of libraries in some programming languages (Pythons Twisted, Rubys EventMachine), however server-side Javascript through Node.js offers the lowest barrier to entry. In 2009,

Ryan Dahl began hacking on a server-side Javascript environment based on Google's V8 Javascript engine. V8 was written in C++ for the Chrome browser and received widespread acclaim due to its performance characteristics compared to traditionally compiled languages. Dahl's framework was named Node.js and quickly became a popular development platform, in part because because web developers could dive in quickly using familiar Javascript syntax [89].

Asynchronous programming can be confusing at first, though the paradigm is an excellent match for handling large volumes of synchronous events (e.g. messages) quickly. The software engineering field has different notions of synchronicity and events than CSCW. In Node language, when we want an event to be handled immediately (as close to real-time as possible) we oddly do not call it synchronous. Where many languages used a threaded approach to respond to single events, such as a click on a GUI, Node uses a single thread and allows events to trigger a callback when they occur. The current thread of execution is momentarily delayed while the body of the callback is executed. In comparison, a synchronous event handler has to sit idly while it waits for an event. CSCW-style synchronous systems benefit from the asynchronous style, veering from a resource-hungry threaded model. There are additional benefits and tradeoffs, though such a discussion would be beyond the scope of this chapter. Figure 4.1 demonstrates C-style pseudo-code differentiating synchronous and asynchronous programming styles.

Serverside API Design

The unified Netpixl API can be seen as a collection of functions within three tasks. These tasks can be classified as: i) data translation across protocols, ii) defining models that can be published to and subscribed from, and iii) maintaining application state across clients. A fourth meta-API includes common utilities for creative coding alongside a logging framework for debugging. The purpose of these APIs, to an application developer, is to assist the assembly of larger building blocks of functionality while abstracting out boilerplate and repetitive communication code.

```
// Asynchronous Style
while(true) {

    message = waitToRecieveMessage();

    // adds a message to a queue
    // will be processed when the thread has a free moment
    if (message.type == 'async') {
        handleMessageAsync(messageHandlingFunction, message)
    }

}

// Synchronous Style
while(true) {

    message = waitToRecieveMessage();

    // spawns a new thread to handle the message
    // increases complexity and consumes resources
    startMyThread(message);

    myThread(message) {
        if (message.type == 'sync') {
            // handle message
        }
    }

}
```

FIGURE 4.1: Asynchronous and synchronous programming styles

Messaging API

The central promise of the Netpixl toolkit is that it affords an easily-approachable model of message synchronicity across platforms and protocols. While employing Node.js removes some problems around message volume and scalability, a developer is still left with the task of consuming, saving, routing, and sending events throughout an application. Moreover, these events may not come just from a users browser via websocket or vice-versa, but an array of devices and software. An Arduino transmitting proximity sensor data, for example, or a browser client offloading sound synthesis to a remote computer. This idea extends not just ubiquitous computing via mobile devices, but pervasive computing as well. Pervasive computing, a popular research interest among

creative coders, describes ubiquity as a function of environment and sociality: within the context of this chapter, the marriage of artistic installations and mobile browser apps.

One of Netpixl's core APIs features the ability to consume messages from many sources and formats. This API was inspired by the separation of concerns principle set forth by Avgerigou and Tandler, who propose the use of a message exchange model that operates at the lowest level of abstraction (the native protocol) [90]. The message API is built on functions for consuming and producing messages through a variety of protocols, many of which are within the domain knowledge of creative coders. Node.js is compatible with libraries for reading and writing serial, OSC, MIDI, and websockets. However, a problem with these libraries is that each presents a slightly different API. The Netpixl messaging API smooths out inconsistencies by providing wrappers to manage and instantiate objects for sending and receiving. Messages consumed by receivers are converted to a common protocol-agnostic format and sent to the Netpixl router. Figure 4.2 shows a snippet of code revealing the differences in using the library-provided API for NodeSerial, and the Netpixl wrapper.

Pixls

So far, we have talked about data models abstractly. Shared models [40] are defined on the server in as a collection of Pixls, the fundamental unit of computation created for the toolkit. Events published a server are converted to an intermediary message structure and consequently routed to a related Pixl. Pixls are defined by an application developer and may represent an attribute of data (integer, float, array, etc), an endpoint (representing a task), or both. Since Netpixl was designed for web applications, the default subscription channel is websockets. Any Pixl that has been published to will emit altered data to connected clients. Additionally, when a Pixl is defined, a developer may manually add new subscribers in cases where events need to be sent through a transport other than websocket. Pixl creation is outlined in Figure 4.3.

Pixls follow a black-box methodology of extension. Within their primary function to relay messages around an application, Pixls may be extended in several ways that are invisible to developers until needed. The purpose of extending functionality is to help developers build tasks that may or may not be related to the unit of data being transacted: tasks model application behavior and are foundational to the collaborative design

```
// Without Netpixl ============================================

var serialport = require("serialport").SerialPort;
var mySerialPort = new SerialPort("/dev/tty.usbmodem1a1211", {
    parser: serialport.parsers.readline("\n")
});

// Reading Messages
mySerialPort.on("data", function (data) {
    console.log( data.split('\t') );
});

mySerialPort.on("error", function (err) {
    console.log(err);
});

// Writing Messages
var message = parseInt(543, 10) + "," + parseInt(345, 10) + "\n";
mySerialPort.write(message);

// With Netpixl ===============================================
var netpixl = require('netpixl');

// Less Verbose
netSerial = new netpixl.Serial("/dev/tty.usbmodem1a1211", {parser: 'default'});

// Automatic data formatting
netSerial.write( { data: {543, 345} );

pixlController.prototype.parseSerial = function (data) {
    // parse + format message here...
    // message is then tagged with metadata about when
    // and where it came from and subsequently routed to a Pixl
    pixlController.preprocessRoute(parsedMessage);
}
```

FIGURE 4.2: A demonostration of the the Netpixl wrapper for serial communication

patterns previously outlined. One of the most useful and notable task-oriented features of the Pixl API is the preprocess function. The preprocessor can be used in conjunction with the utility API to transform, filter, limit, scale, or compare data before it is committed to the model and published to clients. Both preprocessors and postprocessors are transparently built into the event handling chain. The flow of data through a Pixl is given in Figure 4.4 followed by an example of a preprocessing function in Figure 4.5.

Synchronization API

Synchronicity is a built-in attribute of the Netpixl toolkit. That is, all events are published in real-time to connected clients. Synchronization, departing from the definition given in the last chapter, refers to the present state of the model data across two or more entities. When a client connects to the application, it may be desirable to inflate

```
// Pixls are crated with a route identifier (fadeRGB)
// and an optional settings array
pixlController.addPixl('fadeRGB', {

    endpoint: function(data) {
        if (data.red === 255 ) {
            shutdownApplication(); // too red
        }
    },

    // add additional notifiers to the notification stack
    notifiers: {
        serial: function(data) {
            // optionally sanitize data and then send over serial
            netSerial.emit('command',  data);
        }
    }

});
```

FIGURE 4.3: Defining a Pixl. Notably absent is any hint to the actual data stored: Pixls store the most recent primative or JSON object sent through a message to a Pixl.

the local model with the present server data. This API assists in serializing an entire Pixl collection to JSON for easy transmission to a client. The sync API also permits user-defined functions to synchronize the model to a persistent data store such as a local database. The API additionally contains a Distrbuted Database Protocol hook into Meteor. Enabling DDP mode within Netpixl allows data to be persisted to Meteors MongoDB data store while also permitting Meteor, in its facility as an application platform, to act as the sole websocket data publisher. Lastly, the sync API may mediate the resolution of conflicts in model data. Pixls may be assigned a conflict mode and timeout value: different modes may be selected or defined by the developer when multiple messages are received within the timeout frame to update a known Pixl. At present, the toolkit only contains rudimentary resolution functions (discard, keep, interpolate), vaguely resembling more complex Software Transactional Memory (STM) algorithms. As before, conflict modes are easily extensible by a developer.

Utility API

Finally, Netpixl presents methods to encapsulate actions regularly seen in creative coding and web application development. For instance, the toolkit provides a basic data

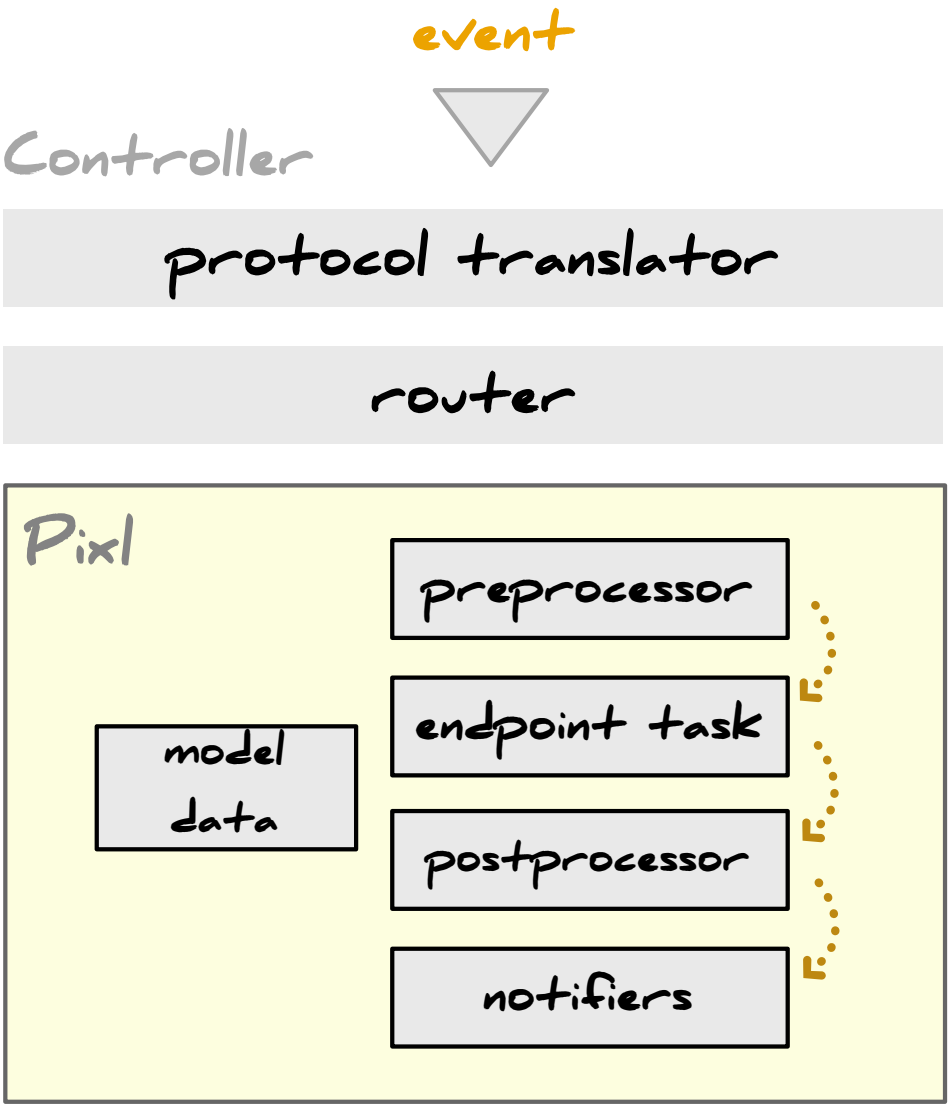

FIGURE 4.4: Illustrating the flow of an event through the Netpixl core

structure for representing colors along with common tasks (converting to HSB, representing in hexadecimal, etc). Inspired by Processing.js, these functions were designed to be used in conjunction with the preprocessors and postprocessors. Additionally, the API contains methods for scaling, filtering, clamping, and interpolating numerical data. Low-resolution timers are also available using Nodes built-in timer classes, permitting periodic function calls. Lastly, the utility API contains a robust logging framework to assist with application debugging. Pixls are pre-built with the ability to log messages in different stages, while the toolkit handles writing to file or over websocket.

```
var util = require('netpixl').utilities;

pixlController.addPixl('serial_ppExample', {

    preprocess: function(data) {

        util.map(data.x, 0, 150, 15, 100); //scale between 15 and 100
        util.clamp(data.x, 15, 100); // clamp values under 15 and over 100

        // Pixls keep history; linear interpolate between the last value
        util.linInterpolate(data.x, data.x, this.lastOutputHistory.x);

    }

});
```

FIGURE 4.5: Example of a preprocessing function

Browser Compatibility

Transitioning away from the server, browser applications built to communicate with Netpixl have no hard requirements other than a compatible Websocket implementation. Several Javascript libraries have built useful abstractions on top of the Websocket API. Of these, Socket.io is one of the most widely known. In general, client-side communication controllers only need to respect a very basic publishing and subscription format for client-server communication.

For prototyping and testing, the initial GR1D application leveraged a library called Backbone.js alongside Socket.io. Backbone adheres to a design pattern introduced in the last chapter: Model-View-Controller [38]. Backbone is a staple of many web applications because it helps developers manage views (collections of visible items in the UI) that can be easily managed by shared data models. Local changes to model properties can be triggered directly by standard events such as taps or clicks, or integrate with other libraries that bind pen, audio, camera, or accelerometer readings to event emitters. Most application development libraries such as Backbone (among a host of others) publish events when local properties change, making bindings to Websockets simple and straightforward. To a developer, this permits any arbitrary data attribute to be bound to a Socket.io event handler, creating the basic bi-directional communication structure between client and server.

4.5 Summary

Future Work

Looking to the future, Netpixl has recognizable areas for improvement, particularly toward the lofty goal of a seamless developer experience. An often neglected area of toolkit development is API documentation, which will be an an ongoing priority moving forward. As Netpixl was developed for and in conjunction with GR1D, the next chapter details its utility in the environment, but does not provide a formal evaluation. Toward this, I would like to spend more time wearing an 'application' developer hat and fully grasping the affordances and restrictions of the toolkit within several artistic situations, forming the basis for a complete analysis. Lastly, another area which the toolkit may provide support for is an automatically generated REST API for model data. REST APIs transact data through basic HTTP requests, permitting model information to be saved or retrieved without a real-time transport layer. NodeJS is an ideal candidate for building REST-style APIs given its wide adoption in that area, while the addition of a basic API for non-real-time communication would be a natural extension of the toolkit.

Summary

Here, I have presented a micro-toolkit based on an idea to ease the developmental burden of mobile applications for multi-user installations. This toolkit can empower developers by providing useful abstractions for browser-based applications and their networked communications, reducing the time needed to build and iterate collaborative systems. For coders with artistic and whimsical ideas, Javascript toolkits like Netpixl afford easier methods of interaction exploration and rapid prototyping, enhancing creativity and guiding new ideas and experiences to reality.

Chapter 5

GR1D, A Responsive Media Facade

GR1D is an interactive display that transforms the role and experience of an audience within an urban environment. Presented as an artistic light installation, the display uses pixels as a metaphor for artistic expression within a grand, architecturally-integrated presentation. Spread across such a large space, familiar one-to-one interaction of traditional addressable displays aims to be disrupted, engendering social play and permitting new situational interactions among users and their environment. The aggressive adoption of mobile devices has enabled the GR1D to leverage a new movement toward responsive environmental computing while maintaining conventional mobile interaction techniques. Special thought was given to the highly-visible nature of the installation, taking into account the context of the area and the culture of artistic people surrounding it at the California Institute of the Arts. This chapter primarily describes the design challenges faced in the conceptualization of GR1D, informing a discussion about future work in interaction design for responsive urban displays.

5.1 Overview

Integrating displays into an urban setting has become increasingly popular in the past decade. There is an emerging research field which considers the integration of display technology and lighting systems into urban architectural spaces. This discipline works

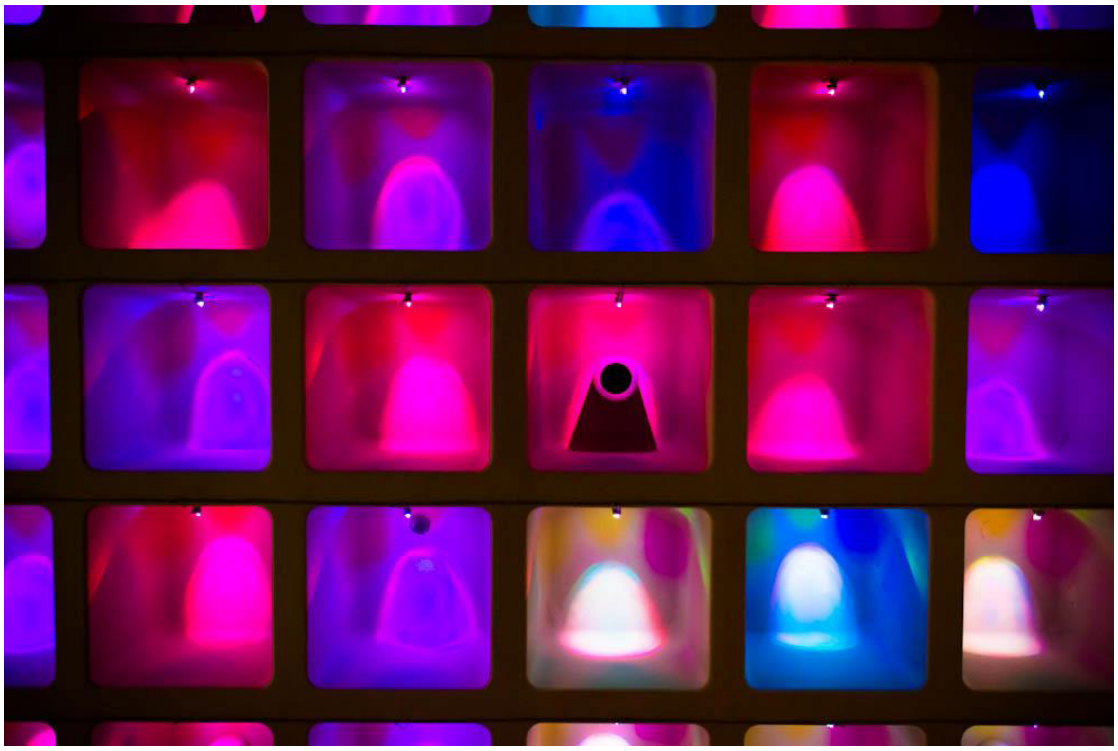

FIGURE 5.1: A point-of-view shot looking up at the facade

within a specific term to describe such environmentally situated displays: *media facades* [91]. Media facades are fundamentally different from mobile or desktop displays and must recognize attributes such as scale, resolution, visibility, and in recent cases, collaboration [92]. Interestingly, many existing implementations of media facades and urban lighting systems do not permit communication or interactivity, a key component of GR1D. Outlined in the next section, a growing body of research suggests that within the past five years there has been a significant push to understand the potential for new types of collaboration and interactivity afforded by a new breed of interactive media facades [93]. GR1D works toward extending the interactive display as a metaphor for universal computing by further exploring the the relationship between urban displays, control mechanisms, users, and audience.

The original model of ubiquitous computing as it was originally envisioned by Mark Weiser at Xerox PARC in 1988 is rapidly approaching reality [39]. Mobile phones and tablets are quickly becoming the default devices for applications in ubiquitous computing [94]. During the evolution and codification of interaction patterns with such devices, we have come to know the display the most visible element of computation. As such, the

addressable display is a fundamental building block in the way we have come to understand data and its organization, especially from the perspective of ubiquitous computing [95]. As a result, mobile device platforms are a familiar and accessible avenue of communication in which to engage media facades, considering both champion the use of a display as an ambassador of ubiquitous computing. In the consideration of GR1D as an artistic installation among artistic people, the approach to a mobile control mechanism was tightly linked with artistic applications over other task-oriented activities such as information visualization or gaming. The background section provides a more thorough explanation of interactive media facades and ubiquitous computing as it has influenced GR1D.

Combined with the pervasiveness of mobile devices and their corresponding usability, the creation of large-scale interactive displays offer several unique challenges in social interaction design. Here, we define social interaction design as the process of conceptualizing and realizing artistically-driven experiences that may evolve depending on public context. One challenge was determining the extent of which GR1D would disrupt the traditional one-to-one relationship of users to their mobile devices. The design of GR1D was informed to satisfy this disruption but not strictly enforce active or passive use. The degrees of participation in environmental control mechanisms has been classified by Verplank et al and can be measured as either pre-defined automated actions or continuous, absolute command [7]. While some media facades focus on the first idea of automated actions – a low factor of participation – the design of GR1D offers efficient and simultaneous direct control for a number of users but requires an active desire to participate. Sensitive to the diversity of contexts in which collaborative interaction can take place with GR1D, another challenge was in the consideration of context and characteristics of the architectural space and also variety of people who might be interested in interacting with the installation, minding the creative art-school mentality at CalArts.

The next section, 5.2, provides an overview of the current state of research in media facades, as well as projects that informed the design and development of GR1D. Following, section 5.3 elaborates on several core topics enumerated in this introduction, including the ecology of mobile devices and challenges related to physical space, social context, and social control. Section 5.4 describes the implementation in greater detail with layouts and schematics of the installation. The chapter concludes in section 5.5 with some specific thoughts about the process of integrating responsive displays into

urban architectural spaces as well as some critical reflections about the purpose and nature of ubiquitous urban experiences.

5.2 Related Work

A main objective of the installation was to conceptualize patterns around collaborative and engaging ubiquitous experiences. Ideologically, GR1D identifies with a movement known as post-industrial design. This practice reflects on the impact of aesthetics and arts alongside the theoretical significance of networks and code [96], a major theme of this thesis. Within this area, GR1D intersects the trending value of mobile devices as networked computational platforms and urban space as an avenue for social computing in an artistic community. Within CalArts, there is a concentrated and unique emphasis on the value of aesthetics. Considering this context, a primary motivation was to present a playful and creative installation with a low barrier to entry and the affordance of direct and simultaneous multi-user control. To describe the conceptual design of GR1D more concretely and associated approaches to context, control, and social interaction design, we follow with an overview of major work within the realm of interactive media facades.

Media Facades and Their Influence

The history of media facades is rich with research in ubiquitous computing, networking, information design, HCI, and responsive architecture. The seminal text on the history media facades is Matthias Haeuslers 2009 book, *Media Facades: History, Technology, Content.*[91] Haeusler sets up genres in the recent emergence of media facades and includes projects that primarily emphasize architectural ornamentation, news display, advertising, art, and gaming. This section reveals recent research within several of these genres and specifies interactive projects from which GR1D took influence.

Peter Dalsgaard is one of the leading figures in HCI-centric media facade research. Recently, Dalsgaard presented eight challenges for urban media displays which are based on Jonathan Grudins original eight challenge for developers of computer supported cooperative work (CSCW) [92, 97]. In his studies, Dalsgaard has worked closely with colleagues in designing and installing several major works the media facade sphere,

including Arhaus By Light [98], The Climate Wall [92], and the Dynamically Transparent Window[92]. Dalsgaards challenges are informed by research in urban informatics [99], participatory design [100], and more theoretical work discussing the imperative to understand situation and context in the introduction of interactive systems to new environments [101, 102]. In documenting his projects, Dalsgaard introduces a technique for designing media facades called the *Design Space Explorer (DSE)*, a method of focusing issues of location, interaction, aesthetics, and content during the design process[98]. Thematically critical to the later discussion of GR1Ds design principles, Dalsgaards DSE and eight challenges are presented in more detail in the next section.

The social affordance of public screens is thoughtfully discussed by Mirjam Struppek in her 2006 publication *The Social Potential of Urban Screens* [103] Since then, follow up explorations have yielded important discussions on the methods in which urban displays can be socially consumed and simultaneously controlled by a group of people [104–106]. Namely, these works stress the necessity of novel types of interaction design – mouseless, sensor based solutions – over traditional WIMP-style interface constructs.. In the related sphere of multiplayer game design, the edited compendium *Space Time Play: Computer Games, Architecture, and Urbanism*, [107] contains articles discussing the cross-influences of architecture and gaming. Several articles underscore a growing sentiment that the next-level in multi-user gaming must arise from the marriage of ubiquitous computing and architecture, a feat that is both technologically and socially challenging. In general, this synthesis of ubiquity and sociality is a popular and recurring theme in many other, non-game-based media facades, including GR1D.

Several researchers have addressed the technical facet of media facade control by introducing new interaction design methods and techniques using mobile devices. Although many facades use integrated sensor networks as control mechanisms, the next few projects exhibit new advances using mobile devices as a ubiquitous control platform [60, 94, 108]. One common technique is the use devices cameras and accelerometers for indirect pointing-based control, as seen in projects like MobiSpray [109]. These methods have been criticized as having limited applicability in busy multi-user environments since they rely on the concept of pointing. [110, 111]. Sebastian Boring et al introduce a new method without the use of fiducial pointing markers on a display[112] by using an augmented reality application that employs a mobile device's camera directed at the facade to allow users to point through the screen, the result of which is then reflected

onto the facade. In 2011, a modified version of this concept was used in the TouchProjector project, an installation consisting of a 1087 pixel display on the Ars Electronica building in Linz, Austria. The creators of TouchProjector only allowed three simultaneous users, though they reported several instances where users felt engaged with their collaborators. This model of locally-embodied social control served as a template for designing the interaction in GR1Ds mobile application.

There are several projects in the HCI literature describing projects using micro and macro-scale pixels as a fundamental unit of control. As mentioned in the next section, GR1D was greatly motivated by an architectural space at CalArts with natural pixels built into the space. Dating back to 2001, the Blinkenlights project predates many media facades in its application of a large low-resolution display with the windows in the Haus de Lehrers building in Berlin [113], similar to the recently mentioned TouchProjector. Other micro-scale projects such as DataTiles [114], Siftables [115], and DisplayBlocks [116] are pixel-like objects based on high-resolution displays for information display on an individual or networked level. Yet more projects like Nami[117], and and Six-Forty-by-Four-Eighty [95] take a low-resolution approach to pixel-based computing , finding applications in urban ambient information displays. UrbanPixels [118] was the first project in this category that integrated literal pixel objects directly into an urban media facade. Limited control were characteristic of these earlier projects, although the visibility and scale of the pixels created a unique and social experience regardless of the interaction. Blinkenlights could only be sent pre-defined animation scripts, while UrbanPixels was limited to mobile control via SMS message. TouchProjector was the first of these pixel-based projects to harness the computational power of mobile devices as a ubiquitous control platform to mediate shared control.

Many of the artistically motivated media facades are indeed situated within an area known as ambient media displays. While the use of GR1D as an ambient media display (AMD) was not a focus of the installation, the role of AMDs in the evolution of media facades can not be understated. Researchers in this genre have emphasized aesthetics in the design of their displays, here defined as the artistic influence on the implementation of the design requirements [119]. Installations in this category demonstrate the power of AMDs to trigger emotional response, enhance communication, and promote reflection beyond purely aesthetic qualities, a highly desirable factor of any media facade [119]. On the topic of interaction design for shared spaces, Nina Valkanvoa goes on acknowledge

the power of AMDs with artistic presentations, with examples of artistic influence in projects like Rafael Lozano-Hemmers Body Movies [120]. More significantly, Weiser cited the Dangling String installation by Natalie Jeremijenko as demonstrating the importance of the ambient media display in ubiquitous computing [39].

Throughout the work presented in this section, several themes emerge. One theme is that context and aesthetics need to be addressed with design-driven approach to maximize the impact and meaning of urban displays. A major reason for this might be that urban interfaces have a fundamentally different social practice over private life which must be conveyed through thoughtful interaction design. Another theme is that social control can create a dialog through a space as long as the methods of control are direct and engaging. To achieve this, smart mobile devices are a promising avenue as they have become the default devices for ubicomp applications. Lastly, macro-scale pixels are uniquely suited for media facades since they afford a highly visible unit of control, ideal for facades which may be engaged from a distance. The confluence of art, design, and technology is highly evident in the work presented here, and serves as a foundational basis on which to discuss the impact of these works in GR1Ds design considerations.

5.3 Design

Research Motivations

One of the first questions people ask when they see GR1D is, “Cool! Where did you get the idea?” To regale a little personal story, the idea appeared late one night when the author and his roommate were leaving CalArts after a long day working on projects. The entrance walkway to CalArts is covered by an overhang with hundreds of ceiling wells that look like the grid patterns on waffle irons. As we were walking, one of us noticed that the fluorescent lights in some of the tiles were in a regular pattern, although it was not immediately apparent since so many of them were burnt out. That was the a-ha moment where we realized we could be looking at pixels, not a semi-random arrangement of drab fluorescents. “Wouldnt it be awesome we made a display out of them!” we schemed on the way back to our apartment.

FIGURE 5.2: A distance view of the waffle-ceiling above the main entrance with an assortment of fluorescent lights

At first, the idea of the installation and the intent were not mutually exclusive. Only after a period of reflection did the purpose of GR1D surface. To reiterate, one of the primary goals of GR1D was to conceptualize an engaging and collaborative urban experience. This intent was largely motivated by the authors observation that, while many artists working at CalArts are pushing the boundaries of their respective disciplines, they do so in relative isolation: working in studios, displaying in galleries. All work is confined in small spaces within the main building on campus. Built in the 1960s, the layout of the building was designed to support this traditional art making and exhibition practice with a variety of studio and gallery areas. The authors objective for GR1D, and to a degree artistic statement, was to create a social dialog in its novel re-appropriation of existing architectural structure. This re-appropriation was central to engendering a collaborative environment that blurs the role of spectator and partcticipant.

Beginning shortly after approval – no one had ever modified the building of CalArts in quite this way – GR1D was designed over an eight-month period as the result of a practice known as Research Through Design. This method is not centered on generating research with an analysis of existing artifacts to formulate patterns, but rather an iterative model focused on a contribution of knowledge by producing an exemplar [121].

Material	Form	Combination	Location	Situation	Interaction Sensing	Interaction Style	Format	Content
Concrete, Wire, Arduino, Light, LED	Square, Hole, Line	Matrix, Grid	Main Entrance, Walkway, Ceiling	Waiting, Arriving, Departing, Resting, Performing, Socializing, Greeting	Passive	Movement, Gestural, Sound	Data	Ornament, Art, Pattern

TABLE 5.1: Starting Design Space Explorer

Research Through Design is intended to encapsulate research in interaction design, a model well suited to GR1D since many of the challenges in the media facade arena have been defined although many ideal solutions have not been explored nor realized. As one of the criterion for determining successful interaction design research is extensibility [121], the design of GR1D was structured around (i) studying previous techniques of collaborative interaction design and (ii) capturing knowledge about the constraints we faced in deriving improved or modified solutions.

5.3.1 Design Space Explorer

As mentioned in brief earlier, Peter Dalsgaard presented a Design Space Explorer for media facades, a structured approach to visualizing the prominent facets of design. It is intended as a platform for discussing the the initial design process, keeping the scope of the project in the foreground while contrasting multiple design concepts in an easy-to-consume format. The key descriptors in a DSE are: material, form, combination, location, situation, interaction sensing, interaction style, format, and content [92]. Several DSE tables were generated early in the GR1D design process for different scenarios (namely, artistically or info-design focused), finally settling on the one below as the starting point for an artistic implementation. A modified DSE is included at the end of this chapter to summarize the changes made throughout the design process.

The space presented in the explorer is interrelated to the themes brought up at the end of the last section, namely that thoughtful interaction design can support engaging social experiences with the help of urban pixels and ubiquitous mobile devices. In more

formal terms, the concrete challenges of creating a successful story around these themes can be constrained within a problem-area consisting of several high-level categories that have already been documented by Dalsgaard et al in Eight Challenges for Urban Media Facade Design. For brevity, the descriptions for each of the eight challenges are available in [92] but omitted here:

- New Interfaces
- Integration into physical structures and surroundings
- Increased demands for robustness and stability
- Developing content to suit the medium
- Aligning stakeholders and balancing interests
- Diversity of situations
- Transforming social relations
- Emerging and unforeseen use of places and systems

To arrive at a realizable conceptual design, these challenges were sorted and ranked according to how well they fit within the overarching goal of creating an artistic experience around the situations presented in the explorer. The situations themselves were derived from several categories put forth by McCullogh in [101]. Chiefly among these challenges, Diversity of Situations, New Interfaces, and Developing Content to Suit the Medium were identified as salient problem areas relevant to an artistic installation, although all are interrelated to some degree. The following questions were derived as these respective challenges were reframed during the process of conceptualizing, iterating, and evaluating potential solutions:

1. What are the salient angles of location and social context that will make the installation appealing to the community?
2. In the consideration of urban interface design, what control mechanisms and content will yield the most engaging experience given changing multi-user and social contexts?

The remainder of this chapter calls attention to these questions by discussing the varying aspects that led to the eventual realized form of GR1D.

5.3.2 Location and Social Context

The introduction of a media facade into an urban space is characterized by the fact that it may be experienced in diversity of situations and may directly or indirectly transform the context of those situations. Talking about staged urban interactions, Martin Brynskov et al state that it is not always obvious what types of social mediation is desirable and acceptable within these situations [122]. The notion of acceptable was conceptualized in the consideration of several factors, expressly that: (i) that the installation is situated in the entrance of CalArts, the most visible location on campus; and (ii) any interaction will invariably change the context of events that happen under or near the installation, although the extent is unknown.

The entrance to CalArts is essentially a social space, notably at night when most performances on Campus are staged. Given this preconceived notion about the entrance (as shown in Figure 5.3), it was deemed feasible that a mediated social experience presented as an artistic work would not be unexpected or unwelcome. To maximize potential engagement where interactivity and play incurs a higher demand of time, GR1D was designed to turn off during the day where time is more strictly appropriated to tasks like attending class and running to meetings. In this iteration of location constraints, weather was also taken into account. Given the outdoor location in Southern California, it was likely that the cooler night air would also be the most comfortable in which to engage an audience. A byproduct of this design decision was that the facade and ambient light would never have to compete, rendering technical issues of brightness and visibility inconsequential.

Understanding that the introduction of a media facade would undoubtedly change the context of some of the situations listed in the design space explorer, an early question was to what extent would control encourage participation. It seemed natural that, given the general societal unfamiliarity with interactive environmental experiences [122], that users should not be imposed upon with an excess of control ability. As stated stated earlier, Verplank et al studied degrees of participation in media for which he measured two main control mechanisms: pre-defined automated actions or continuous, absolute command.

FIGURE 5.3: Waffle ceiling above the doors to the main entrance

Given these degrees, interaction was first conceptualized with automated actions driven by a sensor network, influenced by the work of [123]. In this first ideation, GR1D was designed with a matrix of proximity sensors to to display colorful, movement-driven light patterns as people passed by or under the facade. In another, localized microphones would listen in and cast washes of light over the speakers where volume was loudest.

In conceptualizing use-cases for sensor-based interaction, it became clear that indirect control might be too limited for participants to create a meaningful social experience outside of the novelty factor. This sentiment was echoed by our literature overview of existing media facades, finding that the most engaging and successful ones echoed Verplanks classification of direct, continuous control [7]. We conceded that using a design based on sensor-based interaction with proximity or microphones could be responsive but ultimately too limited to satisfy the stated design goals. Another factor that contributed to this decision was in understanding that sensors had a high abuse potential in the case of excessive arm-waving of yelling, disrupting the context of nearby social situations in a negative way.

Direct user participation satisfied our model of engaged interaction at the expense of limiting which situations could be modified by the facade. Exhibition was then added to the design space explorer as a potential solution. Without sensor-based interaction, passive activities like arriving or waiting could no longer generate a response from the facade. The answer to this was to launch a series of exhibitions where mediated participation would inform the community of interactive potential versus the alternative of static notifications such as signage or news bulletins. The intent was threefold: (i) in hosting an exhibition, participants could experience an urban media facade with a larger group of people; (ii) those participants would then be motivated to share it with their friends on non-exhibition nights; and (iii) an exhibition would be an informal platform for users to discuss their experience with the author.

The choice to move away from sensors changed constraints of the interaction that needed to be prototyped. The facade was no longer married to the idea of passive control from real-time data, but now to the active control of participants. This shifted the focus of research away from environmental interface design to finding alternate methods of control introduced in the next section.

5.3.3 Urban Interface Design & Mobile Content

A greater understanding of situational context allows us to bring the discussion more toward urban interface design. A collection of many decisions were made in the design process to satisfy a social experience mediated by ubiquitous mobile devices. In conceptualizing the type and nature interactions in the artistic domain, we were inspired to mediate a familiar activity that would appeal to the creative community at CalArts: drawing. This reasoning was straightforward in that pixel-based manipulations are a nearly universally recognizable unit of visual and artistic expression. Additionally, we felt this approach was prudent since touchscreen-style interaction is widespread and unlikely to incur a learning curve, lessing the cognitive impact of mastering an interface for the installation. Furthermore, the gesture of turning a pixel on or off could be highly evident; nearby onlookers would not have to wonder too hard about the nature of interaction that is occurring. The major appeal of mobile devices was founded in both their growing availability and codified interaction paradigms. A predominant factor of operating systems like Apples iOS and Googles Android is the potential to more easily address

social, multi-user components over more complicated low-level mobile development of other specialized touchscreen devices.

A major aspect of any interactive media facade is to the extent multi-user operation is supported. Researcher Eva Eriksson argues simple multi-user design is not sufficient to build pervasive and engaging urban systems and encourages the design of systems with true social interaction in mind [111]. A challenge in realizing this interaction within GR1D was in building a democratized drawing application where the number of simultaneous users was only limited by the number of mobile devices on hand, a departure from other media facades like TouchProjector which relied on a limited number of dedicated devices to be shared among participants. Realistically, the upper limit of simultaneous users was experimentally determined to be around 10 given GR1Ds limited number of 40 discrete pixels. With this constraint in mind, we embarked on developing a mobile application that satisfied a goal of providing spontaneous and natural social interaction between users [106].

The constraint of a low-density display was instrumental in supporting spontaneous interaction with the mobile application. The associated challenges of multi-user pointing through sensor and augmented reality-based approaches did not seem like a worthwhile approach given the limited resolution. While iterating over a series of sketches, the design that stood out the most was one that mirrored the facade. The vision was that a 1:1 on-off interaction with the GR1Ds pixels could encourage users to be creative in their drawing, potentially taking influence from pixel-based 8-bit 1980s era videogame graphics. This mobile interface was prototyped, and later realized, as a control surface mirroring the facade in real time. So, how does this impact the continuity of interaction? The design principle of the minimalist-inspired interface was that a users focus should be foremost on the facade and surrounding participants, not on interactions with a mobile device [111]. Toward the idea of spontaneity, we were motivated in part by researcher Eva Erikssons argument that serendipity should be a role in the sense that a user should be able to spontaneously join without too much effort [111]. Informal testing with this prototype led us to understand that spontaneity of interaction was highly related to the socially perceived learnability of the application. This was realized by displacing superfluous widgets and reserving descriptive text to a secondary menu, lessening the cognitive impact of the UI in social settings, both in terms of learnability and loading time [124].

Approaching the idea of natural social interaction was markedly harder on account of the interplay between the interface, display, situational complexity, and architectural space. Finke et al categorized user interaction into three distinct groups: bystanders, actors, and spectators, where actors could enact states such as entering, inputting, or leaving [125]. With this in mind, a nautral interaction would be the social influence of actors on converting bystanders and spectators to active collaborators by encouring them to change state. Paay and Kjeldskov noted in a study about urban interactions with mobile devices that one participant termed a common behavior socializing by proximity[126], a tendency of users to gather socially simply by their shared interaction, even if just in an observational state. On a related topic, Boring concluded that their mobile interaction techniques in TouchProjector were easy to learn, but left users blind to the input of simultaneous collaborators [110]. Noting this difficulty of collaborative multi-user synchronicity but acknowledging social proximity as a potential solution, the interface was prototyped with a real-time layer on top of the interactive pixels so users could see what other users are drawing and creating. The reasoning was that a close spatial relationship between users – a quick glance to a neighbors screen, perhaps – combined with a real time overview of the facade would encourage collaboration among socially unacquainted guests. A contributing factor to this was the orientation of devices themselves. Where viewing angle or perspective differed, input could be encouraged from all angles given the live control UI would appear in a fixed orientation.

Visual aesthetics played a large role in prototyping the application. What would be the correct balance between control and artistic whimsicality? In the first version, different modes could be selected via the top row of pixels on the facade. For instance, a combination of selections could restrict the display to a specific color palette. Testing showed that this extra bit of control added an element of discoverability to the interface, but issues of simultaneous control became problematic in larger groups where democratized modal control tended to yield chaotic behavior. In the next iteration, we chose to rotate color palettes and interaction modes automatically. The drawing mode was prototyped so that users could not directly manipulate color, rather each successive on event would cycle through a Hue-Saturation-Value spectrum. This spectrum was chosen over a different color space such as RGB on account of the more harmonious transitions between color groups. Similarly, the fingerpainting mode used the same spectrum, although users did not have discrete on-off control – pixels would fade out over a short amount of time

Material	Form	Combination	Location	Situation	Interaction Sensing	Interaction Style	Format	Content
Concrete, Wire, Arduino, Light, LED	Square, Hole, Line	Matrix, Grid	Main Entrance, Walkway, Ceiling	Waiting, Arriving, Departing, Resting, Performing, Socializing, Greeting, Exhibiting	Active, Collaborative	Touch, Gestural, Sound	Image, Shape, Data	Ornament, Art

TABLE 5.2: Final Design Space Explorer

after being triggered. Fingerpainting was designed to automatically engage when the number of users increased over a specific threshold of users. This alternate mode was introduced to mitigate a problem encountered where collaborative drawing tended to break down with approximately 5 to 6 simultaneous users. Conversely, when no users are presently interacting with the facade, it was decided that the server should launch a screensaver mode that could play predefined animations and color scripts.

5.3.4 Design Summary

GR1Ds design was a continuous cycle of conceptualizing, prototyping, iterating, refining, and finalizing interaction methods that would offer compelling solutions to our perceived problem areas. The mobile application was designed with a central focus on social interaction where the community of CalArts could spontaneously and naturally engage a familiar artistic activity within an urban pixel array. Special thought was given to the multi-user and collaborative aspects where it supported the goal of forming a social dialog about the participants and their environment. Figure X shows the final Design Space Explorer and the various terms that changed throughout the process. While the explorer had no direct bearing on the physical implementation, it acted as a reminder to the conceptual design values being pursued.

FIGURE 5.4: GR1D with a drawn pattern

5.4 Implementation

Thus far, only the conceptual design of GR1D has been described. This section focuses on the specific hardware and software techniques, and implementation strategies that led the the realization of the display and its installation in the waffle-patterned ceiling. We begin by offering a brief overview of the system and architectural space where the display is installed. We follow with a full description of the various hardware components that GR1D relies on to operate. Lastly, the final prototype of the mobile application is presented.

Overview

Figure 5.5 illustrates a not-to-scale depiction of the generalized interconnections between the user, hardware and software. In the fall semester of 2012, CalArts deployed a new building-wide campus WiFi configuration. Any mobile device capable of connecting to a standard 802.11a/b/g WiFi network can access the internet and internal computing resources. With support from the Institutes Information Technolgy Department, grid.calarts.edu domain name was set up to point to an on-site server. The server

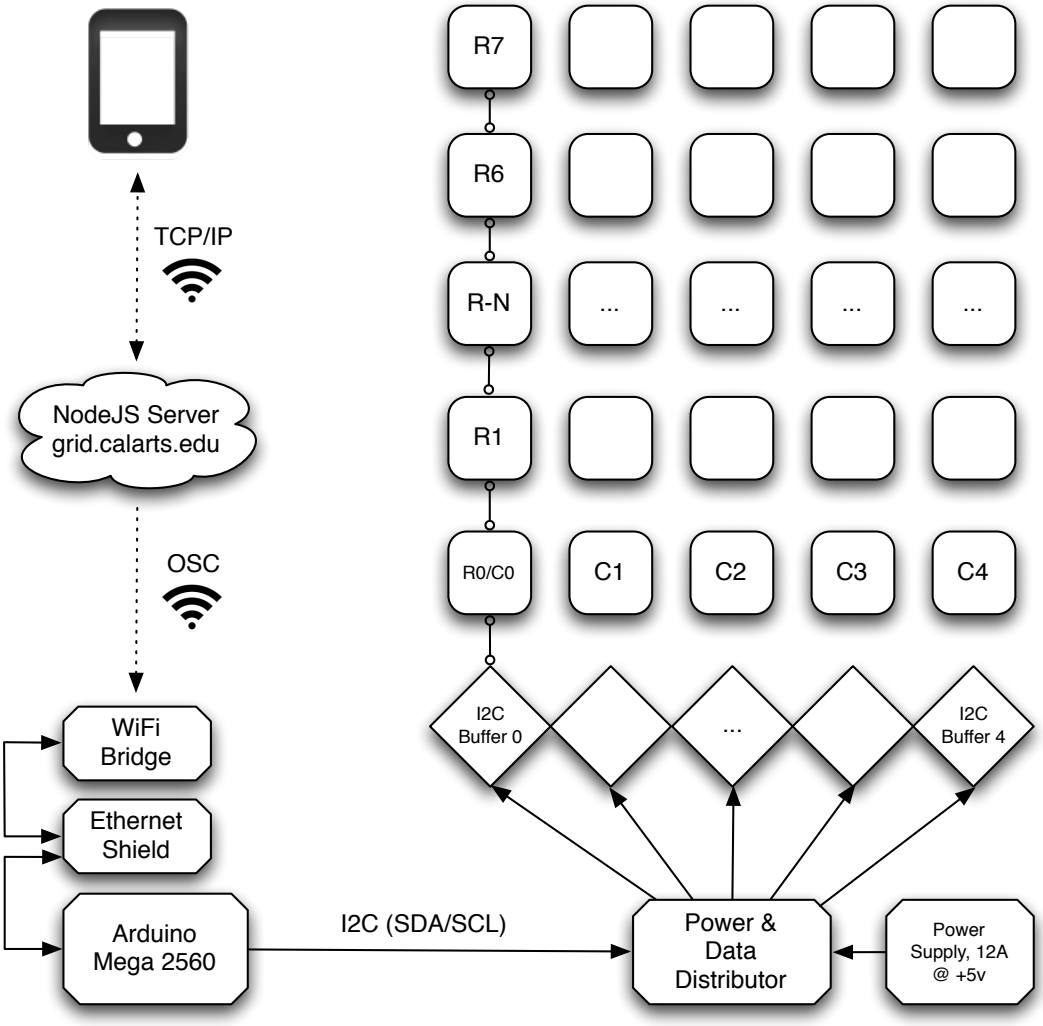

FIGURE 5.5: Overview of the hardware and software components with interconnecting protocols

provided the control interface to mobile clients while maintaining an active connection to the Arduino pixel controller via a WiFi bridge. The bridge was neccessary since no available hardwired network connection was available within 100 feet of the ceiling. From the controller, data and power flows through a custom distributor *shield* (Arduino terminology for a breakout board) and on to one of five custom data-buffer PCBs. The purpose of these PCBs are described in greater detail later. Each column is connected lengthwise via ribbon cable with common data and power for a total of 5 columns with 8 pixels each.

Figure 5.6 depicts the arrangement of ceiling waffles prior to installation. Each concrete waffle is exactly 24 inches deep and 30 inches face to face with a 6 inch gap between

FIGURE 5.6: A detail view of the waffles. Mmm, waffles.

waffles. The area is situated under the main doors to the institute at the end of a long walkway with another 224 waffles not pictured. The 5x8 matrix is bounded by 4 feet gap-areas which were suspected to contain structural reinforcement for the ceiling. This specific area was chosen on account of the well bounded arrangement and proximity to the main entrance doors.

Hardware

The following section describes GR1Ds hardware from a bottom-up approach, starting with the pixel units themselves. One unifying factor throughout the hardware development process was a dual focus on employing off-the-shelf open source hardware and low-cost. Based on early cost assessment, most architectural lighting solutions were far out of the budget to be used for the project. In employing a DIY approach to the electronics and cabling, the entirety of the installation was completed for a modest budget.

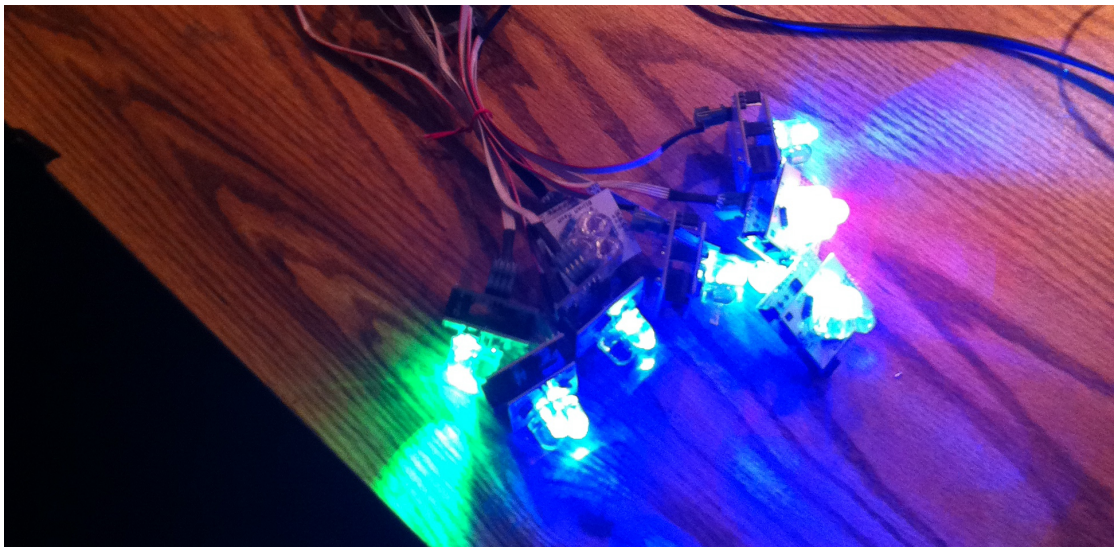

FIGURE 5.7: Eight powered MaxMs prior to installation

Pixels

A popular approach to professional dynamic lighting displays is through a basic serial protocol known as DMX (footnote). This protocol is prevalent in theatrical and architectural lighting design community where the vast majority of equipment conforms to the protocols standard. Unfortunately, most RGB panels with DMX compatibility range in the hundreds to thousands of dollars per unit, although they are often daylight-competitive. Since GR1D did not need to be seen in daylight, the author began exploring the possibility of manufacturing a DMX-compatible RGB LED driver. Based on the size and complexity of the PCB, it was infeasible to manufacture 40 boards for less than $60 each, although higher quantities would yield significantly less per-unit price. Very few open-hardware components are DMX compatible, however, an abundance of general RGB driver boards are currently available. One of these, the BlinkM MaxM by ThingM industries (footnote), fit the pixel requirements well using another protocol known as the Two Wire Interface, abbreviated as TWI but more commonly known as I2C (footnote). The MaxM board has 3 discrete R/G/B LEDs with full 24-bit color control for a maximum brightness of 445,000 millicandelas(footnote). 45 MaxMs are used in GR1D for a cost of $26 per unit (footnote). A bundle of 8 MaxMs pre-installation are shown below in Figure 5.7.

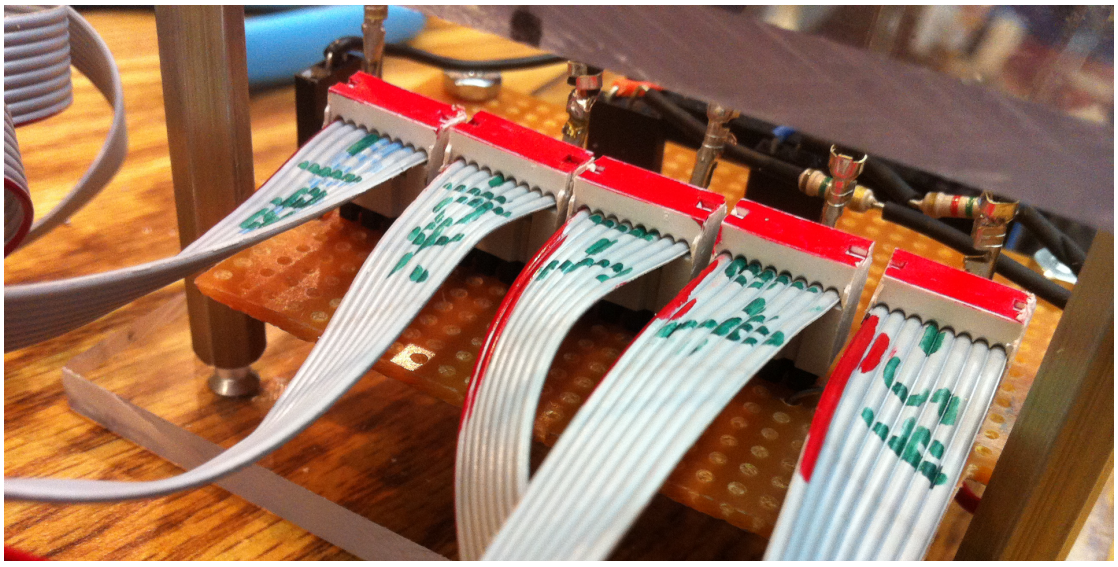

FIGURE 5.8: Edge of the controller board showing the I2C and power distribution mechanism

I2C and Cabling

A benefit of using the I2C protocol is that up to 127 nodes can be attached along a common bus and programmed with individual addresses, eliminating the need for separate wires to each pixel. Only four wires are required for individual nodes: power, ground, data, and clock. Although I2C seems suitable for an application like GR1D, it is not without significant problems. I2C was originally designed for communication between microcontrollers located on the same PCB with a maximum distance of around 2 centimeters. By lowering factors such as I2C clock rate – where the default is 400 kHz – the distance can be extended to around 50 feet. Each column of the display needs 28 feet including the distance from the beginning of the column to the controller. Since column needed to share a common bus, the total length of the I2C bus is about 150 feet, three times longer than the theoretical maximum. Using conventional techniques, reliable communication is not achievable even when the clock rate is lowered to 10 kHz.

I2C is length-limited by the maximum capacitance on the bus. The protocol specifies that the total bus capacitance should not exceed 400 nanofarads (pF) (footnote). A number of semiconductor manufacturers produce bus-buffer chips which overcome this limit by isolating capacitance on one side of the bus. The author designed a breakout board for National Semiconductors P82B96 I2C bi-directional bus-buffer IC which is inserted at the top of every column(footnote). Illustrated below in in Figure 5.9, the

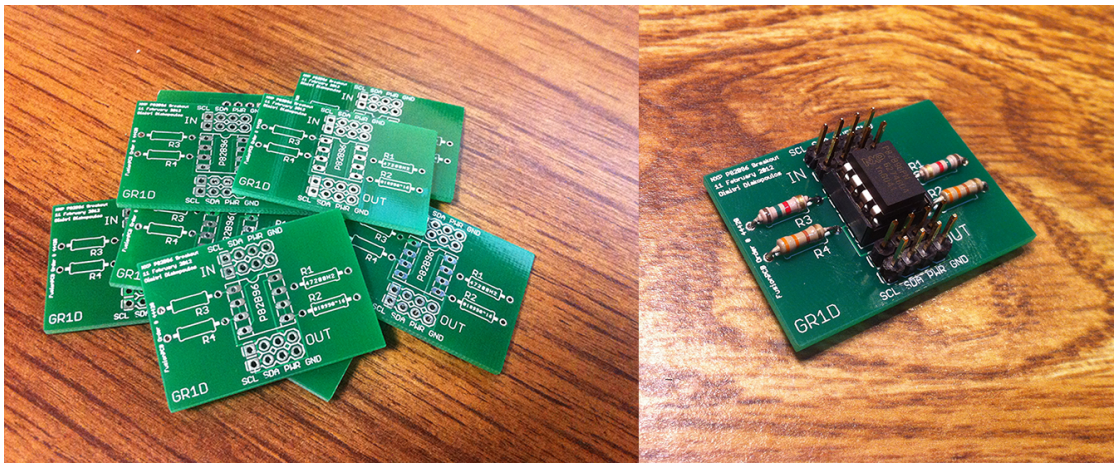

FIGURE 5.9: A number of the I2C PCBs (left); A single board populated with components (right)

role of the board was to host a number of resistors required for the IC in addition to passing through power and ground to the column using a single ribbon cable. The nominal capacitance of ribbon cable is approximately 14 pF per foot, meaning that each column hosts around 392 pF of capacitance. The total bus capacitance is only affected by the unbuffered length of cable between the controller and the buffers, well within the 400 pF tolerance. Two other methods are used conjunction with the bus-buffer to ensure reliable communication. The first method lowers the effective bus speed to 50 kHz, a best practice in long-cable environments. The second is doubling the number of conductors in the ribbon cable. Where only four wires are needed, eight conductor ribbon cable is used such that every other conductor is grounded to prevent crosstalk between the data and power lines. These wires are illustrated above in Figure 5.8.

WiFi and Controller

Communication across the I2C bus is managed by an Arduino Mega 2560 acting as a master node(footnote). Although I2C is a bi-directional protocol, the Arduino is only used for pushing on-off and color messages to the pixels. Power, ground, and data wires are sourced from a custom distributor board located next to the Arduino. This board also contains a transistor connected to ground line of the entire I2C bus and a digital pin on the Arduino, permitting a remote reset capability. Connected on top of the microcontroller is an Ethernet shield(footnote). The use of the Ethernet shield was required in the absence of a reliable WiFi Arduino shield. Control messages are received by the Ethernet shield using the OSC protocol and translated to I2C on the Arduino.

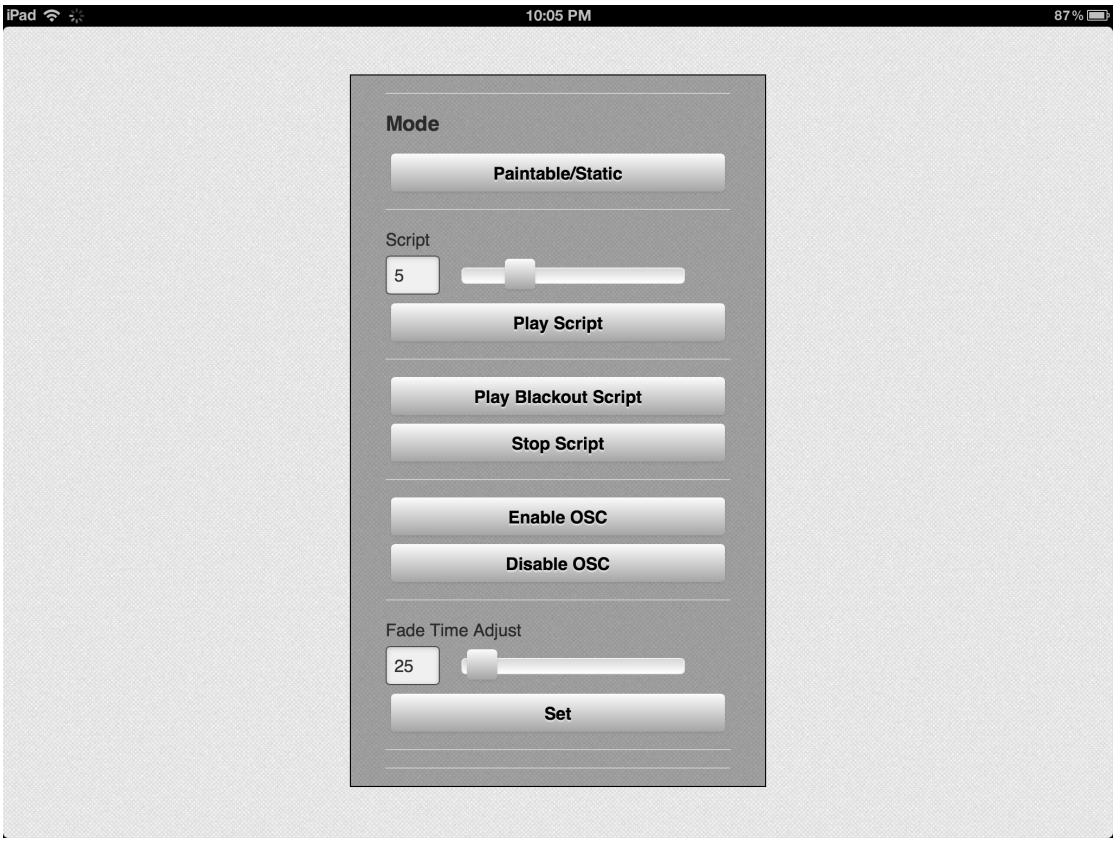

FIGURE 5.10: The administrative screen

Without a hardwired Ethernet cable in the locality of the installation, a separate WiFi to Ethernet bridge is used to connect the Ethernet shield to the campus-wide wireless network. The particular WiFi hardware used is an off-the-shelf D-Link wireless router hacked with an open-source DD-WRT firmware which contains an easily configurable bridging mode (footnote).

Server and Mobile Application

A server located in the Institutes data center hosts the interface and acts as a single source of control for the facade. The back-end is written on a heavily modified Netpixl core, previously discussed in Chapter 5. These modifications were neccessary to provide a comprehensive administrative interface, modal changes based on user activity, and time-based interaction restrictions. The administrative screen shown in Figure 5.10is also designed for mobile use.

The Javascript-based front-end uses a mixture of custom code within the Netpixl framework for synchronous interaction. The application is developed with a number of HTML5

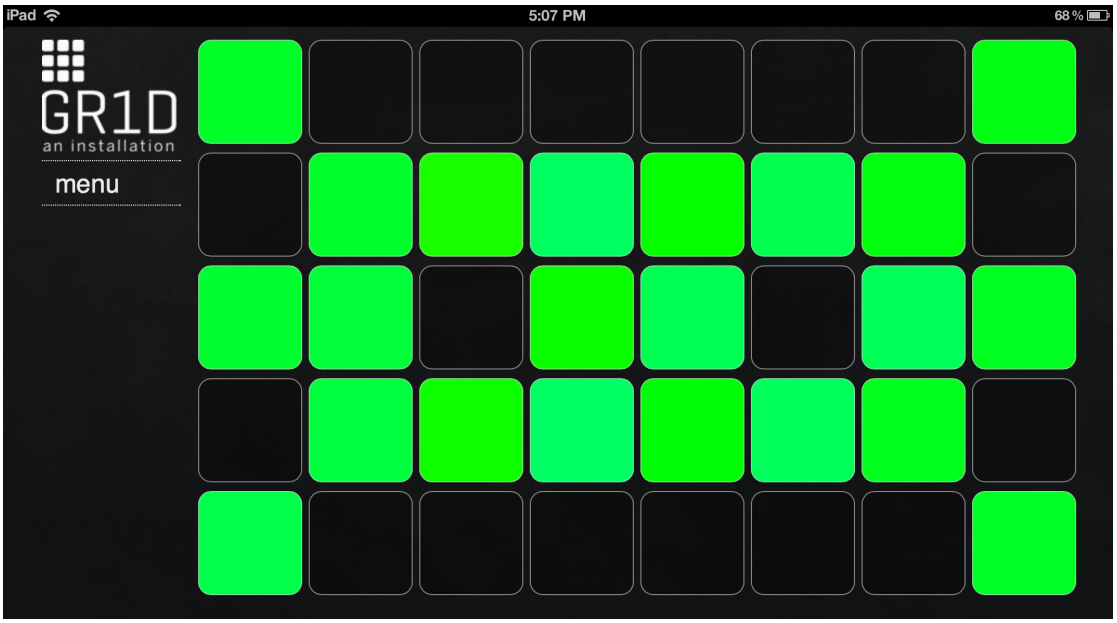

FIGURE 5.11: A space invader drawn on an iPad

and iOS tags which specify that a browser can run the app in a special mode omitting extraneous browser UI elements(footnote). Users may begin using the the app by scanning a QR code from a poster located on the wal or manually typing in grid.calarts.edu. Webapp mode is meant to be engaged by using Android or iOS *add to homescreen* feature, which renders a custom shortcut icon on the a device's homescreen. Using the app in this special mode requires reloading the interface a second time, but permits an easy channel for later use. A screenshot taken on a 3rd generation iPad is shown in Figure 5.11.

One notable implementation feature is the use of CSS3 @media tags for specifying different interface layouts for various screen sizes (footnote). The GR1D application utilises a number of these media tags to scale the interface to the most common mobile device screen resolutions. On account of the facade being in a rectangle, the application is limited to running in a landscape orientation so screen real-estate is maximally utilized. CSS3 media queries are also employed to restrict the use of the interface from users who may be tempted to load the grid.calarts.edu address from a desktop computer. As there is no easily-implementable way of determining *where* the application is accessed from, this restriction is enacted to help ensure that users might be on a mobile device somewhere in the vicinity of the facade.

FIGURE 5.12: The author's girlfriend drawing a heart

The Netpixl server seamlessly handles the addition and removal of users from the application. When a user first loads the interface, the application inflates itself from a data model saved on the server. This model is what drives the state of the display and the state of all users screens, as shown in Figure 5.12 Underneath the interface is also a number of custom analytics set up to track user behavior and engagement through the Google Analytics platform. These statistics, along with the some thoughts about the exhibitions, are briefly discussed in the following section.

5.5 Discussion and Evaluation

Over the course of a month, users were monitored and informally interviewed about their experiences with GR1D during each of the two exhibitions that were held. The installation had a soft non-interactive launch on April 2nd 2012, the day students returned to campus from spring break. A more formal introduction followed as an exhibition held on the evening of April 11th and again on April 19th. Both events were promoted through the Institutes event calendar as well as a Facebook event notification. Staging

FIGURE 5.13: Several users interacting with GR1D (left); a father showing his young son the interface (right)

the first exhibition was designed to familiarize a core group of people with the interface such that they would be motivated to bring their friends on other nights and gain a general fluency with public displays in general.

On the first exhibition night, the attendance over the course of an hour totaled approximately 40 people. The installation was well-received by participants with many positive comments and a number of questions. The direct action/reaction of the display seemed to facilitate fun personal interactions with the facade. Several attendees without compatible devices were eager to borrow one so they could participate. It appeared that a shortage of devices indeed encouraged more social participation as several users would associate with a single users who had loaded the interface, impatiently waiting for a turn. This verbal teamwork hinted at a social interaction that was not originally foreseen; the goal was to overcome a physical constraint by grouping together around a single device. Situations such as this have been noted before as one of Dalsgaards challenges: unforeseen situations are a hallmark of media facades where Dalsgaard stipulates that interaction design needs to adapt or compensate [92].

One of the most common questions asked was related to the implementation: how does it all work? Other questions about the installation related to future work: what other modes could be implemented? On the second exhibition held a week later, a recurring comment often started with, “wouldnt it be cool if.... These types of responses particularly interested the author where several participants commented that they would like to control the display for their own creative uses. Several lengthy discussions were

FIGURE 5.14: A pattern drawn by group of users collaborating to play checkerboard

sparked on this topic were about the role of public displays in a creative environment like CalArts. The general feeling was that it was a novel concept with many future avenues of exploration. Of the 20 people in attendance, nearly all of them had become acquainted with the facade in the preceding week. The nature of discussion revealed that participants still enjoyed drawing-style interaction but with strong undertones that new multi-user modes should be introduced to maintain a novelty factor. Several others commented that they would like to see other types of non-touchscreen interaction.

Several social behaviors were observed both casually and on exhibition nights. Firstly, the proximity effect proved to be a powerful collaboration tool. On many occasions, the compact form factor of the mobile display allowed neighboring participants to more easily see the content and contribute. In once instance, two users took turns loading up pictures of classic 8-bit videogame characters while another would draw it on the facade. Another clear social pattern emerged where users were already interacting with the facade. During the two exhibitions, several students stopped to become an observer among a much larger crowd. Often, this principle of spectacle attracted more and more visitors throughout the evenings: a crowd attracts a larger crowd.

Since the interface was integrated with analytics capture, some basic stats collected

in April 2012 present a more complete picture of user engagement. During this time, many casual observers noted that they felt the media facade was always there, indicating that the integration with the environment and building had been successful. While the author did routine maintenance over the months of April and May, many passersby indicated a desire to see the display permanently integrated into the structure, even if non-interactively.

Statistics

Over the course of April, the facade was engaged by non-administrative users a total of 349 times, representing 141 unique users. April 11th logged 28 unique users while April 19th logged 12 unique users. The present student body in 2012 is 1454 people, indicating that that roughly 10% of students at CalArts entered an input state with the installation for 10 seconds or more. The remaining 101 users were engaged on various evenings between April 11th and April 30th. On average, the length of a unique visit was just shy of 5 minutes at 4:50. The most commonly used devices were the Apple iPhone, followed by iPad, and then a mixture of other Android devices. One interesting statistic is that iPad users engaged with the installation an average of 10:40, nearly twice as long as the average mobile user.

5.6 Summary

Relative to the student body size, the total percentage of participants was relatively low. The author had originally envisioned the engagement number to approach 25% of the student body. What could be improved in future iterations to increase this statistic? While the general feedback on the multi-user and social aspects was positive, using a QR code or manual address entry might be one potential bottleneck for new users. Why engage with facade on your own device when a friend already has it loaded? One solution that might be to implement a *captive portal*. Using a dedicated WiFi access point, it is possible to capture new connections and redirect devices to a specific website address when a browser is loaded. Another aspect that may be considered is the dual-integration of mobile-level and sensor-level control mechanisms. Perhaps a small array of proximity

sensors could activate when a users is neighby, beckoning them toward a prompt to load the interface.

In evaluating the analytics, the iPad figure did pop out as problematic. The length of iPad visits mark a potential inadequacy of the interface when applied to devices smaller than the 10 screen of a 3rd generation iPad. On smaller devices, a pixel-level control mechanism might be too fine-grained for drawing applications where fat-finger mistakes dramatically increase the time to achieve a goal. While a general downside of low-resolution urban displays in general, alternative strategies for gestural interaction need to be further explored that do not rely on the expectation of pixel-level control. With the facade in place, GR1D is an ideal platform to test further designs and generate research in the interaction design space for low-density urban displays.

Another desire expressed by a number of participants was the lack of a multimedia experience. One of the largest schools within CalArts is music, representing about 20% of the student body. While the author himself is affiliated with the school of music, the state of audio in mobile webapps is highly problematic due to OS level constraints on mobile devices. One avenue might be to bundle the application as a native app. This process subjects an application to the approval mechanisms of the Apple Store or Android Martketplace, however, native applications have the benefit of fully accessing platforms full audio API. Another avenue currently being explored is the possibility of using a streaming server which would only require a user to play a single audio file that is actually being live-streamed from a central server synthesising on-the-fly sounds from the interface.

The emergent nature of media facades means that solutions to interaction techniques and technical challenges will only evolve over time with new prototypes and design strategies. With an emphasis on designing a system for synchronous social collaboration using common interaction techniques on mobile devices, GR1D is an ideal case-study in designing an application for a low-density pixel-based media facade. By documenting the design principles through a research through design approach, we have presented a clear path through the complex design process of an urban interface. There is no interaction design pattern language for large scale urban displays, although a major hope is that the work presented here, among the growing number of related projects,

may begin forming a lexicon of techniques which illuminate the creative, artistic, social, and cultural value of collaborative displays situated in urban environments.

Chapter 6

Conclusion

In this thesis, I have presented four instances of applications or tools intended to illustrate thoughtful approaches computational creativity and synchronous collaboration. Both new tools for creative coders and new social applications have great potential to reinvigorate the way artists create in the digital realm. I am convinced of a future where visual and musical expression, as leveraged through social, synchronized, and embodied systems, can motivate artists to create works with Big C potential.

As a result of Netpixl and GR1D, I observed that consumer-level sensing devices can mediate audio and visual interaction in a socially meaningful way. I am further interested in exploring how such sensing systems contribute a physical dimensionality to these social interactions. This issue is particularly pertinent to future research as it directly addresses embodiment: a network should not simply recreate physical collaboration but rather define and enrich a new collaborative experience. Simultaneously, I am motivated to investigate algorithms which help participants learn and create useful sensor mappings to such artistic systems, as quality and consistency of interaction is critical in engendering participatory action.

If you've made it this far, thanks for reading!

Chapter 7

Acknowledgements

This thesis and the projects therein would not have come to fruition without the wise advice, encouragement and support of my mentor, Ajay Kapur. I would also like to thank Jim W. Murphy for his help in conceptualizing GR1D, withstanding relentless LaTeX questions, and other helpful comments and criticism. I thank both Owen Vallis and Jordan Hochenbaum for their valued collaborations to many ideas presented here. My brothers are also deserving of thanks, Chris and Nick, for pushing me in my academic and creative endeavours throughout my undergraduate and graduate tenure; as well as my parents, who always made sure I was doing my homework.

Appendix A

Glossary

Abstraction

In software engineering, abstraction is used as a method of reducing complexity by building interoperable layers which hide implementation details. Abstracted layers are often used by programmers through simplified interfaces. A good example is the Websocket Protocol, built on top of the following abstracted and generalized layers: TCP/IP, OS Socket API, Socket, Networking Hardware.

Application

A software program written for user-facing interactions. Built on top of or with the help of frameworks, toolkits, and libraries.

Architecture

Referring to Software Architecture. Software Architecture is the high-level organization of a software system, defining the layers of abstraction and the code-level interaction of components within.

Asynchronous

In software engineering, asynchronous programming is used to denote a pattern of programming in which external events (database calls, GUI operations, etc.) can be handled as needed without fully blocking the current thread of execution.

Bot

See Electromechanical Instrument.

Black-Box

The black-box model in software engineering is a method of managing complexity by allowing developers to extend functionality by re-defining select functions through the use of a hook system. In contrast, the white-box model normally requires developers understand the complete framework, toolkit, or library in order to derive new reusable components.

Electromechanical Instrument

This term is used to describe the acoustic and electrically amplified instruments used in the Machine Orchestra. They are a mixture of traditional and custom instruments which are actuated by motors, solenoids, and servos. This term is used interchangeably with Robot and Bot, although the instruments have no intelligence of their own.

Framework

A software framework can used to build applications by providing a reusable set of extensions that can be used to design overall control and flow. Often, frameworks exist to speed the development process up by providing useful abstractions on top of OS-level APIs, such as graphics, audio, and threading.

Groupware

Groupware refers to genre of software applications that engage and support multiple-users working toward a shared goal.

Media Facade

A term used to denote the output of an emerging discipline mixing HCI, CSCW, environmental computing, context-awareness, audience participation, and architecture. Media Facades are architectural-level installations that make use of large-scale display devices such as LEDs or high-resolution, high-output projectors.

Synchronous

As used in this thesis, synchronous refers to the perceptual real-time state of communication that can be achieved in modern networked systems.

Synchronicity

The state of being synchronous. See Synchronous.

Synchrony

See Synchronous.

Library

A library is a collection of code-level components that can be used to build applications, toolkits, and frameworks. Libraries often encapsulate related algorithms. Often, operating systems permit applications access to low-level features by exposing library functions. Developers often create libraries as a way of encapsulating useful abstractions that may not be as expansive as a toolkit or framework in scope.

Models

Models encapsulate the abstractions in a software system.

Real-time

See Synchronous.

Robot

See Electromechanical Instrument.

Toolkit

Toolkits are collections of reusable components that can be used to help build applications. Toolkits can be built on top of libraries and frameworks and are typically specific to one area such as widgets, networking, graphics, or audio.

Ubiquitous Computing

A term coined by Mark Weiser in 1991 to describe the impending pervasiveness of computers in our environment.

Bibliography

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Creativity: Flow and the Psychology of Discovery and Invention.* New York: Harper Perennial, 1996.

[2] Paul Nemirovsky. *Improvisational interaction: a framework for structural exploration of media.* PhD thesis, Massachusetts Institute of Technology. Dept. of Architecture. Program In Media Arts and Sciences, 2006.

[3] Julian Oliver, Gordan Savičić, and Danja Vasiliev. The Critical Engineering Manifesto, 2011. URL `http://criticalengineering.org/`.

[4] David Mellis and Leah Buechley. Scaffolding Creativity with Open-Source Hardware. In *Creativity and Cognition*, 2011.

[5] Dalia El-Shimy, Thomas Hermann, and Jeremy R Cooperstock. A Reactive Environment for Dynamic Volume Control. In *New Interfaces for Musical Expression*, pages 273–383, 2012.

[6] Stacey Kuznetsov and Eric Paulos. Rise of the expert amateur: DIY projects, communities, and cultures. In *Nordic Conference on Human-Computer Interaction: Extending Boundaries*, pages 295–304, 2010.

[7] Bill Verplank, Craig Sapp, and Max Mathews. A Course on Controllers. In *New Interfaces for Musical Expression*, pages 1–4, 2001.

[8] Gideon D'Arcangelo. Creating a context for musical innovation: a NIME curriculum. *New Interfaces for Musical Expression*, pages 1–4, 2002.

[9] Marcelo Mortensen Wanderley and Nicola Orio. Evaluation of Input Devices for Musical Expression: Borrowing Tools from HCI. *Computer Music Journal*, 26(3): 62–76, 2002.

[10] Perry Cook. Principles for Designing Computer Music Controllers. *New Interfaces for Musical Expression*, 18(7):1–4, 2001.

[11] Perry R Cook. Re-Designing Principles for Computer Music Controllers: A Case Study of SqueezeVox Maggie. In Noel Zahler, Roger B Dannenberg, and Tom Sullivan, editors, *New Interfaces for Musical Expression*, volume 11, pages 218–221. Carnegie Mellon University, 2009.

[12] S Wilson, M Gurevich, B Verplank, and P Stang. Microcontrollers in Music HCI Instruction. *New Interfaces for Musical Expression*, 2002.

[13] Alicia M Gibb. *New Media Art, Design, and the Arduino Microcontroller: A Malleable Tool*. PhD thesis, Pratt Institute, 2010.

[14] Hans-christoph Steiner. Firmata: Towards making microcontrollers act like extensions of the computer. *New Interfaces for Musical Expression*, pages 125–130, 2009.

[15] Dan Overholt. Musical Interaction Design with the CREATE USB Interface Teaching HCI with CUIs instead of GUIs. *International Computer Music Conference*, 2006.

[16] Andy Schmeder and Adrian Freed. uOSC : The Open Sound Control Reference Platform for Embedded Devices. *New Interfaces for Musical Expression*, pages 175–180, 2008.

[17] Andrew Schmeder and Adrian Freed. A Low-level Embedded Service Architecture for Rapid DIY Design of Real-time Musical Instruments. In *New Interfaces for Musical Expression*, pages 121–124, 2009.

[18] Owen Vallis, Jordan Hochenbaum, and Ajay Kapur. A Shift Towards Iterative and Open-Source Design for Musical Interfaces. *New Interfaces for Musical Expression*, pages 1–6, 2010.

[19] John Bischoff, Rich Gold, and Jim Horton. Music for an Interactive Network of Microcomputers. *Computer Music Journal*, 2(3):24–29, 1978.

[20] Scot Gresham-Lancaster. The Aesthetics and History of the Hub.pdf. *Leonardo Music Journal*, 8(1):39–44, 1998.

[21] Scott Smallwood, Dan Trueman, Perry R Cook, and Ge Wang. Composing for Laptop Orchestra. *Computer Music Journal*, 31(1):9–25, 2008.

[22] Gil Weinberg. Interconnected Musical Networks: Toward a Theoretical Framework. *Computer Music Journal*, 29(2):23–39, 2005.

[23] Alvaro Barbosa. Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation. *Leonardo Music Journal*, 13:53–59, 2003.

[24] Brian Kane. Aesthetic Problems of Net Music. In *Proceedings of the 2007 Spark Festival*, 2007.

[25] Gil Weinberg. The Aesthetics, History, and Future Challenges of Interconnected Music. *International Computer Music Conference*, 2002.

[26] Ajay Kapur, Michael Darling, Dimitri Diakopoulos, Jim W Murphy, Jordan Hochenbaum, Owen Vallis, and Curtis Bahn. The Machine Orchestra: An Ensemble of Human Laptop Performers and Robotic Musical Instruments. *Computer Music Journal*, 35(4):49–63, 2011.

[27] Tina Blaine and Sidney Fels. Contexts of Collaborative Musical Experiences. *New Interfaces for Musical Expression*, 2003.

[28] Andrew R Brown and Kelvin Grove. Network Jamming : Distributed Performance using Generative Music. In *New Interfaces for Musical Expression*, pages 15–18, 2010.

[29] Juan-Pablo Cáceres, Robert Hamilton, Deepak Iyer, Chris Chafe, and Ge Wang. To the Edge with China: Explorations in Network Performance. In *International Conference on Digital Arts*, pages 7–8, 2008.

[30] Juan-Pablo Cáceres and Chris Chafe. JackTrip: Under the Hood of an Engine for Network Audio. *Journal of New Music Research*, 39(3):183–187, September 2010.

[31] JO Borchers. A Pattern Approach to Interaction Design. *Cognition, Communication and Interaction*, 2008.

[32] Ge Wang and Perry R Cook. ChucK: A Concurrent, On-the-fly, Audio Programming Language. In *International Computer Music Conference*, pages 1–8. San Francisco: ICMA, 2003.

[33] Mark Cerqueira. *Synchronization over Networks for Live Laptop Music Performance*. PhD thesis, Princeton University, 2010.

[34] Chris Chafe and Michael Gurevich. Network Time Delay and Ensemble Accuracy. In *Audio Engineering Society (AES)*, 2004.

[35] Matthew Wright. Open Sound Control: an enabling technology for musical networking. *Organised Sound*, 10(03):193–200, November 2005.

[36] Peter F Driessen, Thomas E Darcie, and Bipin Pillay. The Effects of Network Delay on Tempo in Musical Performance. *Computer Music Journal*, 35(1):76–89, 2011.

[37] Dimitri Diakopoulos and Ajay Kapur. HIDUINO: A firmware for building driverless USB-MIDI devices using the Arduino microcontroller. In *New Interfaces for Musical Expression*, 2011.

[38] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*, volume 206 of *Addison-Wesley Professional Computing Series*. Addison-Wesley, 1995.

[39] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3): 94–95, 98–102, 104, 1991.

[40] Saul Greenberg. Toolkits and interface creativity. *Multimedia Tools and Applications*, 32(2):139–159, November 2006.

[41] Jieun Oh and Ge Wang. Audience-Participation Techniques Based on Social Mobile Computing. In *International Computer Music Conference*, pages 665–672, 2011.

[42] Peter Tandler. *Synchronous collaboration in ubiquitous computing environments*. PhD thesis, Darmstadt Technical University, 2004.

[43] D Robson. Play!:sound toys for the non musical. In *New Interfaces for Musical Expression*, 2001.

[44] Brad Meyers and et al. Flexi-modal and multi-machine user interfaces. In *Human-Computer Interaction Institute*, 2002.

[45] Dimitri Diakopoulos and Ajay Kapur. Argos: An opensource application for building multi-touch musical interfaces. In *International Computer Music Conference*, 2010.

[46] Thomas E Hansen, Juan Pablo Hourcade, Mathieu Virbel, Sharath Patali, and Tiago Serra. PyMT: a post-WIMP multi-touch user interface toolkit. In *Interactive Tabletops and Surfaces*, ITS '09, pages 17–24. ACM, 2009.

[47] Laufs Uwe, Chris Ruff, and Jan Zibuschka. Mt4j-a cross-platform multi-touch development framework. In *Engineering Patterns for Multi-Touch Interfaces*, 2010.

[48] Alain Crevoisier and Greg Kellum. Transforming ordinary surfaces into multi-touch controllers. In *New Interfaces for Musical Epxression*, 2008.

[49] Dietrich Kammer and Georg Freitag. Taxonomy and overview of multi-touch frameworks: Architecture, scope and features. *Engineering Patterns for Multi-Touch Interfaces*, 2010.

[50] Ge Wang, Ananya Misra, and Perry R Cook. Building Collaborative Graphical interFaces in the Audicle. In Norbert Schnell and Frédéric Bevilacqua, editors, *New Interfaces for Musical Expression*, NIME '06, pages 49–52. IRCAM — Centre Pompidou, 2006.

[51] C Müller-Tomfelde and M Fjeld. Introduction: A Short History of Tabletop Research, Technologies, and Products. *Tabletops-Horizontal Interactive Displays*, 2010.

[52] Sergi Jordà, Martin Kaltenbrunner, Günter Geiger, and Marcos Alonso. The reactable: Exploring the synergy between live music performance and tabletop tangible interfaces, 2007.

[53] James Patten, Ben Recht, and Hiroshi Ishii. Audiopad: a tag-based interface for musical performance. In *New Interfaces for Musical Expression*, pages 1–6, 2002.

[54] Philip L Davidson and Jefferson Y Han. Synthesis and control on large scale multi-touch sensing displays. In Norbert Schnell and Frédéric Bevilacqua, editors, *New Interfaces for Musical Expression*, pages 216–219. IRCAM — Centre Pompidou, 2006.

[55] Jordan Hochenbaum and Owen Vallis. Bricktable : A Musical Tangible Multi-Touch Interface. *Berlin Open*, 2009.

[56] Jordan Hochenbaum, Owen Vallis, Dimitri Diakopoulos, Jim Murphy, and Ajay Kapur. Designing Expressive Musical Interfaces for Tabletop Surfaces. In Kirsty Beilharz, Bert Bongers, Andrew Johnston, and Sam Ferguson, editors, *New Interfaces for Musical Expression*, pages 315–318, 2010.

[57] Robin Laney. Issues and Techniques for Collaborative Music Making on Multi-Touch Surfaces. In *Sound and Music Computing*, 2010.

[58] A Xambo, Robin Laney, Chris Dobbyn, and S Jordà. Multi-touch interaction principles for collaborative real-time music activities: towards a pattern language. In *International Computer Music Conference*, 2011.

[59] Koray Tahiroğlu, Atau Tanaka, Adam Parkinson, and Steve Gibson. Designing musical interactions for mobile systems. In *Proceedings of the Designing Interactive Systems Conference*, 2012.

[60] Rafael Ballagas, J Borchers, and Michael Rohs. The smart phone: a ubiquitous input device. *IEEE Pervasive Computing*, 5(1):70–77, 2006.

[61] Jieun Oh and et al. Evolving the mobile phone orchestra. In *New Interfaces for Musical Expression*, 2010.

[62] Ge Wang, Essl Georg, and H. Pentinnen. *The Mobile Phone Orchestra*. Oxford University Press, 2010.

[63] NJ Bryan and Jorge Herrera. MoMu: A mobile music toolkit. In *New Interfaces for Musical Expression*, pages 174–177, 2010.

[64] Essl Georg. Urmusan environment for mobile instrument design and performance. In *International Computer Music Conference*, 2010.

[65] Michael Nebeling and M Norrie. jQMultiTouch: lightweight toolkit and development framework for multi-touch/multi-device web interfaces. *ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 61–70, 2012.

[66] James Lin, San Jose, and James A Landay. Employing Patterns and Layers for Early-Stage Design and Prototyping of Cross-Device User Interfaces. In *Human Factors in Computing Systems*, pages 1313–1322, 2008.

[67] Charles Roberts, Graham Wakefield, and Matthew Wright. Mobile Controls On-The-Fly : An Abstraction for Distributed NIMEs. In *New Interfaces for Musical Expression*, 2012.

[68] Robyn Taylor, Pierre Boulanger, Patrick Olivier, and Jayne Wallace. Exploring participatory performance to inform the design of collaborative public interfaces. In *Human Factors in Computing Systems*, page 3721, New York, New York, USA, 2009. ACM Press.

[69] Nathan Weitzner, Mcmillan St, Atlanta Ga, Jason Freeman, Stephen Garrett, Yanling Chen, and Georgia Tech. massMobile an Audience Participation Framework. In *New Interfaces for Musical Expression*, 2012.

[70] M. Roseman and S Greenberg. Building real-time groupware with groupkit, a groupware toolkit. In *ToCHI*, 1996.

[71] D. Buszko, W. Lee, and A. Helal. Decentralized adhoc groupware api and framework for mobile collaboration. In *CAM SIGGROUP Conference on Supporting Group Work*, 2001.

[72] C Wolfe and et al. User-centered development of adaptive groupware systems. In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2009.

[73] Richard Bentley, Thilo Horstmann, and Jonathan Trevor. The world wide web as enabling technology for cscw: The case of bscw. In *Computer Supported Cooperative Work*, 1997.

[74] Carl Gutwin, Michael Lippold, and TC Graham. Real-time groupware in the browser: testing the performance of web-based networking. In *Computer Supported Cooperative Work*, pages 167–176, 2011.

[75] S. Morgan and W. Wang. The impact of web 2.0 developments on real-time groupware. In *IEEE Conference on Social Computing*, 2010.

[76] Matthias Heinrich and Martin Gaedke. Websoda: a tailored data binding framework for web programmers leveraging the websocket protocol and html5 microdata. In *Web Engineering*, 2011.

[77] Brian Mayton, Nicholas Joliat, and Joseph A Paradiso. Patchwerk : Multi-User Network Control of a Massive Modular Synthesizer. In *New Interfaces for Musical Expression*, 2012.

[78] Christoph Endres. A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing. *Mobile Information Systems*, 1(1):41–80, 2005.

[79] Ben Shneiderman. Creativity support tools: A grand challenge for hci researchers. In *Engineering the User Interface*, 2009.

[80] Claudia Iacob. Mining for Patterns in the Design of Systems for Synchronous Collaboration. In *Asian Pattern Languages of Programs*, 2011.

[81] Jeff Dyck, Carl Gutwin, T. C. Nicholas Graham, and David Pinelle. Beyond the lan: techniques from network games for improving groupware performance. In *Proceedings of the 2007 international ACM conference on Supporting group work*, 2007.

[82] Curtis Mckinney and Chad Mckinney. OSCTHULHU: Applying video-game state-based synchronization to network computer music. *New Interfaces for Musical Expression*, 2012.

[83] Ian Hattwick and Marcelo M Wanderley. A Dimension Space for Evaluating Collaborative Musical Performance Systems. In *New Interfaces for Musical Expression*, 2012.

[84] Peter Tandler. Software infrastructure for ubiquitous computing environments: Supporting synchronous collaboration with heterogeneous devices. In *International Conference on Ubiquitous Computing*, pages 96–115, 2001.

[85] Gerhard Steinebach, Subhrajit Guhathakurta, and Hans Hagen. *Visualizing Sustainable Planning*. Springer, 2009.

[86] Angela Chang, James Gouldstone, Jamie Zigelbaum, and Hiroshi Ishii. Simplicity in interaction design. In *Tangible and Embedded Interaction*, page 135, New York, New York, USA, 2007.

[87] Donald A. Norman. *The Design of Everyday Things*. Basic Books, Inc., New York, NY, USA, 2002.

[88] M. Resnick, B. Myers, K. Nakakoji, B. Shneiderman, R. Pausch, T. Selker, and M. Eisenberg. Design principles for tools to support creative thinking. In *Report of Workshop on Creativity Support Tools*, 2005.

[89] S Tilkov and S Vinoski. Node.js: Using javascript to build high-performance network programs., 2010.

[90] Paris Avgeriou and Peter Tandler. Architectural patterns for collaborative applications. *International Journal of Computer Applications in Technology*, 25(2/3): 86, 2006.

[91] Matthias Hank Haeusler. *Media Facades*. Avedition Gmbh, 2009.

[92] Peter Dalsgaard and Kim Halskov. Designing Urban Media Façades: Cases and Challenges. In *Human Factors in Computing Systems*, pages 2277–2286, 2010.

[93] Peter Peltonen, Esko Kurvinen, Antti Salovaara, Giulio Jacucci, Tommi Ilmonen, John Evans, Antti Oulasvirta, and Petri Saarikko. "It's Mine! Don't Touch!": Interactions at a Large Multi-Touch Display in a City Centre. In *Human Factors in Computing Systems*, pages 1285–1294, 2008.

[94] Rafael Ballagas, Michael Rohs, Jennifer G Sheridan, and Jan Borchers. BYOD: Bring Your Own Device. In *Ubiquitous Display Environments at Ubicomp*. Citeseer, 2004.

[95] Marcelo Coelho, Jamie Zigelbaum, Zigelbaum Coelho, and Joshua Kopin. Six-Forty by Four-Eighty: The Post-Industrial Design of Computational Materials. In *International Conference on Tangible, Embedded, and Embodied Interaction*, pages 253–256, 2011.

[96] Jamer Hunt. A Manifesto for Postindustrial Design. *ID magazine*, 2005.

[97] Jonathan Grudin. Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, pages 92–105, 1994.

[98] Peter Dalsgaard, Kim Halskov, and Rune Nielsen. Towards a design space explorer for media facades. In *Australasian Conference on Computer-Human Interaction (OZCHI)*, page 219, 2008.

[99] Marcus Foth. *Handbook of Research on Urban Informatics: The Practice and Promise of the Real-Time City.* IGI Publishing, 2008.

[100] Douglas Schuler and Aki Namioka, editors. *Participatory Design: Principles and Practices.* CRC / Lawrence Erlbaum Associates, 1993.

[101] Malcolm McCullough. *Digital Ground: Architecture, Pervasive Computing, and Environmental Knowing.* MIT Press, 2004.

[102] Adam Greenfield and Mark Shepard. *Urban Computing and its Discontents.* The Architectual League of New York, 2007.

[103] Mirjam Struppek. The Social Potential of Urban Screens. *Visual Communication*, 5(2):173–188, 2006.

[104] Odilo Schoch. My Building is my Display. In *Education and research in Computer Aided Architectural Design*, pages 610–616, 2006.

[105] Laurent Migonneau and Christa Sommerer. Media Facades as Architectural Interfaces. In *The Art and Science of Interface and Interaction Design*, pages 93–104. Springer Berlin, 2008.

[106] José Pablo Zagal, Miguel Nussbaum, and Ricardo Rosas. A Model to Support the Design of Multiplayer Games. *Presence: Teleoperators and Virtual Environments*, 9(5):448–462, 2000.

[107] F Borries, S Walz, and M Bottger, editors. *Space Time Play: Computer Games, Architecture and Urbanism: The Next Level.* Birkhäuser Architecture, 2007.

[108] Jan Seeburger and J Choi. Remixing Social Media for Location Sharing on Public Urban Screens. In *Digital Cities 7 - Real World Experiences*, 2011.

[109] Timo Ojala and Jurgen Scheible. MobiSpray: Mobile phone as Virtual Spray can for painting BiG anytime anywhere on anything. In *SIGGRAPH*, volume 42, pages 332–341, 2009.

[110] Sebastian Boring, Dominikus Baur, Andreas Butz, Sean Gustafson, and Patrick Baudisch. Touch Projector: Mobile Interaction through Video. In *Human Factors in Computing Systems*, pages 2287–2296, 2010.

[111] Eva Eriksson, Thomas Riisgaard Hansen, and Andreas Lykke-olesen. Reclaiming Public Space: Designing for Public Interaction with Private Devices. In *International Conference on Tangible, Embedded, and Embodied Interaction*, pages 31–38, 2007.

[112] Sebastian Boring, Sven Gehring, Alexander Wiethoff, Magdalena Blöckner, Johannes Schöning, and Andreas Butz. Multi-User Interaction on Media Facades through Live Video on Mobile Devices. In *Human Factors in Computing Systems*, pages 2721–2724, 2011.

[113] Blinkenlights Project. Blinkenlights, 2001. URL `http://www.blinkenlights.de`.

[114] Jun Rekimoto, Brygg Ullmer, and Haruo Oba. DataTiles: a modular platform for mixed physical and graphical interactions. In *Human Factors in Computing Systems*, CHI '01, pages 269–276. ACM, 2001.

[115] David Merrill, Jeevan Kalanithi, and Pattie Maes. Siftables: towards sensor network user interfaces. In *International Conference on Tangible, Embedded, and Embodied Interaction*, volume pp of *TEI '07*, pages 75–78. ACM, 2007.

[116] Pol Pla and Pattie Maes. Display Blocks: Cubic Displays for Multi-Perspective Visualization. In *Human Factors in Computing Systems - Extended Abstracts*, pages 2015–2020, 2012.

[117] Kelly B Heaton, Robert D Poor, and Andrew J Wheeler. Nami. In *SIGGRAPH*, Computer Graphics, page 215, 1999.

[118] Susanne Seitinger, Daniel S Perry, and William J Mitchell. Urban Pixels: Painting the City with Light. In *Human Factors in Computing Systems*, pages 839–848, 2009.

[119] Nina Valkanova. Interface design for shared spaces. In *ACM SIGCHI Conference on Designing Interactive Systems - Extended Abstract*, pages 29–30, 2010.

[120] Lucy Bullivant. *Responsive Environments Architecture, Art and Design*. V&A Contemporary, London, UK, 2006.

[121] John Zimmerman, Jodi Forlizzi, and Shelley Evenson. Research through design as a method for interaction design research in HCI. In *Human Factors in Computing Systems*, pages 493–502, 2007.

[122] Martin Brynskov, Peter Dalsgaard, Tobias Ebsen, and Jonas Fritsch. Staging Urban Interactions with Media Façades. In *International Conference on Human-Computer Interaction (INTERACT)*, pages 154–167, 2009.

[123] Alexander Wiethoff and Sven Gehring. Interacting with Light. *International Joint Conference on Ambient Intelligence*, 2011.

[124] Antti Oulasvirta, Sakari Tamminen, Virpi Roto, and Jaana Kuorelahti. Interaction in 4-second bursts: the fragmented nature of attentional resources in mobile HCI. In *Human Factors in Computing Systems*, volume 05, pages 919–928. ACM, 2005.

[125] Matthias Finke, Anthony Tang, Rock Leung, and Michael Blackstock. Lessons Learned: Game Design for Large Public Displays. In *International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA)*, pages 26–33, 2008.

[126] J. Paay and J. Kjeldskov. Understanding Situated Social Interactions: A Case Study of Public Places in the City. *Computer Supported Cooperative Work*, 17 (2-3):275–290, 2008.