SWARMED: Captive Portals, Mobile Devices, and Audience Participation in Multi-User Music Performance

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
abram.hindle@ualberta.ca

ABSTRACT

Audience participation in computer music has long been limited by resources such as sensor technology or the material goods necessary to share such an instrument. A recent paradigm is to take advantage of the incredible popularity of the smart-phone, a pocket sized computer, and other mobile devices, to provide the audience an interface into a computer music instrument. In this paper we discuss a method of sharing a computer music instrument's interface with an audience to allow them to interact via their smartphones. We propose a method that is relatively cross-platform and device-agnostic, yet still allows for a rich user-interactive experience. By emulating a captive-portal or hotspot we reduce the adoptability issues and configuration problems facing performers and their audience. We share our experiences with this system, as well as an implementation of the system itself.

Keywords

Wifi, Smartphone, Audience Interaction, Adoption, Captive Portal, Multi-User, Hotspot

1. INTRODUCTION

Audience interaction has classically been limited by physical and monetary resource constraints. If one wanted to share an instrument with an audience and allow them to interact often an entire installation fitted with sensors was needed, or the audience needed to provide their own materials. Smartphones have gained popularity in computer music performance, with smart phone orchestras and groups being formed [10]. Often these groups share the same apps and play together either through their speakers, or over MIDI, or via a mixer. One barrier to audience interaction in the case of smartphones was coding something that was cross-platform enough to run on the major vendors smartphones [17]. In the case of the iPhone, one would even need a developer license and their application would have to appear in the app store. Yet aiming for the smartphone makes sense, each audience member essentially is bringing their own computer and running your software. Yet the least restrictive and most cross-platform method of utilizing smartphones is not apps, it is their ability to view and interact with webpages [1].

In this paper we present and propose a method of allowing anyone with a web-enabled mobile device to participate in a performance by providing an interface to a shared computer music instrument over a wireless network (WiFi), or an Internet connection, via an interactive webpage. We direct users directly to the relevant pages by taking advantage of captive-wifi portal/hotspot technology. Thus the issue of distributing and downloading an app is avoided, as the web-interface of the instrument is distributed locally via WiFi or globally via the internet.

Providing internet access to 20 or more devices can often be prohibitive due to bandwidth constraints and latency issues, thus in this paper we focus mostly on providing instrument interfaces via a local wifi-network configured much like a captive-wifi portal in a hotel. Hotel wifi networks are often captive-wifi portals that require authentication or payment in order to enable internet access, these captive portals intercept and replace your *hypertexttransportprotocol* (HTTP) requests with an authentication and payment interface. Instead of asking you to authenticate, one can use a captive-portal to drive the smart-phones to the computer music instrument's interface.

For the rest of the paper we will describe and discuss *SWARMED: a Shared Wifi Audience Reactive Musical Extemporization Design.* SWARMED is a captive-wifi-portal that serves up web-based user interfaces which allow users to interact with and control a running computer music instrument with their web-enabled devices such as smart-phones, tablets and laptops. SWARMED solves many tool adoption and portability issues by relying on standardized and portable web technologies to deliver interactive instruments.

Thus our contributions in this paper are:

- We propose using captive-portal to drive WiFi users an instrument's *user interface* (UI);
- We describe the SWARMED system abstractly and concretely;
- We provide a free open-source implementation of the SWARMED system [5];
- We discuss our experience performing with SWARMED and suggest future uses for such a system;

2. PREVIOUS WORK

In this section we discuss some of the previous work relevant to audience collaboration, adoptability, and the use of portable digital devices in audience-interactive computer music performances.

In terms of experiments and discussions regarding audience participation using smartphones and mobile devices Oh et al.[11] in "Audience-Participation Techniques Based on Social Mobile Computing" argue that smartphone oriented performances are convenient for both the audience

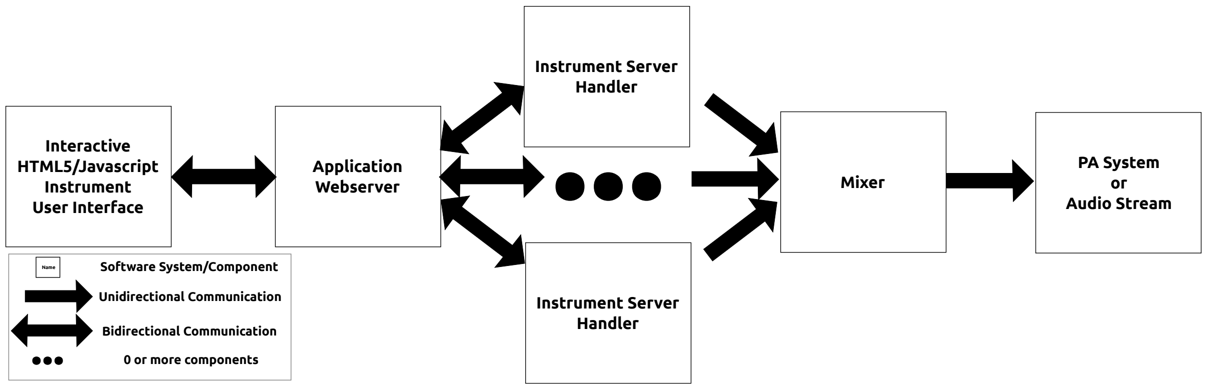
Interactive HTML5/Javascript Instrument User Interface
Application Webserver
Instrument Server Handler
Instrument Server Handler
Mixer
PA System or Audio Stream
Name
Software System/Component
Unidirectional Communication
Bidirectional Communication
0 or more components

Figure 1: Abstract Software Architecture and Communication of Software Components of the SWARMED Network

and performer. Furthermore they discuss different kinds of social interaction ranging from social network interaction (Tweetdreams [3]), through to interactive web applications. They highlight that audience interaction does not have to be musical in nature, it can be about communicating back to the performers or researchers.

Unsilent Night is an example of audience based participation and adoption. Unsilent Night, by Phil Kline [9], invited audience members to join in a performance of playing a limited set of audio samples in an open space with their own boom-boxes and portable speakers. Kline also provided an iPhone app one can install in order to participate.

TweetDreams, by Dahl et al. [3], is an excellent example of audiences interacting with a music instrument via a web interface (Twitter). The audience participates by tweeting at the performer's Twitter account. Their architecture is very similar to SWARMED except that they rely on Twitter to mediate the messages. TweetDreams requires an internet connection and the use of Twitter.

From an infrastructural point of view, Jesse Allison [1] describes an implementation of a distributed performance system that is web-based (HTML5) and uses an application webserver (Ruby on Rails). Allison argues that adhering to HTML5 standards allows for client-side instrument UIs to be distributed without having to address excessive portability issues.

massMobile by Weitzner et al. [17] describes a system meant to allow smartphones to interact with and Max/MSP via web standards (HTML5), over a variety of wireless networks. Their configuration includes a persistent internet accessible sound server running Max/MSP. Our work differs because we leveraged a captive-portal technique and do not necessarily rely on Max/MSP (although we tended to use CSound [15] on the backend).

Tanaka [13] carefully evaluates the capabilities of mobile devices and their affordances for mobile music performance and interaction. Tanaka's work seeks to address the limitations and affordances of the mobile devices and argues that developers of mobile instruments should pay attention to how users use their devices. For instance many users hold their phone with one hand and manipulate it with the other, thus two-handed interaction is not an affordance of mobile phones.

Jordà [7] discusses methods used to play or interact with multi-user Instruments and how instruments differ regarding shared collective control. The SWARMED instrument would be classified as a variable-number of users, single-role (but changing) instrument, with differing inter-dependencies, as some instruments share state between users.

Other fields, such as software engineering, have focused on the issue of adoptability, especially in the context of software [14]. Adoptability is different than portability. Portability is about whether or not software can run on a platform, adoptability is whether or not the software is compatible enough with a user such that the user adopts and uses a tool. In the case of a computer music interfaces on mobile devices, both portability and adoptability are important as the audience will have a wide range of mobile devices (portability) but they will receive minimal training and yet must navigate to and learn the user interface (UI) quickly: they will have to adopt the use of the UI. Software engineers try to make tools adoptable by integrating their tools into user interfaces that users already use (like Microsoft Office) [14].

Thus we propose a system different from previous systems that addresses user adoption via captive-portal technology, addresses portability via web-standard browsers, and attempts to enable audience collaboration by leveraging the mobile devices of the audience.

3. ABSTRACT FRAMEWORK

In this section we describe the abstract structure of a SWARMED instrument. A SWARMED instrument would include the user, the performer, the network, the infrastructure, and the computer music instrument under control. This is depicted in Figure 1.

The user interacts with the instrument by logging onto a captive-wifi-portal, a wireless network. The captive-portal whisks the user's web browser away to an initial webpage that determines the capability of the the user's mobile device and webbrowser and then navigates the user to appropriate and compatible user interfaces.

When the user interacts with the instrument their browser sends HTTP commands over the wireless network back to an application webserver. This application webserver is dedicated to passing on the user's input to the computer music instrument. Application webservers, such as Ruby on Rails [1] or Mongrel2 [12], are used because they allow *handlers* to run, which are functions or *processes* (running programs) that respond to a request. These handlers often interact with longer running processes, such as an instrument. A *handler* receives and responds to a HTTP request at a particular *Uniform Resource Locator* (URL), massages the input, and then sends appropriate signals to the computer music instrument.

Often it is useful to model the instrument as a continuously running process that receives instructions from the user interpreted by the application webserver. Often the handler is the computer music instrument itself and the input from the user is immediately interpreted. The instrument is then responsible for producing audio that can be again routed through audio interfaces and played back to the audience.

Thus a SWARMED instrument consists of web-based user interfaces delivered via a captive-portal to the web-browsers hosted on the audience's mobile devices. These UIs send requests back to an application webserver, that massages the requests as necessary and then signals the computer music instrument, which produces audio for the audience to hear, often over a central public announcement system (PA).

4. CONCRETE FRAMEWORK

We performed SWARMED 3 times using the same, but slowly evolving, implementation. Our concrete version of SWARMED is available for download at `http://url.ca/chkz5` [5], and is Open Source. Figure 2 depicts a concrete configuration of SWARMED.

The wireless network was provided by a Linksys WRT (a popular Wi-Fi router) which routed traffic to a laptop that acted as an internet gateway. The laptop was running `dnsmasq` [8] which rewrote all *domain name system* (DNS) requests back to the laptop itself, and acted as a *Dynamic Host Configuration Protocol* (DHCP) server, which assigns IP addresses to computers, for the Linksys WRT. Thus all hostnames would resolve back to the laptop so any HTTP URL, such as `http://ualberta.ca/` would be redirected to application webserver on the laptop.

The laptop ran the `mongrel2` webserver [12], an application webserver. For instruments that had provided no feedback to the user, we passed off the requests off to a 1-way handler. For applications that had a shared state across multiple users, that had bi-directional communication, we implemented separate bidirectional handlers.

The handlers would usually take the input, transform the inputs and then set global values or trigger instrument events within an running CSound-based [15] instrument.

A generic handler, a front-page, routed the users to the instruments. It also checked the user-agent string of the user's web browser and limits the user to instrument UIs compatible with their browser.

4.1 Wireless Network Configuration

While a sole laptop can operate as the master in a WiFi network, we opted to use a third party WiFi router (a Linksys WRT) to handle the wireless side of the network. This router was plugged into the laptop's Ethernet port and assumed that the laptop was an internet gateway.

The laptop ran DNSMasq and acted as a gateway to the Linksys WRT, the Linksys WRT would route requests from clients to the webserver via this gateway. The DNSMasq program would act as a DNS server and rewrite all DNS queries back to the laptop. This meant that if you attempted to go to any URL or webpage while on the SWARMED network you were routed directly back to the SWARMED webserver.

4.2 SWARMED WebServer

The SWARMED Webserver had 3 goals: to inspect the user's browser and route it to the appropriate instrument, to host the instrument HTML and JavaScript (JS) files, and to ferry requests from the clients to the handlers.

We used the mongrel2 webserver to host our instruments because it was known to have good performance and it allowed handlers to be long running and independent processes. Handlers have to respond to events, but they can maintain state, connections, and do other work in the background, such as maintaining an instance of CSound [15] running so that we could send it real-time instrument events.

Furthermore the webserver allowed us to host the static files, HTML, *cascading style sheets* (CSS), JavaScript (JS), images, that were part of the instrument UIs that we send to the client browsers. The mongrel2 webserver supports static content directories in which we placed our instrument UI definitions.

We also configured the webserver with instrument handlers meant to interpret HTTP gets, HTTP posts, and HTTP XMLHTTPRequest messages (asynchronous HTTP requests) sent from the client browsers relevant to the instruments.

4.3 Front Page Handler

The welcome page was actually a 404 error-code (page not found) server-side handler meant to handle any URL that did not already exist. This meant any file not hosted would be handled by this handler and redirected to our instruments. The handler would inspect the HTTP headers of the client's HTTP request, and determine which instruments were compatible with the client.

Browsers for Android 2.3 and prior tended not to support *scalable vector graphics* (SVG), while most versions of iOS did. Blackberry phone browsers tended to support SVG and so did many of the Windows phones. Laptops and desktop devices supported SVG as well. Any compatibility issue like this that was found would be encoded into this welcome page handler. Thus it would determine the appropriate instruments compatible with the user-agent string of the client browser.

Once the browser compatibility was resolved, an instrument was chosen from the compatible list. We later added a feature where at different times different instruments were enabled or disabled. This meant that the set of instruments in use would slowly evolve and change and the sound of the improvisation would evolve and change as well. The welcome page served as a way to distribute instruments to allow for some composition of the audience's improvisation.

4.4 Handlers

The Mongrel2 webserver is built around the idea of processes that act as handlers. The webserver communicates via an interprocess-communication (IPC) protocol (zeromq) to the processes acting as handlers, and sends a request package containing the information that the client browser had sent. Then the handler is meant to interpret the request and build a response, which is sent back to the webserver and client browser.

In the asynchronous case, where the client browser does not update state from the handler, a simple OK message was returned. In the synchronous case, where state or data from the handler would be shared with the client, a different response would be built (in our case a JSON response message) and sent back to the client. The client code would interpret the response and update the UI. Complicated instruments, with shared global state could be built using the handler's unidirectional or bidirectional communication capabilities.

We had one instrument (see Figure 7) that relied heavily on bidirectional communication, as it had to synchronize state between clients. Clients would view and modify an array of values, represented as a bar graph plot of a waveform. They could drag the bars to change the value. When a value was changed that new value was sent back to the handler via an XMLHTTPRequest. The response object would often provide a new state that integrated that new change and thus update the client's UI with the new values. More than one client could modify the same values and thus client browsers had to also poll for state updates, allowing

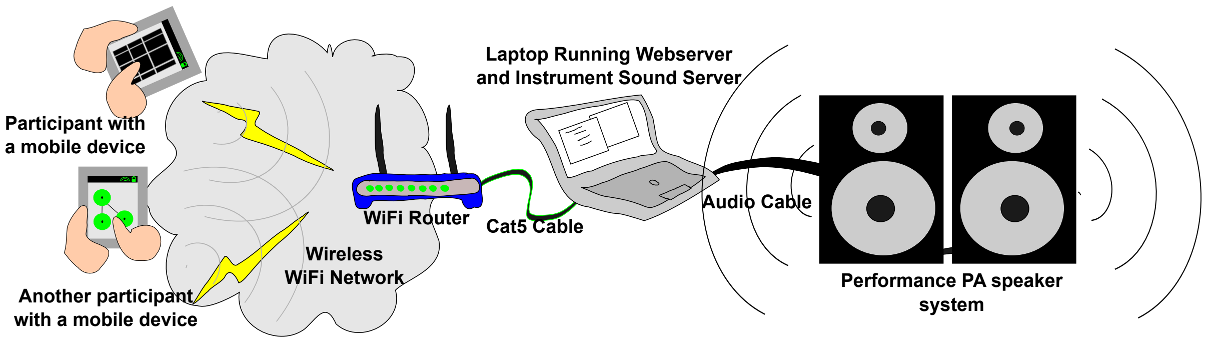
Participant with a mobile device
Another participant with a mobile device
Wireless WiFi Network
WiFi Router
Cat5 Cable
Laptop Running Webserver and Instrument Sound Server
Audio Cable
Performance PA speaker system

Figure 2: Diagram of Concrete Network Architecture

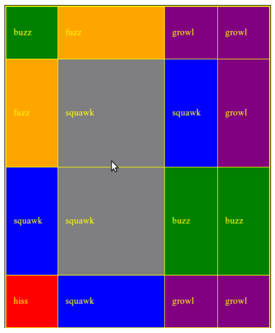

Figure 3: Basic HTML Soundboard UI

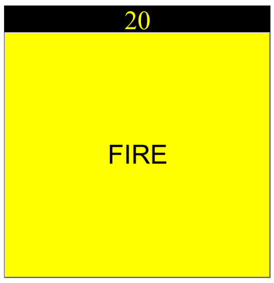

Figure 5: Virtual Bell with wait-time based amplitude

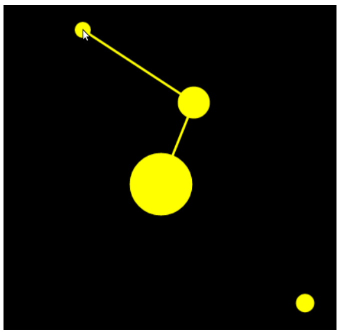

Figure 4: Springy Bouncy Ball Flute Instrument UI based on Siver K. Volle's JS1k entry [16]

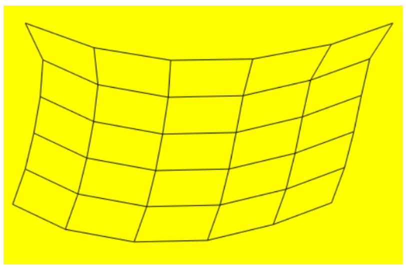

Figure 6: A Virtual Cloth Instrument based on Andrew Hoyer's cloth simulation [6]

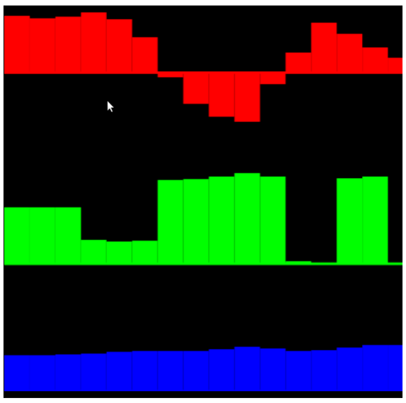

Figure 7: Drawable Wavetable instrument

users to collaborate using the same instrument.

Most handlers simply had to interpret client messages, update their internal state and then update the underlying Csound [15] instrument controlled by the handler. Usually each handler ran its own instance of Csound [15], which would be routed and audio-mixed by the `jackd` audio-routing daemon.

4.5 Instruments

Each instrument consisted of 3 parts:

- An HTML/JavaScript User Interface,
- A handler/server to massage events,
- And a Csound instrument that generated the audio.

We had 7 instruments concurrently running that could play at the same time:

A virtual bell, that grew louder the more the user waited between triggering the bell. It consisted of a simple HTML button whose click would be received by a handler set to calculate the duration and loudness of the bell based on the

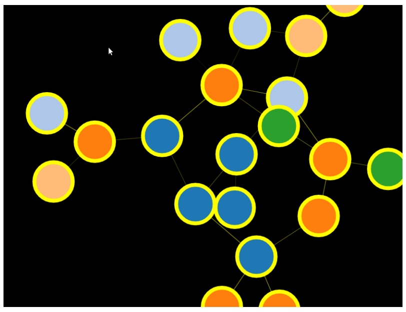

Figure 8: Force Directed Springy Graph Instrument UI based on D3.js by Bostock's et al. [2]

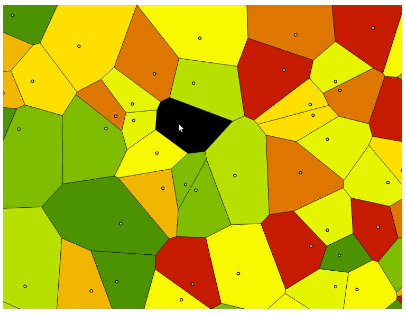

Figure 9: Voronoi Map location and cell based instrument on D3.js by Bostock's et al. [2]

time between button clicks. See Figure 5. This instrument used only HTML and JS and was deployed to all browsers.

The Force Directed Springy Graph Instrument is a graph visualization where the tension and oscillation of graph nodes created buzzy filtered noise and jiggly sounds FM synthesis sounds, see Figure 8. The users can pluck a node and it snaps back into place with spring-like motion. Node motion was a parameter of excitation to the instrument, fast motion makes aggressive sounds. Using HTML, JS and SVG this instrument was only deployed to SVG compatible devices: laptops, iOS, BlackBerry, and later versions of Android.

The soundboard, a simple HTML table based instrument that produces audio based on the Force Directed Springy Graph Instrument, it played similar sounds but in a more syncopated manner based on which part of the grid was clicked. When one clicked a grid box, it quickly inflated, triggering the associated noises and then deflated. See Figure 3. Using only HTML and JS this instrument was deployed to all devices and browsers.

A Voronoi map (a mathematical structure that looks like stained glass, see Figure 9) instrument where the users could drag and select shards, where the size and position of the shard where used as parameters to a dissonant screeching piano/string instrument. The Voronoi map was randomly generated, and not shared. Using HTML, JS and SVG this instrument was only deployed to SVG compatible devices.

A virtual cloth simulation where the shape and control point position of the cloth, depicted in Figure 6, control 36 different oscillators whose frequency and phase were offset by their positions on the cloth. When the cloth was moved or manipulated, the resulting motion of the fabric produced a kind of wooshing sound. This instrument used HTML, JS and Canvas and was deployed to laptops and most phones except Android version 1.6 or less due to performance issues.

*Springy Bouncy Ball simulation*4 relies on the spring-like motion of a weighted ball oscillating between a fixed point and a point that the user drags. It was used to trigger flute

Figure 10: SWARMED being performed at the 2012 Edmonton Works Festival (left) and the 2012 Victoria Noise Festival (right)

sounds where the speed of the weighted ball determined the amplitude of the flute and the direction of the ball's motion determined its pitch. Using only HTML, JS and Canvas this instrument was deployed to laptops and most phones.

Finally, a drawable wavetable where the users would draw and share the same waveform, waveform pitch table, and pitch table of the waveform pitch table. Users could see the changes to these values caused by other users as shown in Figure 7. This instrument was similar to Fabiani et al.'s MoodifierLive [4]. We allocate 4 or more shared instances to the audience, each of which runs concurrently in two modes: linear interpolation and no-interpolation. Linear interpolation sounded smooth and fluid while no-interpolation had a very digital abrasive aspect to it. Using only HTML, JS and Canvas this instrument is deployed to all platforms.

Events were triggered by motion of objects on the user's screen and the user's interaction. If a node on a spring was plucked it would generate events until it was done oscillating. These events would be serialized across HTTP as XMLHTTPRequest objects, sent asynchronously from the user's browser. Too many XMLHTTPRequests can flood the webserver and flood the wifi network (networks have limited bandwidth), thus batching up requests or limiting requests to 10 to 30 per second was appropriate (this can be done within the JS UI running on the client browser).

5. DISCUSSION

SWARMED was performed at the 2012 Edmonton Works Festival and at the 2012 Victoria Noise festival. The maximum audience size was 80 individuals with at least 20 mobile devices (although the system has been performance tested allowing for 60+ concurrent users). See Figure 10 for a photo montage of the performances.

5.1 User Perspective

From the point of view of a user, they received instructions how to connect to the network, and they proceeded to scan for WiFi networks and then connected their device to the SWARMED network. Afterwards their browser was redirected to a welcome page that has chosen an instrument for them to use. They clicked or waited and were redirected to an instrument that they could interact with. Through the event's PA they could hear their interaction broadcast to everyone in the room. The instrument would timeout after a few minutes and switch to a new instrument.

5.2 Intuitiveness

We found that many users lacked the knowledge to connect their phones to a Wifi network. Thus simply connecting to a network was a barrier for some users. This kind of issue can often be addressed with simple instructions printed out on a flyer (Oh et al.[11] used QR codes).

The UIs for the instrument should be simple enough and intuitive enough, in particular they should be aimed at mouse or touch input without a keyboard. Buttons and objects being clicked should be large enough for users on small screens to pick out. One should address the affordances of the devices being used [13].

Since SWARMED relies on audience participation, often the performer has to decide how much time during their performance (if there is any) will be used to address the technical issues of the audience. During our SWARMED performances we often took the role of IT support or sysadmin, as some users will need help getting on the network, some might need help using or navigating the instrument UI.

5.3 Improvisation and Collaboration

One problem with a large number of participants is that each participant wants to hear their contribution, thus one should consider methods of limiting input from a large number of participants. The virtual bell instrument allowed users to play bell sounds, but too many bells would flood the performance, so we limited the amplitude of the bell by scaling the bell amplitude by the duration between a triggering a bell. Thus users who waited longer would produce louder bells than users who did not.

5.4 Timed Sections

One instrument or one set of parameters used throughout a performance can get monotonous. We limited the use of an instrument UI to 3 minutes, after which the user would be shuttled to another instrument UI. We had different distributions or proportions of instruments at different times, this allowed the performance to evolve and change, as well it avoid boring the audience with a single simple user interface. Thus while the audience interaction could be classified as improvisation since their actions are not scripted, they are limited by these timed sections which provide some level of composition and limit free improvisation.

6. FUTURE WORK

This kind of web controlled collaborative instrument can also be used in other settings. For an installation this method of interaction reduces the wear and tear costs but also allows users to interact with an installation using potentially complex software. Furthermore web delivery avoids app compatibility and deployment headaches.

This work can be leveraged in realms other than audio, such as the Heart map by Oh et al. [11]. A user visiting an installation could control the mood, the lighting, or interactions within the installation without actually touching or physically manipulating the installation. Museums could make dioramas more interactive, as users could interact with a rear-screen projection of the diorama's sky.

Furthermore, the flexibility of the web approach allows URLs and hyperlinks to become powerful musical commands. Simple client-side embedded JavaScript or Greasemonkey plugins could turn websites into musical instruments.

7. CONCLUSIONS

In conclusion we have described a framework, that provides for adoptable and interactive audience participation in a musical piece using the audience's own mobile devices. SWARMED is both an abstract and concrete framework for allowing the audience to use the browser that accompanies their mobile device to control part of an audio performance or installation. We described the benefits of this kind of local but virtual interaction in terms of cost, equipment, as well as software engineering concerns such as portability and adoptability. Our use of the captive portal paradigm is novel as it allows for easy configuration and adoption of the audience's mobile devices to control the instrument, and hopefully will be of use to other mobile musical interaction musicians, researchers and developers.

8. ACKNOWLEDGEMENTS

Thanks to Scott Smallwood for inspiration, and to NSERC (Discovery Grant) for their support of my research.

9. REFERENCES

[1] J. Allison. Distributed performance systems using html5 and rails. In *Proceedings of the 26th Annual Conference of the Society for Electro-Acoustic Music*, 2011.

[2] M. Bostock. D3.js - Data-Driven Documents. `http://d3js.org/`, 2012.

[3] L. Dahl, J. Herrera, and C. Wilkerson. Tweetdreams: Making music with the audience and the world using real-time twitter data. In *International Conference on New Interfaces For Musical Expression*, Oslo, Norway, 2011.

[4] M. Fabiani, G. Dubus, and R. Bresin. MoodifierLive : Interactive and Collaborative Expressive Music Performance on Mobile Devices. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 116–119, 2011.

[5] A. Hindle. SWARMED blog post. `http://url.ca/chkz5` and `http://skruntskrunt.ca/blog/2012/06/23/swarmed/`, 2012.

[6] A. Hoyer. The cloth simulation. `http://andrew-hoyer.com/andrewhoyer/experiments/cloth/`, 2010.

[7] S. Jordà. Multi-user Instruments: Models, Examples and Promises. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 23–26, 2005.

[8] S. Kelley. Dnsmasq, a dns forwarder for nat firewalls. `http://www.thekelleys.org.uk/dnsmasq/doc.html`, 2012.

[9] P. Kline. Unsilent night website. `http://unsilentnight.com/`, 2012.

[10] J. Oh, J. Herrera, N. Bryan, and G. Wang. Evolving the mobile phone orchestra. In *International Conference on New Interfaces for Musical Expression*, Sydney, Australia, 2010.

[11] J. Oh and G. Wang. Audience-participation techniques based on social mobile computing. In *Proceedings of the International Computer Music Conference 2011 (ICMC 2011)*, Huddersfield, Kirkless, UK, 2011.

[12] Z. Shaw. Mongrel2 web server project. `http://mongrel2.org`, 2011.

[13] A. Tanaka. Mapping Out Instruments, Affordances, and Mobiles. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 88–93, 2010.

[14] S. Tilley, H. Muller, L. O'Brien, and K. Wong. Report from the second international workshop on adoption-centric software engineering (acse 2002). In *Software Technology and Engineering Practice, 2002. STEP 2002. Proceedings. 10th International Workshop on*, pages 74 – 78, Oct. 2002.

[15] B. Vercoe. *Csound: A manual for the audio processing system and supporting programs with tutorials.* MIT, 1992.

[16] S. K. Volle. JS1k, 1k demo submission [385]. `http://js1k.com/2010-first/demo/385`, 2010.

[17] N. Weitzner, J. Freeman, S. Garrett, and Y.-L. Chen. massMobile - an Audience Participation Framework. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Ann Arbor, Michigan, May 21-23 2012. University of Michigan.