COLLABORATIVE MUSICKING ON THE WEB

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MUSIC
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Hongchan Choi

May 2016

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Jonathan Berger, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Chris Chafe

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Ge Wang

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

The Web has evolved far beyond its original function as the vehicle of information sharing through hypertext, but the spirit of collaboration remains central to the Web. With the arrival of HTML5 and new browsers capable of streaming audio and complex graphics, new and unprecedented opportunities have arisen. The advent of Web Audio API in 2010 took real-time audio processing within web browser to a new level, giving birth to a new breed of interactive, visually rich, and networked music software.

This thesis investigates the potential of web music technology within the context of collaborative musicking. I adopt the verb 'musicking', coined by the late Christopher Small, in the title, to underscore the nature of music as an active process rather than a static object. The dissertation commences with a concise review of the intersections and convergence of the Web and music technology. Following this historical review of web music technology, a theoretical framework is proposed. The subsequent chapters describe technical efforts to implement this theoretical basis and empirical experimentation through a series of 'case studies'. We conclude with a critical evaluation of the work and a look toward the future.

The work offers a novel model for classification of collaborative computer music production (Chapter 2), and an innovative software framework that facilitates the development of web-based music applications (Chapter 3) that are illustrated in case studies (Chapter 4).

Unlike many written dissertations, this thesis documents a living and organic project that is best studied directly engaging with the environment described within. The reader is encouraged to pursue the thesis while directly interacting with this project on the Web. All code examples and operational prototypes discussed in the text are available at `https://ccrma.stanford.edu/~hongchan/WAAX` keeping in mind that the very nature of this project is one that enables and encourages collaborative creativity - indeed, 'musicking' - on an unprecedented massive scale.

Preface

Active engagement with music (henceforth termed '*musicking*') has been radically affected by technology. Recently the progressive convergence of the accessibility of the Web and the flexibility of music technology tools has created new opportunities for collaborative and creative musicking. With the implementation of real-time digital audio processing and interactive control via MIDI on the web platform, we can imagine an ideal framework which could provide a complete and comprehensive music production system that will run on a web browser.

Although the Web was originally invented as a platform for collaboration in scientific research, it has since expanded in scope and reach, and in the process, absorbed a variety of technologies that broadened the range of collaborative possibilities. Inevitably, music technology was a part of this process of incessant expansion. Today the web browser 'speaks' the language of computer music while maintaining its core purpose of collaboration. The broad implications of this new collaborative framework has not yet been examined or articulated both in terms of possibility and in terms of potential reach to a broader set of users. What novel modes of interaction and collaboration are available?

Despite general agreement that music creation is a highly collaborative process, there is considerable divergence in defining issues of scope and methodology of collaboration. These disagreements are primarily due to the subjective nature of assessing and measuring 'successful' collaboration. Therefore the quest toward constructing a qualitative metric for its evaluation is destined to be contentious and fragile.

The Web is known for openness and productivity but efforts to take advantage of these benefits via music making have yet to prove fruitful. The technical breakthrough itself does not unfold the future for us along an unambiguously directional path. We have arrived at the intersection of the Web and music technology yet what lies ahead are uncharted lands.

Without fully knowing what can be done, it is impossible to build a metric for evaluation of success in terms of creative and collaborative use. Collecting data points through numerous iterations will be the first step in mapping the unexplored territory. This thesis aims to establish the foundation upon which this expedition will be accelerated and realized.

Acknowledgments

First and foremost, I am truly thankful to Prof. Jonathan Berger. Without his guidance, support and wisdom, none of my work described here could have come to this world. Immense gratitude to Prof. Chris Chafe and Prof. Ge Wang for mentorship and inspiration throughout my doctoral research. I am also deeply grateful to Prof. Paul DeMarinis and Prof. Michael Bernstein for being on my thesis defense committee and offering valuable input from various perspectives.

I am profoundly indebted to Prof. Jun Kim and Prof. Woon Seung Yeo for helping and supporting me to open a new chapter of my academic career at Stanford.

Very special thanks to John Granzow and Juhan Nam for enriching my research and life at CCRMA. I also thank Jieun Oh, Sang Won Lee and Hyung Suk Kim for insight and friendship. I have been incredibly fortunate to work with superb colleagues at CCRMA; Luke Dahl, Jorge Herrera, Nick Bryant and Fraois Germain. My sincere thanks to Debbie Barney, who helped me from the beginning and still helping me so I can survive the life of Stanford.

I am honored to work with brilliant people in Google Chrome team; Chris Rogers,

Chris Wilson, Raymond Toy, Boris Smus and Greg Simon. Chrome browser pioneering Web Audio API was the biggest motivation of this entire research project.

Finally, my heartiest thanks to my wife Sung Hyun for her sacrifice and unwavering support, to Jaewon and Eugune for giving me strength with their beautiful smiles.

Contents

List of Tables

List of Figures

List of Listings

Chapter 1

The Past: The Web and Music Technology

1.1 The birth of the Web

At CERN, the largest particle physics laboratory in the world, Tim Berners-Lee observed the challenges researchers were having in sharing information. He devised a solution based on the rapidly-developing internet and the emerging technology called Hypertext [113]. He subsequently created the first web page editor, browser and server application. By the end of 1990, the first web page was served publicly to the open internet [101].

In 1993, Marc Andreessen and Eric Bina created NCSA Mosaic, considered to be the first graphical web browser [79]. Mosaic was released to the public for free and supported UNIX and Mac OS. Not only did it allow its authors to add GIF

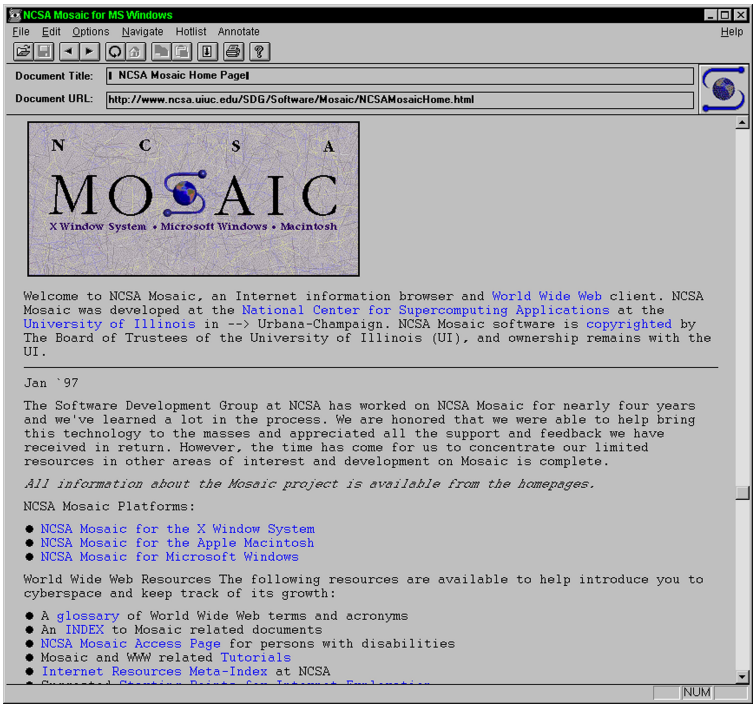

Figure 1.1: NCSA Mosaic Version 1.0

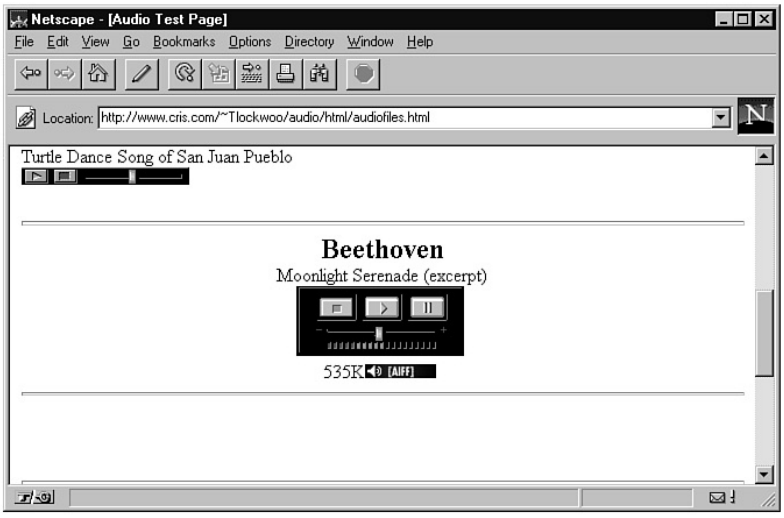

Figure 1.2: LiveAudio Plugin in Netscape Navigator

image files to the web page but also enabled the audition of various audio files such as AIFF (Audio Interchange File Format), AU (NeXT, Sun Microsystem) and WAV (Microsoft) [77].

1.2 Live Audio on the Web via Plugin

A year later, together with Jim Clark, Andreessen created Netscape Navigator. The browser was equipped with various advanced features. It was an instant success. In the second version of Netscape Navigator, they introduced NPAPI (Netscape Plugin Application Programming Interface) enabling developers to author plugins [82]. This was adopted and used by other browsers for decades to come.

The most common use case of the plugin architecture is extended functionality of the browser through support for a variety of audio and video file formats. The plugin ran within a web page, whereas older browsers had to launch an external application to handle the downloaded content. For example, the LiveAudio plugin could play audio or MIDI files directly on the page with just a few lines of JavaScript code [72].

Being able to serve multimedia content consistently across various platforms (i.e. different operating systems or devices) was hugely advantageous both for consumers and industry. The plugin system quickly gained industry momentum and eventually became a de facto standard of delivering multimedia content on the Web.

However, this advance concurrently introduced a potential security disaster. NPAPI and ActiveX (Microsoft's plugin architecture) by design open up a gateway for a third-party plugin to obtain complete access over a user's system [2]. Technically

this loophole makes any computer running a NPAPI plugin vulnerable to malicious attack from the Web. Nevertheless, the rapid growth of the plugin lasted for another decade.

The industry learned a valuable lessons: one can never be too thorough when introducing a new feature to the browser. The modern engineering process of the browser takes extreme care when it comes to the security of its billions of users.

1.3 MIDI Playback on the Web

Since MIDI (Musical Instrument Digital Interface) captured the attention of the music industry, the computerized studio setup based on MIDI connectivity and communication was widely adopted by many musicians. Apple's 1994 introduction of QuickTime 2.0 broadened the scope of MIDI with a built-in software synthesizer and MIDI playback. Although the quality of sound was barely acceptable for professional purposes, a hardware synthesizer was no longer required to play MIDI files [96].

Sharing MIDI files for both production and consumption became the norm, and web browser support of MIDI playback to facilitate these exchanges naturally ensued. To that end, QuickTime plugin for Netscape Navigator 3 was released shortly after enabling playing a MIDI file without leaving the web browser [80]. Microsoft also released a similar software component offering the GS sound set from Sound Canvas which was licensed by Roland corporation in 1996 [95].

Despite the compactness of MIDI files, the discrepancy in the quality of sound

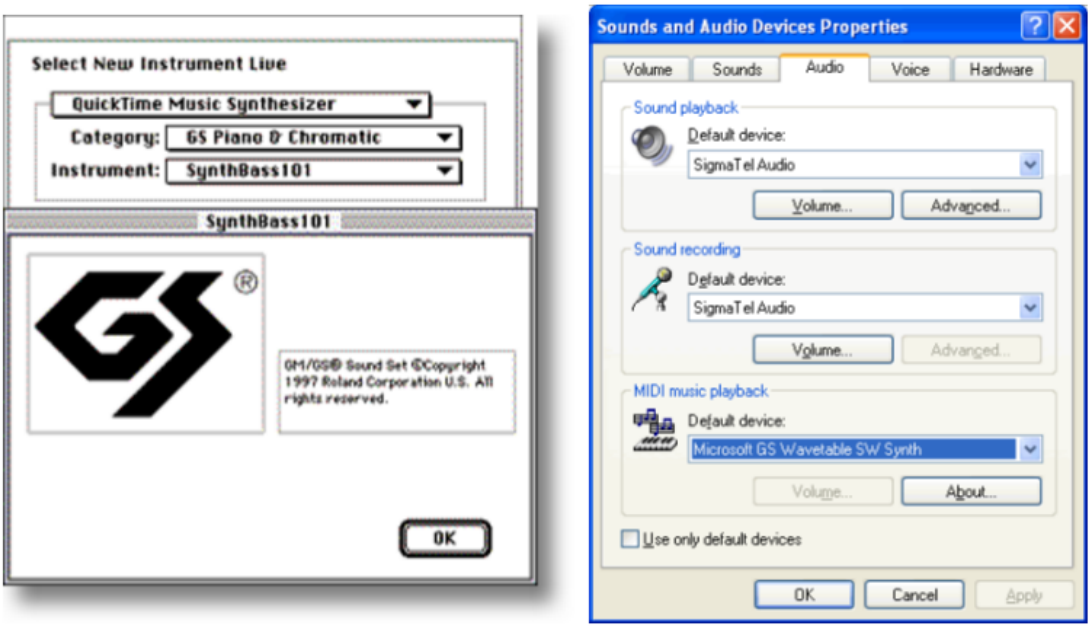

Figure 1.3: Software synthesizer setting in QuickTime and Windows

across various software synthesizers made it difficult to advance as a popular distribution platform. Eventually embedding MIDI file on the Web started fading away as MP3 audio arrived. The massive success of MP3 swiftly penetrated into the web ecosystem replacing the majority of embedded audio content on the Web [65].

1.4 Flash, The Doomed Future

In the early days of the Web, serving animated pictures on the web page was wildly popular even when the technology was limited in several ways. The specification of the GIF89a format, the revised version of GIF, while supporting animation, clearly stated that it was sub-optimal for animation.

> "Animation - The Graphics Interchange Format is not intended as a platform for animation, even though it can be done in a limited way [40]."

Despite considerable loading time and static looping, web authors were willing to make these compromises to have animated images on their work. Although the animated GIF was capable of handling most common use cases, it lacked the flexibility for re-sizing and user interaction. This missing bit eventually motivated the new direction of animation on the Web.

FutureSplash, created by a small company named FutureWave, was released to showcase the new interactive animation technology on the Web. This project later became Flash when the company was acquired by Macromedia in 1996 [56]. With a one megabyte investment for the plugin module, Flash offered scalable vector graphic, animation, audio and interactivity. The cross-platform support also played a great role in its unprecedented popularity.

With the tremendous success of "making the Web interactive", Flash started to evolve into comprehensive multimedia infrastructure. In 2000, the first version of ActionScript, a Flash-based script programming language, introduced an application programming interface (API) for web developers [54]. This programmable multimedia environment with cross-browser support constituted a significant breakthrough for web-based content delivery.

In 2006, a real-time audio feature was released with version 3 of ActionScript conceptually facilitating an interactive music application running directly in the browser. A web service like AudioTool is an exemplary project of interactive audio built with

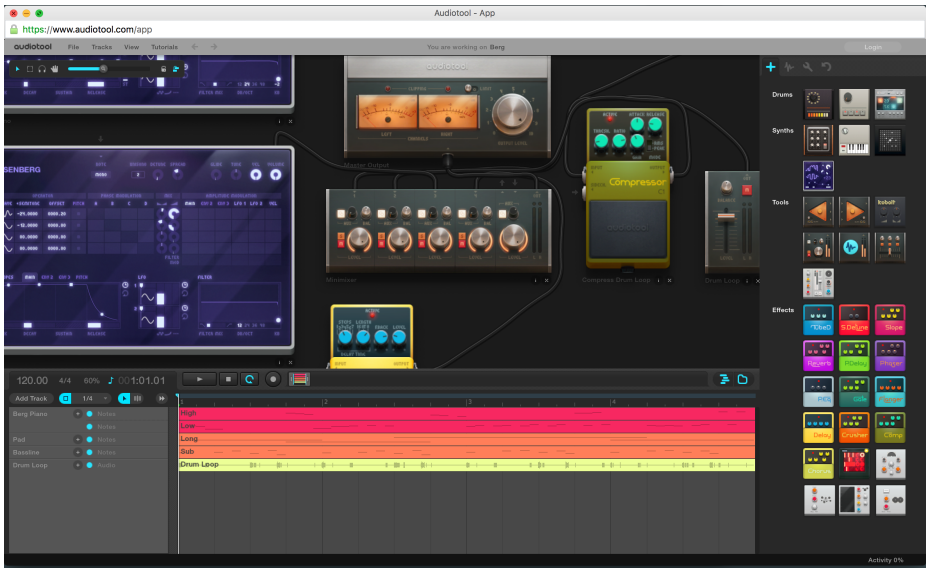

Figure 1.4: Audiotool, a Flash-based music tool [55]

Flash [55].

However, it was only a rose-colored glimpse of the future. The entire Flash system was built on top of NPAPI and the plugin module was the most exploitable target for hackers because of its unprecedented popularity and numerous security loopholes [3]. In addition, Flash failed to adapt to the era of mobile device.

> "Flash was created during the PC era for PCs and mice. Flash is a successful business for Adobe, and we can understand why they want to push it beyond PCs. But the mobile era is about low power devices, touch interfaces and open web standards all areas where Flash falls short. ... *(omitted)* ... New open standards created in the mobile era, such as HTML5, will win on mobile devices (and PCs too). Perhaps Adobe should focus more on creating great HTML5 tools for the future, and less

on criticizing Apple for leaving the past behind [104]."

Discarding this most beloved technology was far from easy. After Steve Jobs announced, in an open letter, the disabling of Flash from Apple's mobile platform and the new specification HTML5 rose on the horizon, the industry finally accepted that it was time to part ways with Flash.

1.5 HTML5 and Modern Browsers

The rise and fall of Flash bequeathed several valuable lessons. Although it was a mere glimpse at possibilities, the industry was unequivocally convinced about what the interactive web could do. With the need for extreme caution to ensure security and protect user privacy there was a collective effort to create the next generation of web technology called HTML5. For a decade since 2004, HTML5 specification has been written by WHATWG and W3C [59].

Beside being the next wave of the web standard, HTML5 effectively replaces Flash when supplemented with adjunct components such as JavaScript, CSS and other Web APIs. Particularly remarkable, were the performance improvement on JavaScript after the introduction of HTML5 [60]. The engineering effort from industry leaders such Apple, Google, Microsoft and Mozilla pushed the boundary making the browser a viable application platform.

In a concerted effort to avoid repeating past mistakes, the web community established several key concepts for the unbiased growth of the platform and the most important one is "Open Web Platform", which manifests the philosophical foundation

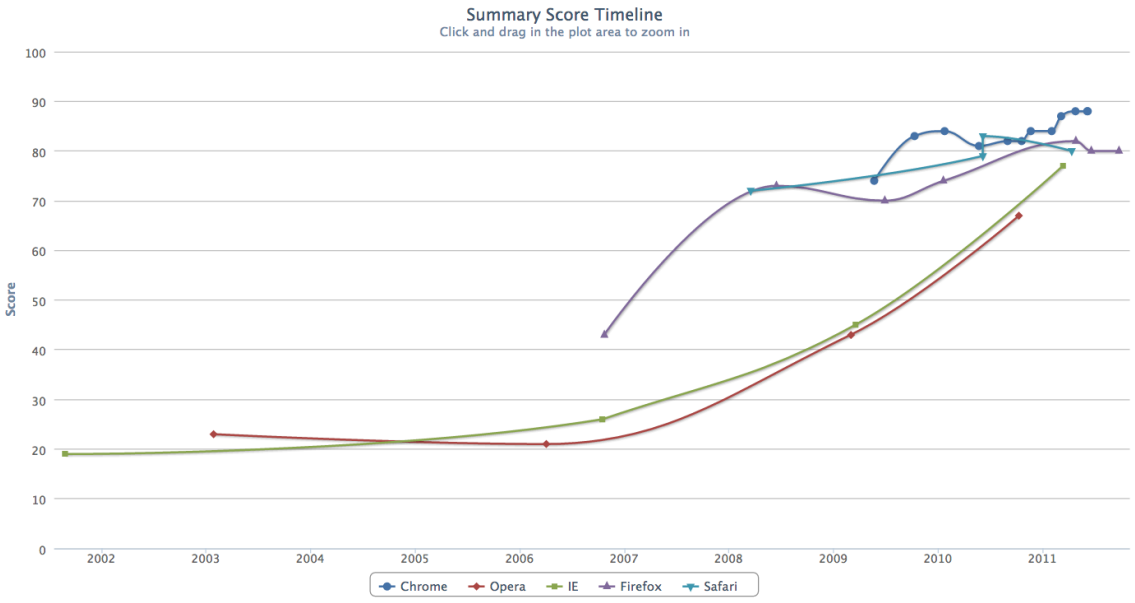

Figure 1.5: Web browser performance improvement between 2001-2011 [60]

of the future direction of the Web.

> "The Open Web Platform is the collection of open (royalty-free) technologies which enables the Web. Using the Open Web Platform, everyone has the right to implement a software component of the Web without requiring any approvals or waiving license fees [103]."

As stated above, HTML5 is a standard that encourages open competition between implementers within the guideline of specification. Thus the discourse toward the agreement between parties is always open for debate and publicly available on the Web [23]. It is a painstakingly laborious process but we learned the necessity of being meticulous from the Flash disaster.

To embrace the drastic changes in the specification, the browser implementers

needed the modern software framework to build the browser with security and performance. Through their engineering effort, the new generation of browsers were released to support HTML5; Firefox by Mozilla (2002), Safari by Apple (2003) and Chrome by Google (2008) - that are backed by modern document rendering machinery and JavaScript engine [38, 110, 98, 14, 66, 21]

Although it was the first meaningful step toward an improved and more robust web ecosystem, the majority of the HTML5 specification remained in flux until it reached 'recommendation' status in 2014 [61]. Different browser vendors implementing numerous features based on a moving target caused cross-browser incompatibility and web developers are still struggling to make their software work on every browser [76]. However, it is a price worth paying rather than dealing with the security flaws and instability issues of Flash.

1.6 Web Audio API and Web MIDI API

The concept of 'compliance and competition' successfully held each other in check while improving performance consistently, but it also encouraged pushing boundaries of different aspects of browser functionality. For example, Canvas and WebGL API were introduced for interactive 2D/3D graphics [105, 69]. These cutting edge technologies transform the browser into a powerful graphic workhorse that can effectively replace Flash's graphics functionality.

From the beginning of HTML5, the $\langle audio \rangle$ element has existed as an audio counterpart to enable direct audio streaming via internet. Although it was sufficient for

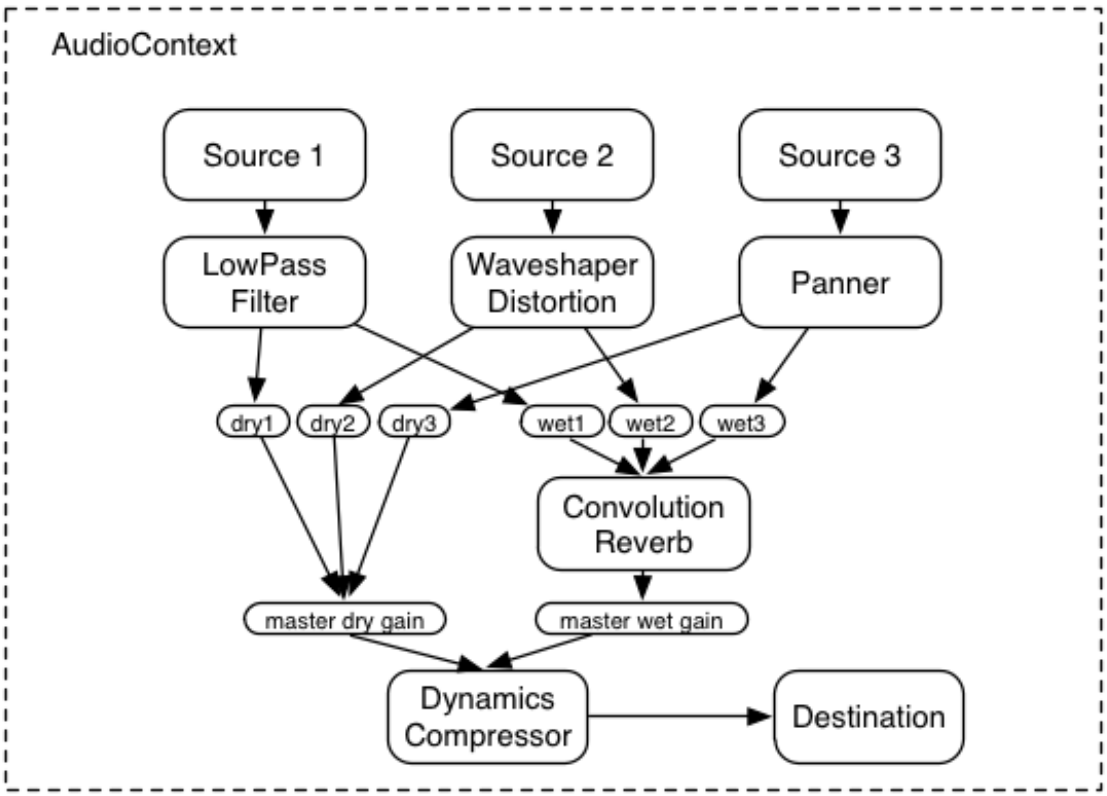

Figure 1.6: Modular routing example with Web Audio API

common use cases such as background music of a page or music player application, it lacked the support of the real time audio processing or synthesis through JavaScript API, making the game industry hesitant to drop Flash as an entirety. To that end, Google Chrome and Mozilla FireFox wrote up two contrasting proposals named Web Audio API and Audio Data API around 2008 [107, 7].

Web Audio API is a high-level JavaScript API that offers a variety of audio nodes such as an oscillator, filter, panner and convolution for sound synthesis and processing. By connecting nodes, one can create an application that generates and changes the

sound dynamically. Thanks to its convenience, W3C chose to move forward with Web Audio API. Audio Data API was deprecated because the latter merely offers a way of manipulating the audio stream without built-in functionalities.

While Web Audio API offers the machinery of sound synthesis and processing, the main goal of Web MIDI API is to empower the web browser to speak MIDI, the most prominent networking protocol for the musical instrument, which has been tested and proven extensively for decades [109]. It is designed to be nicely aligned with Web Audio API by opening low-level access to MIDI data stream to MIDI-compatible musical instruments.

At the time of writing, Web Audio API is supported by all the major browsers such as Chrome, FireFox, Safari and Edge. Web MIDI API is relatively recent and currently supported only by Chrome.

1.7 Problems Remain

This chapter constitutes an opinionated retrospection on the development of web-based music technology. Its aim is to contextualize the current situation and how we arrived where we are. Although the recent development of Web Audio/MIDI API opened up new possibilities of doing computer music on the Web, being able to use them does not necessarily mean that we can build something useful or possibly better than what we have. To move forward with imagination, a pivotal question needs to be answered:

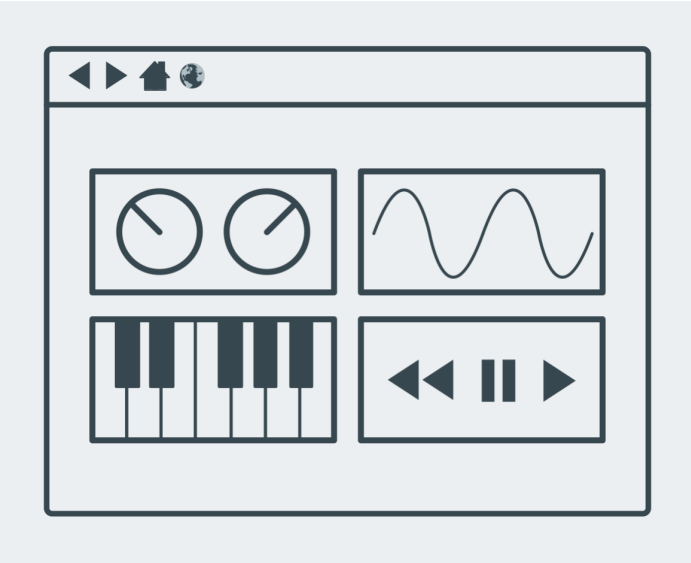

Figure 1.7: Browser as music platform

"Can the Web be a viable platform for music making?"

Few claim that this is a solved problem. Although there are some issues need to be addressed to fully compete with the native music software, the web-based application has remarkable advantages and it is evolving rapidly to catch up with the native one [20, 89, 115]. If the answer to the question above is 'yes', what else does remain? The question which has not been fully explored is:

*"Is it possible to **collaborate** for the music creation using the Web?"*

This question requires more than the technical improvement. The following chapter continues the discussion by analyzing how music collaboration works and building a theoretical framework as a basis of the experimentation.

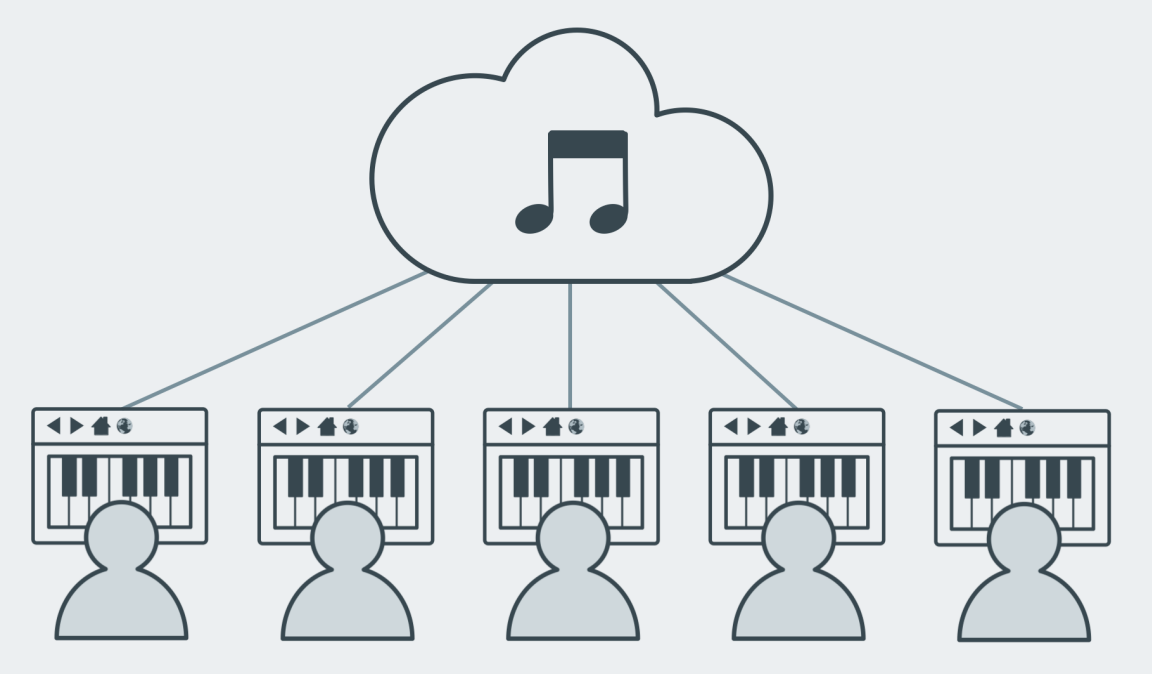

Figure 1.8: Collaborative Musicking on the Web

Chapter 2

Modeling Computer Musicking

This chapter reviews research that addresses various theoretical issues relating to, and perspectives on computer-supported collaboration including computer-mediated musical activities.

However, prior to assessing existing literature it will be useful to clarify the scope and organization of this chapter. An idea proposed by Dahl to explain exotic traits of NIME research deeply resonates with the view of this chapter. Borrowing an idea from HCI and design studies, Dahl notes the similarity between NIME research and the process of solving a *'wicked problem'*, that is the phrase originally formulated by researchers in the field of social planning.

> "Designing new instruments may not have the same degree of consequence as addressing truly wicked problems such as poverty or urban planning. Yet our task is made of many interacting components, few clear guidelines exist, and the criteria for success are seldom explicit [25]."

The problems within modeling computer musicking are indeed 'wickedly difficult'. This is due to the fact that the technology is relatively new and the data points to define its position and capacity is not sufficient, let alone to offer explicit metrics for evaluation.

Rather than aiming to provide generalizable knowledge, this survey chapter aims to establish a framework that would serve as a 'launch pad' for accelerated iteration. In Dahl's same account, he cites Fallman's idea of 'design exploration' to emphasize the swift iteration instead of defending the work from questions.

> "Design exploration is a way to comment on a phenomenon by bringing forth an artifact that often in itself, without overhead explanations, becomes a statement or a contribution to an ongoing societal discussion [34]."

As a result of the survey, a new classification model that has a set of parameters for modeling computer musicking is presented. Having a model especially tailored for musical activity not only allows us to accelerate experimentation, but also makes each iteration more meaningful by elucidating the interpretation of the outcome.

2.1 Musicking

> "To music is to take part, in any capacity, in a musical performance, whether by performing, by listening, by rehearsing or practicing, by providing material for performance (what is called composing), or by dancing [93]."

In his book - *Musicking*, Small redefines music as a verb - an active process - rather than as a static object. This brilliant shift of paradigm places people into the focal center of musical work, rather than the outcome of the process. Small furthermore attempts to expand the boundary of doing music into listening, preparing and even being inspired. More importantly, he stresses the meaning of 'taking part' in a musical performance by highlighting collaborative and social aspects.

Although the statement seems to be controversial, it underscores the fact that omitting the value of the process makes for a limited and impoverished perspective. Furthermore, he insinuates that the musical activity itself is worth sharing. Like musicking, sharing also has a broad spectrum; one can passively observe the process while others can actively engage by participating. Either way it affects the process and the outcome, thus sharing the context of collaboration will make differences in music making.

Similarly, Hudak and Berger systematically approach the benefit of musicking. They see musicking as a 'mutually recursive process', where everyone involved in the collaboration interprets and synthesizes the interim outcome which triggers the next iteration of interpretation and synthesis by other participants [62]. Although their focus is the case of close-range collaboration such as musical ensemble or jazz improvisation, the idea of mutual recursion network can transcend time and space thanks to the recent breakthrough of the computing technology.

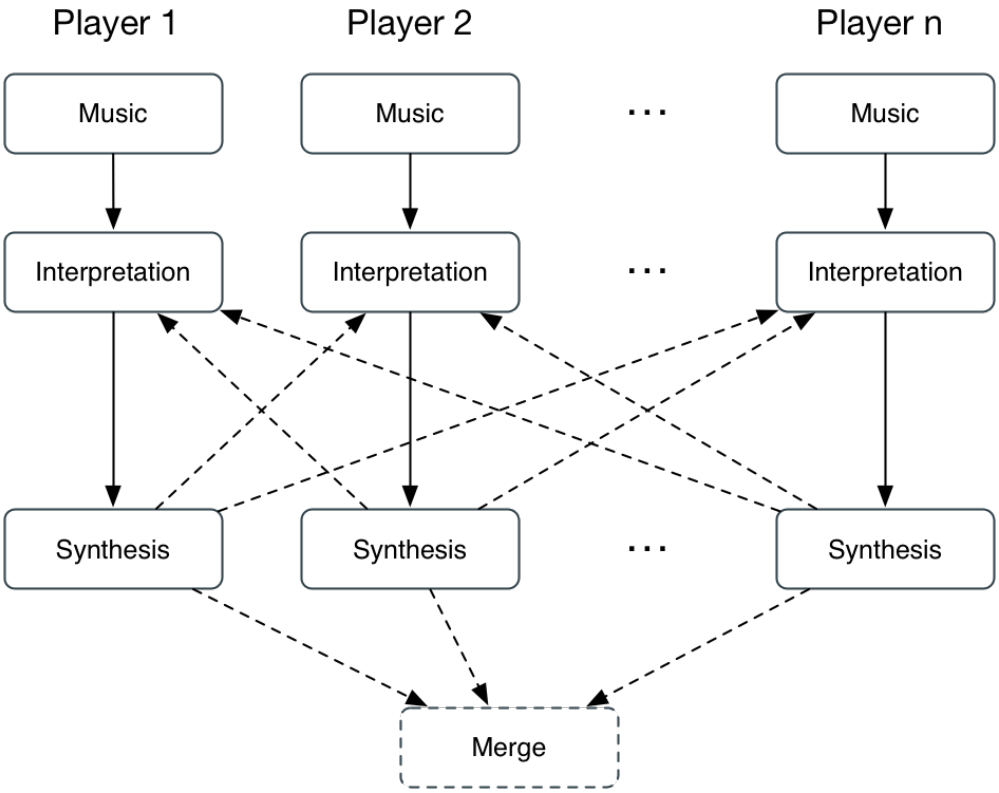

Figure 2.1: Ensemble Interaction: mutual recursion [62]

2.2 Computer-Supported Cooperative Work

CSCW (Computer-Supported Cooperative Work) is a design-oriented academic field based on the idea of utilizing computing technology to improve productivity and cooperation. Although there is considerable variance in the definitions of success and productivity across different cooperative tasks, the fundamental ideas are still valid. Rodden asks a number of questions to underscore why CSCW is more than merely creating networking tools.

> "The research being undertaken poses a number of questions. How can computers be exploited to maximise the synergy of groups?; What kinds of software should be developed?; How do we define group work? To address these problems CSCW involves researchers across a range of disciplines including psychology, sociology, organisational theory, and anthropology [90]."

Rodden also presents a pivotal framework for the classification of CSCW systems. It uses two principal parameters to position a collaborative task in the modeling space: the form of interaction (synchronicity) and the geographical nature of users (proximity). This method only assesses superficial aspects of an activity and not the quality of group work, however, it offers an excellent starting point of the discussion nonetheless.

Inspired by Rodden's framework, Barbosa attempts to classify several collaborative computer music systems using the time and space matrix [13]. He categorizes

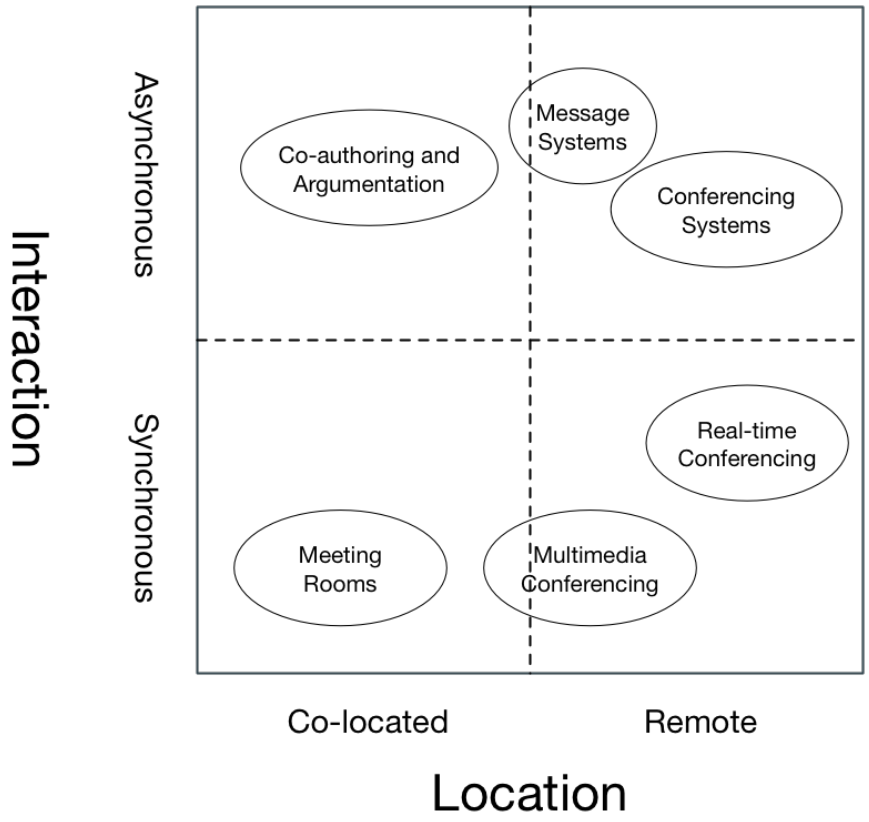

Figure 2.2: Time and Space matrix: classification space of CSCW systems [90]

several case studies into four representative classes prior to projecting them on the matrix:

- Local interconnected musical networks: used in organized events for groups of performers who interact in real time with a set of musical instruments. (e.g. laptop orchestra and electroacoustic)
- Musical composition support systems: used to assist more traditional forms of musical composition and production. (e.g. recording studio)
- Remote music performance systems: used in organized events for groups of multiple remote performers, displaced in space, improvising and interacting synchronously. (e.g. networked performance)
- Shared sonic environments: a new class of emerging applications that explores the Internet's distributed and shared nature. (e.g. sound installation)

His model overlays an additional layer of abstraction on top of Rodden's model to address the technical structure of each class. Although these classes are largely based on actual case studies, it fails to impart an intuition of what they actually are and why they are located at a particular area in the space.

I believe a better perspective can be established by using canonical forms of computer musicking such as electroacoustic music, laptop orchestra or networked performance rather than another layer of abstraction. Also thanks to the recent development in the software infrastructure for computer-supported collaboration, we can

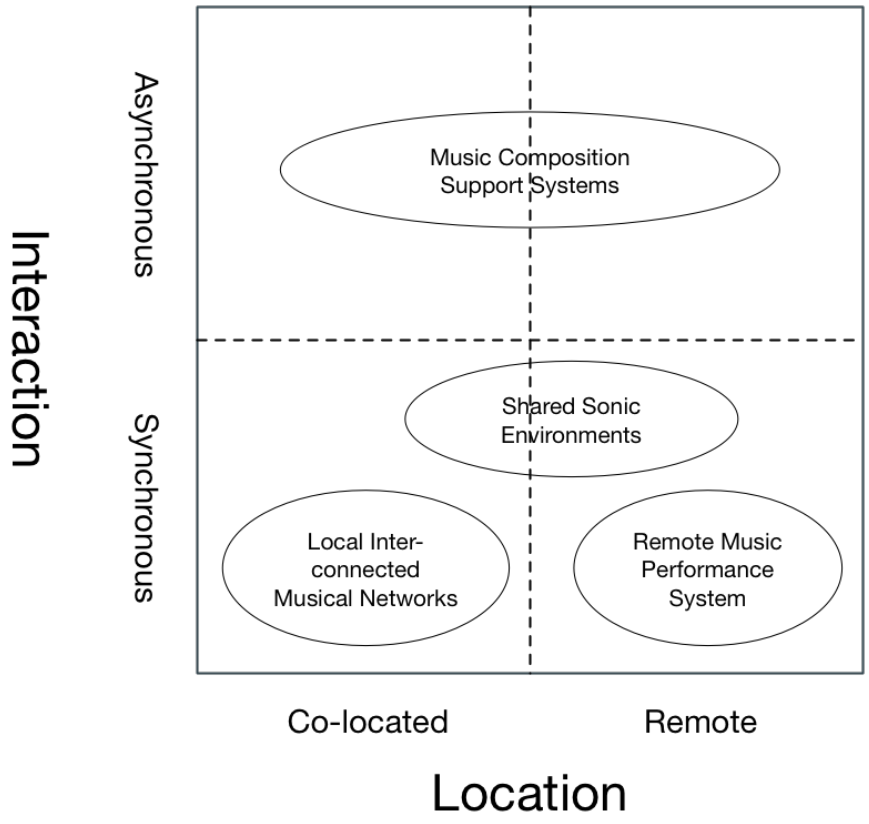

Figure 2.3: A classification space for computer-supported collaborative music [13]

Electroacoustic	Networked Performance	Laptop Orchestra
Studio Recording	Sound Design	DJing
Live coding	Sound Installation	Music gaming

Table 2.1: Canonical computer-mediated musicking

obtain much finer granularity on the both axes of the space. The proposed model in the following sections includes such improvement.

2.3 Computer-Mediated Musicking

As the first step of designing a new model, the enumeration of canonical forms in computer musicking is presented as Table 2.1.

This list is, by no means, exhaustive. The selection here is limited to my own personal observation in past years. The list's primary advantage is that we need not retain every possible form of musicking at hand to build the model. If a new trend rises, we can directly apply the model to such form without worrying about which class it belongs to.

I label these typical activities as *computer-mediated musicking (CMM)*. My definition of computer-mediated musicking is *any kind of musical activity that is involved with human and computer*. The verb 'mediates' makes sense because in many cases the role of computer is crucial rather than being merely supportive. Also the concept of 'working together' is deeply embedded in the term musicking. It is broad enough to cover the most of activities in the field of computer music and succinctly highlights

social and collaborative aspects as well as the computer's involvement.

2.4 Landscape of Synchronicity

The time and space model by Rodden splits the axis of interaction into two segments; synchronous and asynchronous. Although such division seemed sensible when classifying general collaborative tasks few decades ago, now we are given a chance to redesign the axis of synchronicity thanks to the remarkable progress of the software infrastructure for remote collaboration.

> "Whether it's called cloud computing or on-demand computing, software as a service, or the Internet as platform, the common element is a shift in the geography of computation. When you create a spreadsheet with the Google Docs service, major components of the software reside on unseen computers, whereabouts unknown, possibly scattered across continents [52]."

As stated above, the meaning of being 'synchronous' in the computing technology has drastically changed from that of 20 years ago; now the modern infrastructure for collaboration allows people to work together concurrently no matter where they are. Naturally Near real-time (NRT) collaboration became widely popular with the advent of cloud-based software service [64]. Google Docs is an exemplary CSCW system that operates on a massive scale [52]. This movement of cloud-based collaboration became the mainstream of the industry.

However, there is one critical problem we could not resolve over last two decades; the network latency. Because this is the key variable that has a tremendous impact on every aspect of collaboration, there have been multiple studies on the effect of latency in CSCW systems and how to work around it.

Gutwin investigated the influence of delay and jitter to the quality of group work such as coordination and prediction. According to his analysis, the coordination errors are significantly increased with delay of 240 ms or more and the prediction time is significantly affected with jitter of 600ms or more [49]. Chafe analyzed the effect of delay in a musical ensemble, particularly in the context of coordinated rhythmic performance over a dedicated wired connection [17]. He proposed 4 groups of delay:

- Shortest delays (0 to 8 ms): Tendency to anticipate, acceleration.
- Natural delays (8 to 25 ms): Best synchronicity, first plateau, stable tempo.
- Challenging delays (25 to 60 ms): Second plateau, deceleration, and mitigating strategies.
- Conditions above challenging delays (60 ms and beyond): Deterioration, and rapid degradation of playing accuracy.

According to his findings, natural coordination between performers can be achieved around 25 ms of delay whereas the latency of 60 ms and beyond makes the synchronized coordination very difficult. Another interesting fact is subjects were actively coping with the latency by using different strategies for a given delay time; the 'true ensemble' style for the natural delay, and 'leader/follower' for the challenging delays.

		Time of results					
Service	Avg.	21:50:12	21:41:28	21:27:47	21:01:23	20:56:28	20:54:21
Fanout	(203.00)	167.89	217.78	158.58	145.47	268.79	263.44
Firebase	(591.00)	345.79	619.79	969.79	556.32	328.11	729.79
Hydna	(117.17)	117.79	135.74	150.95	168.21	66.47	67.89
PubNub	(257.33)	42.63	97.89	61.32	992.84	121.05	231.84
Pusher	(270.67)	27.72	116.05	72.72	1026.73	227.42	156.00
Realtime.co	(117.00)	27.16	209.37	72.63	54.33	194.47	146.37
DataMcFly	(-)	749.22	1022.45	621.39	669.00	1330.00	NaN

Figure 2.4: A latency benchmark from various real-time hosted services [86]

In both studies from Gutwin and Chafe, the primary control parameter in the experiment is the network latency. They observed that how the delay time affects the interaction between collaborators. However, we can reverse the view point: what kind of collaborative activity can be properly performed at a given range of latency? If we can map these data points on the axis of interaction, it conversely leads us to find an optimum activity for a given delay time. In that way, we can use it as a modeling parameter of a computer-mediated musicking.

At the time of writing, the most recent benchmark on a group of real-time hosted services indicates that the latency varies between 30 ms and 1000 ms [86]. Although this level of delay and jitter seem acceptable for push notification or instant messaging, using one of these platforms for performance-oriented collaboration (e.g. networked concert) simply will not work.

The survey so far suggests that the network ensemble, which requires the latency below 25ms, is not viable with a general purpose infrastructure currently available.

However, it is worth pointing out that network jamming is only a part of the multitude of collaboration classes. Also some classes of CMM requires loose coupling between participants.

The other interesting element to consider is the internal synchronization rate of music software. This rate is commonly imposed by the limitations of the underlying audio/video hardware and the software architecture. I propose the scale of synchronicity as 5 classes shown below:

- A-rate (1 audio sample, 0 to 0.2 ms) : sample-accurate synchronicity. The fastest and the most accurate level of synchronization in music software. It is only possible between software components designed for digital synthesis components (e.g. unit generators).

- K-rate (128 to 512 audio samples, 3 to 12 ms): render quantum synchronicity. The render quantum commonly refers the buffer size of audio callback function invoked by the underlying hardware clock. K means 'control' and usually involved with synthesis parameter change.

- V-rate (16 to 33 ms): video frame synchronicity. 60 fps (frame-per-seconds), which is about 16.67 ms, is considered the threshold of optimum user experience when the real-time visual 1interaction is required.

- N-rate (50 to 800 ms): network-level synchronicity. The range of this class is fairly large due to the fluctuation of delay and jitter imposed by the network system.

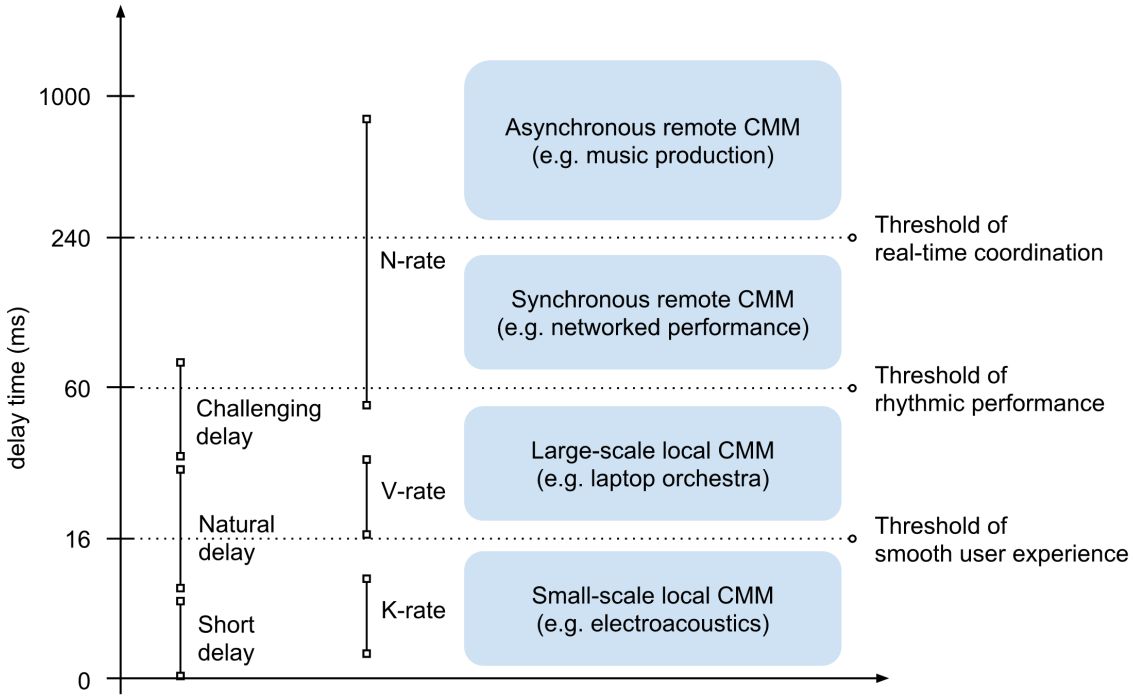

Figure 2.5: The degree of synchronicity

2.5 Proxemics: Degree of Distance

The term *Proxemics* was coined by Edward Hall in his book 'The Hidden Dimension'. According to Hall, Proxemics is *the interrelated observations and theories of man's use of space as a specialized elaboration of culture* [51]. The idea has been a valuable framework in the discussion toward the effect of space, distance and body language between people.

Especially the notion of Personal Space shows that a variety of immediate spaces surrounding a person has different meanings and purposes.

- Intimate distance for embracing, touching or whispering. (15 cm to 46 cm)
- Personal distance for interactions among good friends or family. (46 cm to 122 cm)
- Social distance for interactions among acquaintances. (1.2 m to 3.7 m)
- Public distance used for public speaking. (3.7 m to 7.6 m)

This concept is applicable to canonical CMM tasks. The model from Rodden or Barbosa splits the interaction axis into only two slices: 'co-located' and 'remote'. The idea of Proxemics allows us to have the finer granularity on the 'co-located' side:

- Intimate space: Private work space. Useful for 'holistic conception' (discussed later in this chapter) or studio-based music production.
- Personal space: For small scale real-time collaboration such as electroacoustics.

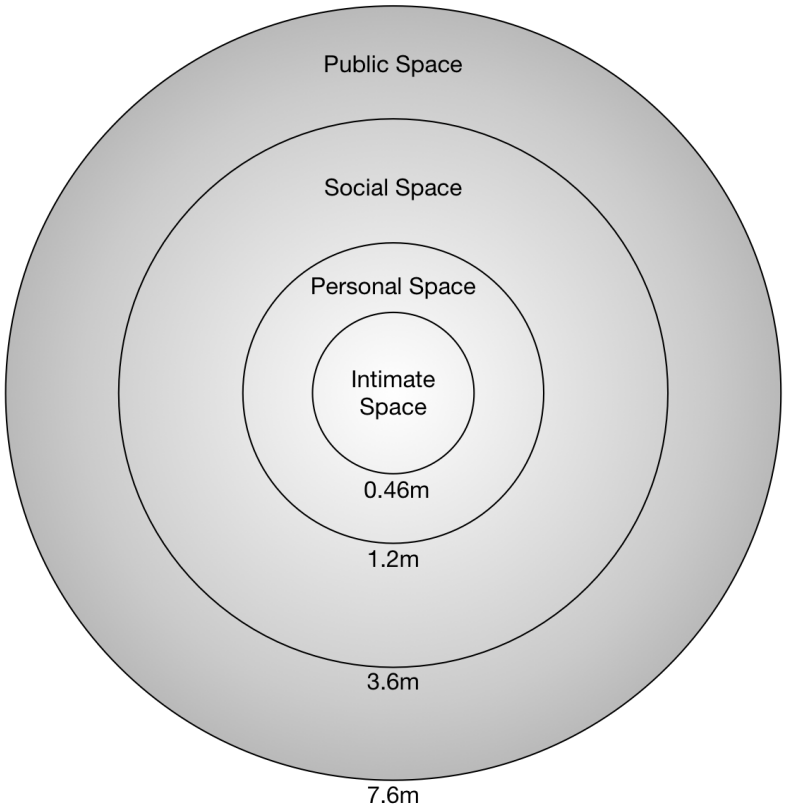

Figure 2.6: Different spaces in Proxemics [51]

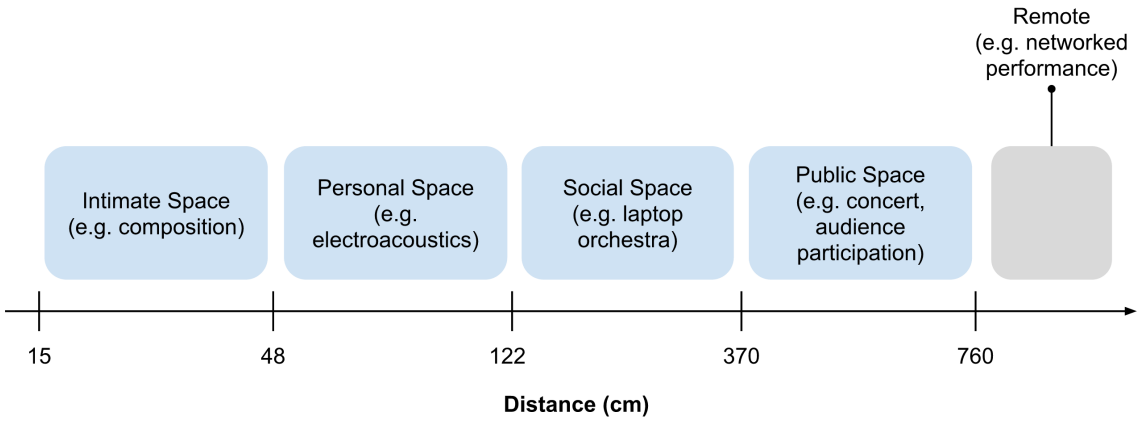

Figure 2.7: The degree of distance: Proxemics on CMM

- Social space: For large scale real-time collaboration such as laptop orchestra.
- Public space: For passive forms of collaboration such as concert or audience participation. This is the threshold of real-time musicking within physical proximity.
- Beyond public space: Classified as 'remote' where CMM is the only way to work together.

2.6 Delay & Distance Model

By incorporating all the ideas presented so far, I propose an enhanced classification framework named Delay and Distance model.

Why this finer granularity on both scales matters? First, the musical interaction is extremely sensitive to synchronicity and proximity compared to non-musical group work. A small glitch or drift in the time can change the entire dynamics of collaboration. Secondly, the physical distance shapes the art form. For example, a duo ensemble and an orchestra can be equally classified as co-located tasks, but there is a clear distinction between these two activities with respect to the physical and mental proximity. Lastly, now we can observe that both axes are roughly in logarithmic scale. I argue that this is a meaningful improvement from the binary nature of the original time-space matrix. This logarithmic scale makes sense because this is consistent with how we perceive synchronicity and proximity in a group work.

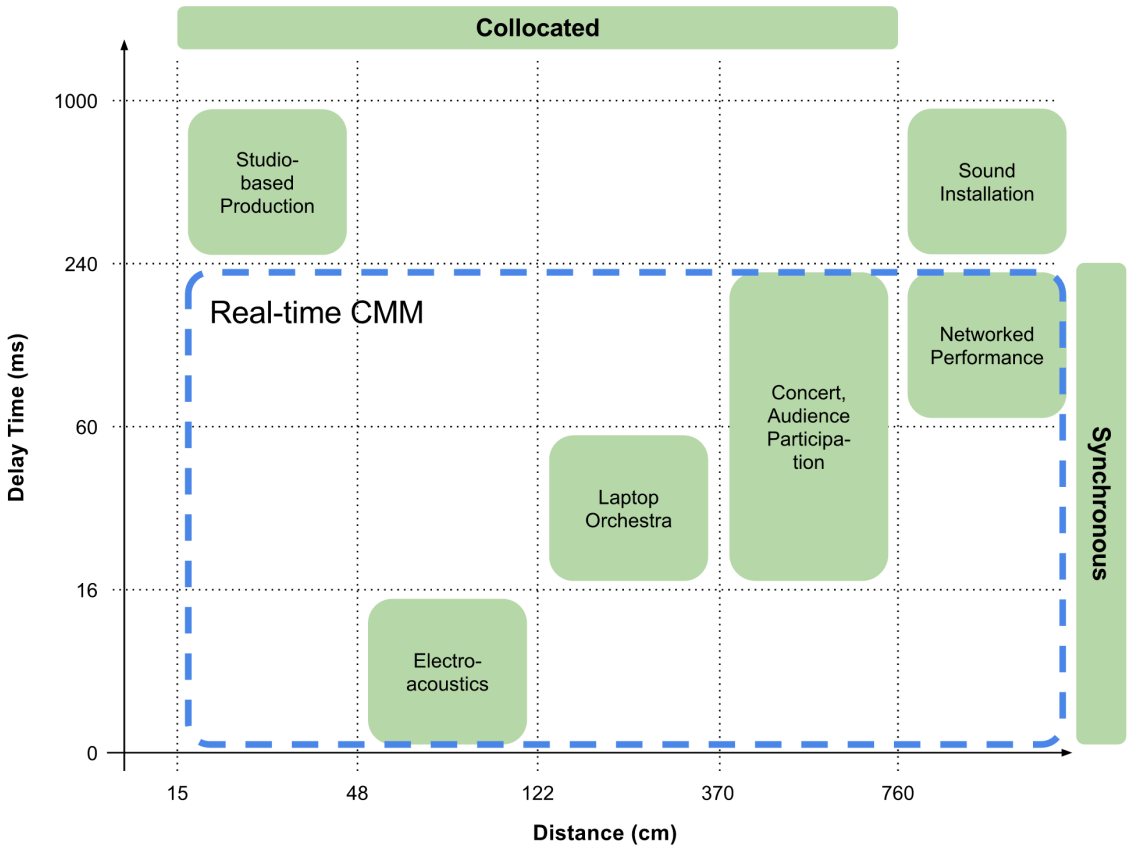

Figure 2.8: The Delay & Distance Model

2.7 Awareness and Privacy

Thus far, we have plotted various styles of CMM into a plane of synchronicity and proximity. However, classifying a task through this model does not reveal its inner nature. What does it mean for a collaboration to have a different degree of synchronicity or a different geological topology? Among numerous outcomes from these two variables, the most important ones are *awareness* and *privacy*.

2.7.1 Awareness

The biggest benefit from the high degree of synchronicity is the strong sense of awareness: awareness is an understanding of the activities of others, which provides a context for your own activity. This idiom also came from CSCW research. Stated by Dourish and Bellotti, the effect of awareness information is universal to any kind of collaborative tasks [28].

When the group interaction happens within the physical proximity, participants tend to keep the the experience organic rather than using a computer-supported technology. This is because we naturally know that the highest synchronicity is essential for the group efficiency, and the face-to-face collaboration is the easiest way to achieve that. Then how can we make people choose the computer-supported technology over the physical interaction even when they are sitting at the same table?

Gutwin and Greenberg proved the enhanced awareness in group work significantly boosts the efficiency [50]. Through a series of experiments, they found that the additional awareness information allows people to use different strategies to complete

the tasks. More importantly participants greatly preferred the awareness-enhanced system even when the organic method is available for the collaboration.

The idea of augmented awareness or value-added awareness was elegantly summarized by Holland and Stornetta in their article *'Beyond being there'* [57]. They proposed how the collaboration system should be designed to transcend the physical interaction:

> "If we ever hope to solve the telecommunication problem, we must develop tools that people prefer to use even when they have the option of interacting in physical proximity as they have heretofore. To do that requires tools that go beyond being there [57]."

The design pattern of shared feedback by Dourish and Bellotti is another example of the augmented awareness information.

> "Shared feedback makes information about individual activities apparent to other participants by presenting feedback on operations within the shared, rather than the private, workspace [28]."

Gutwin and Greenberg also suggested sharing the miniaturized overview of collaboration vastly facilitates the interaction. This is a great example of augmented awareness information that is not possible in the physical group work.

> "The main finding of the study is that adding workspace awareness information to the miniature - visual indications of viewport location, cursor

> movement, and object movement - can significantly improve speed, efficiency, and satisfaction. These awareness components should be included in shared-workspace applications [50]."

Improving the quality and the value of awareness information is the key to CMM system that transcends the physical collaboration. This topic is extensively investigated by numerous CSCW researchers, but it is one of the least explored area in the field of computer music.

2.7.2 Privacy

The creation of art is a very complicated task. As similar to musical instrument design being a wicked problem, music creation also can neither be defined nor evaluated with incontrovertible criteria. This is where CSCW and CMM research part ways. Besides the vague definition and the criteria for success, what makes the CMM stand out among other collaborative tasks?

Wiggins points out that the initial phase of compositional process is spontaneous and cannot be divided. This phase of exploration and experiment, so called 'holistic conception', is mostly about building up ideas and refining them into playable musical motives [114]. Sawyer also argues that creators do not share all their acts of creativity with the world [91].

Rodden claims various co-authoring systems are using two separate workspaces; the private space is for the generation and the refinement of ideas, the public space for sharing ideas with other colleagues. Gurevich applies this concept to CMM; he

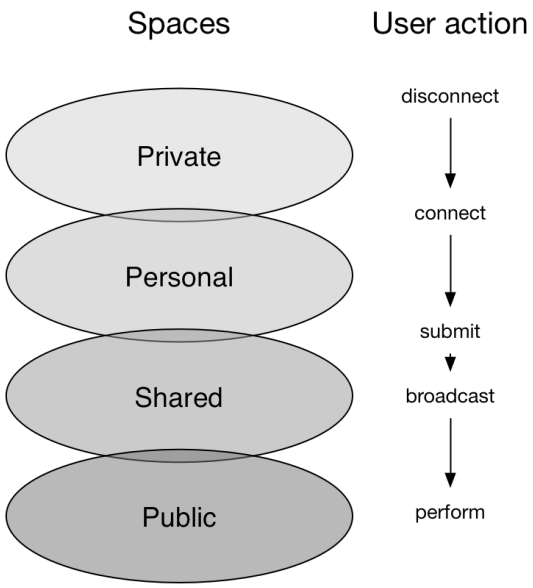

Figure 2.9: Different types of psychological spaces in JamSpace [48]

presents four layers of workspaces along the degree of privacy within the prototype named *JamSpace* [48]. Note that these are territories in psychological terms, not the physical space.

- Private Space: A completely isolated workspace psychologically and technically.
- Personal Space: Other connected users become aware of the user's presence, and he or she becomes aware of them.
- Shared Space: By broadcasting a real-time jam and/or submitting tracks to the system, users can actively share the material with others. Users can see who else is listening to them, and may choose to listen in turn and engage in jamming.

- Public Space: a user airs his or her expression in front of all other users, regardless of whether they are listening.

The necessity of private space is recognized in common CSCW applications, but generally it is not enforced. In an artistic collaboration, however, implementing a certain degree of privacy allows user to be more creative by assisting the process of holistic conception. Fencott and Bryan-Kinns proves the influence of privacy in CMM through the experiment with three different modes of user interface [35].

- Public space: Where everything is audible and visible to all participants at all times.
- Public space + Private space: Participants are able to create or place modules in their own private space. The content in a participant's private space cannot be seen, edited or heard by other users.
- Public space + Personal space: Participants can view and hear other user's personal space.

The analysis of the result from this experiment is noteworthy. The degree of engagement is substantially higher when users have their own private or personal space. Users created 50% more ideas and spent 20% more times on refining their ideas when the private space is given. We can use this psychological device to be more productive and inspiring in the artistic group work.

> "Given the extent to which participants exploited the private and personal spaces to formulate contributions, and the role this played in shaping the

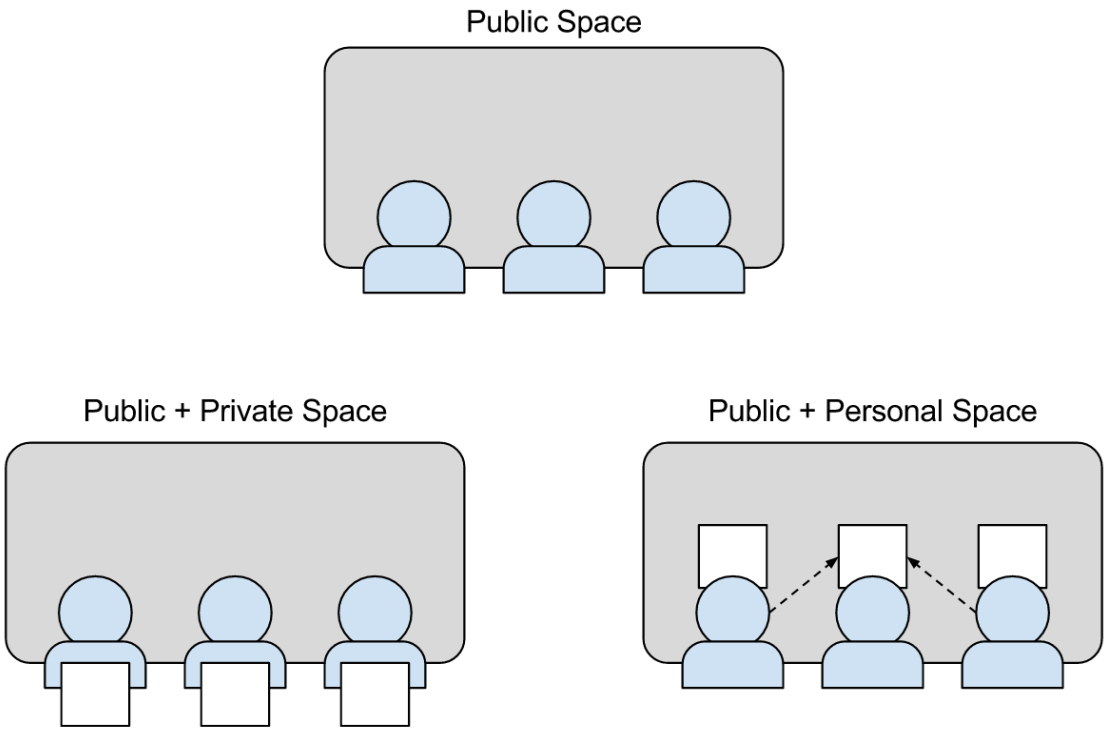

Figure 2.10: Various Privacy Modes [35]

> way groups collaborated, we argue that this is a key design consideration [35]."

To sum up, the awareness is the key element of computer-supported group work. This is even more true to CMM. The interaction between collaborators in CMM is greatly affected by the granularity of the system-imposed synchronicity. The higher synchronicity makes awareness information more useful and that helps collaborators to be more creative and inspired. This mutual recursion eventually benefit everyone involved in the interaction. I argue that this is a great example of "beyond musicking there".

Secondly, the scope of exposure on awareness information is something to consider. Several studies indicate the importance of the privacy in the artistic group work. The mode of interaction in CMM needs to be carefully engineered by taking into account of the awareness information and the privacy.

2.8 Challenge and Opportunity

Creating a thriving CMM platform is not an easy job. Let alone previously mentioned challenges in musicking, it comes with all the obstacles that are commonly found in the CSCW systems. According to Luther and et. al, the open source movement has been an exemplary CSCW project, but only 20% of projects is considered successful. They also develops criteria for success from multiple case studies on web-based collaboration [73].

- Planning & Structure: Collaborations with initial planning and structure, especially technical specifications, are more likely to be successful.
- Reputation & Experience: Creators who are well-known in the community or have experience in group work are more likely to lead successful collaboration.
- Communication & Dedication: Collaborations whose members or leaders frequently communicate are more likely to be successful.

I argue that the common denominator of these factors is the strong engagement between coworkers. The most obvious way to achieve it is to offer seamless user experience across a variety of devices allowing users to engage with collaborators regardless of time and place. Candy and Edmonds also discovered interesting ideas through the close observation about the creative collaboration between an artist and a technologist [16]. According to their study, supporting a creative group work by assisting collaborators to have strong partnership is the key to the successful collaboration.

> "In particular, there is a need to pay attention to the individual artist's requirements for collaboration as well as the context in which partnerships may flourish. It was clear that developing a partnership, as distinct from having an assistant relationship was a significant factor in the success of the collaboration between artists and technologist. ... *(omitted)* ... In other words, to be successful, creative partnerships needed appropriate organizational support [16]."

If the provision of the high quality awareness information is the primary support,

this notion of organizational support is the secondary, which is about fostering the partnership through the strong engagement and the effective communication. Again, this falls into the research area of collaborative user experience engineering and it definitely needs more attention in the future.

2.9 Summary

In this chapter, I reviewed various literature from the CSCW (Computer-Supported Collaborative Work) and NIME (New Interfaces for Musical Expression) research communities. Dahl's concept of NIME research being a 'wicked problem' outlines the scope and the direction of this thesis. I proposed Delay and Distance model specifically tailored to CMM (Computer-Mediated Musicking) with finer granularity on both time and space scales. Based on the enhanced classification framework, I defined awareness information and privacy as the primary support from the collaboration system. CMM also can be improved by the provision of secondary aids such as organizational support.

Chapter 3

Browser, The New Sound Machinery

The preamble of this chapter describes the current state of web music technology and how we can make the most of it. The review of the context elucidates why the web platform is ideal for computer musicking. In the second half of the chapter, I present a software framework that facilitates the development of web music application by incorporating many ideas introduced in the thesis.

3.1 Omnipresence and Immediacy of the Web

Since its beginning, the Web has been the window of the internet. Although the advent of mobile devices had a great impact on how people access the information on the internet, the Web is still favorable in terms of the omnipresence and the

immediacy.

Omnipresence

Functioning across platforms is among the greatest advantages of the Web. The web browser is available for various devices or different operating systems. Thanks to this universality of the web browser, accessing the Web is a basic feature for any platform thus users do not have to compile, buy, download or install it manually. The slogan like "write once and run everywhere" highlights the cross-platform strength of the web platform.

Immediacy

The other edge is the instantaneous nature of the Web. Sharing a URL is a battle-tested method of sharing information over the last decades. Clicking a link on the browser instantly opens a deluge of information. The modern web application abuses this swiftness. We mindlessly click countless URLs a day and we do not try to comprehend or decipher them anymore. It just works. The way of using URLs has been widely transformed from the early days, but the degree of immediacy remain unchanged. Only the world of the Web has gotten tremendously bigger.

Discoverability

These two brilliant qualities result in another advantage; discoverability. The modern search engines are restlessly crawling the enormous amount of web pages, thus the web presence is essential for any kind of business. Through the search engine, web pages

have almost unfathomable reach to the audience at large. Discoverability within the mobile app is far inferior to the web page. Furthermore, the design of modern web pages takes search engine crawlers into account by crafting pages with more relevant information, in turn, enhancing discoverability of web-based applications.

Freedom and Openness

Another advantage of the Web over native/mobile application ecosystem is that there is no dictatorial control over content, and thus no censorship. There is no single centralized software marketplace. The Web has been driven by the community and by industry competition since its birth. Also the specification and the standard around the web technology is publicly open and anyone can participate in the discussion. Anyone can publish a page into the public space and this makes the web platform even more viable for experimental use cases such as audience participation.

3.2 New Browser Technology

Although the web platform has multiple advantages, it is not perfect. The web-based software running on top of the browser has several disadvantages compared to the native one (i.e. an application built with Java or C/C++) which is highly optimized for its purpose. However, W3C (World Wide Web Consortium) and WHATWG (Web Hypertext Application Technology Working Group) has been consistently expanding the browser's capability to transform the web browser to the solid bedrock for competent application. This section reviews several cutting-edge web technologies that

make the browser musically capable.

3.2.1 Web APIs for Computer Music

The idea of Web Audio API was originally proposed by Chris Rogers around 2010 [111]. The primary objective of API is to enable the browser for process and synthesis audio programmatically. The official specification work started around 2011 by W3C Audio Working Group. Although user can dynamically form a graph with given building blocks, the direct manipulation of audio stream is limited by design. The API abstracts the underlying implementation of audio processing away from the web developer thus it is considered a high-level audio API.

> "This specification describes a high-level JavaScript API for processing and synthesizing audio in web applications. The primary paradigm is of an audio routing graph, where a number of AudioNode objects are connected together to define the overall audio rendering. The actual processing will primarily take place in the underlying implementation (typically optimized Assembly/C/ C++ code), but direct JavaScript processing and synthesis is also supported [107]."

Writing dynamic web music software has never been possible without using a third-party plugin like Flash. By using Web Audio API, now web authors can create a music application such as synthesizer, sequencer and effect processor on the browser.

3.3 Powerful Graphics and JavaScript Engine

Flexible and optimally responsive graphics is crucial in building a user interface that is free of discontinuities and jerkiness in the display. This is where the web platform shines. It offers multiple graphics APIs for various purposes: SVG [105], 2D Canvas [58] and WebGL [69]. These technologies are thoroughly tested over years and numerous powerful libraries built on top of them are also freely available. The developers can easily utilize them when they want to build graphics function or UI for the web application.

In addition to that, the remarkable performance improvement of JavaScript engines in the browser changes the landscape of the web development. The execution speed of JavaScript code has gotten to the level where it is fast enough for common applications like word processor, spreadsheet, mail client, and calendar. Naturally the online service like Google Drive was emerged and it has proven its worthiness by hundreds of millions active users [42].

Along with the progress of the engine performance, the new directions are being explored to bring the speed of native code into the browser. *asm.js*, proposed by Mozilla, realized the idea of 'JavaScript as compile target' from the C/C++ code. This is possible by a tool chain that converts the C/C++ source code to the *asm.js* code, which is a heavily restricted subset of JavaScript [33]. A benchmark showed an *asm.js* code compiled from C/C++ code can reach 70% of native speed [4]. *asm.js* in conjunction with WebGL is the key element of recent advancements in real-time 3D graphics project such as the web-based version of Unity and Unreal Engine 4 [29] [24].

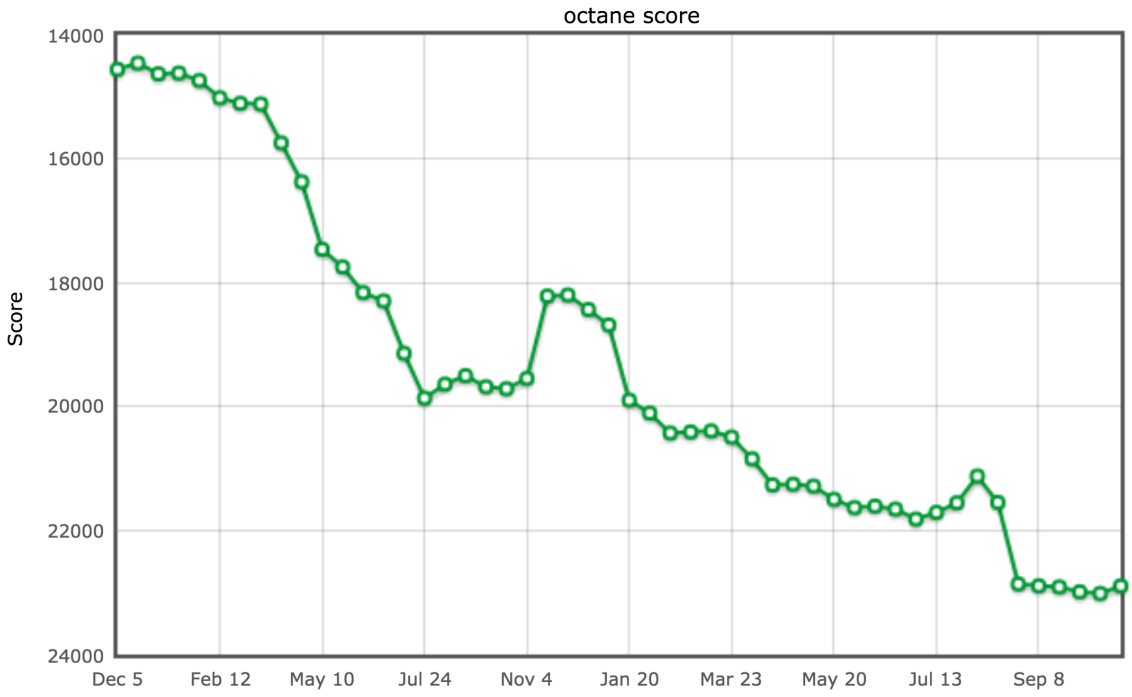

Figure 3.1: Octane benchmark score of Chrome V8 JavaScript engine between 2012 and 2014 [6]

After the successful debut of *asm.js*, a larger scale effort to run the optimized JavaScript code on the browser is also on the way. WebAssembly is a collective proposal supported by Apple, Google, Microsoft and Mozilla [31]. The main difference from *asm.js* is that WebAssembly parses source codes into a binary blob which is compatible across browsers thus no script parsing is required afterward. This is considered the next step of *asm.js* and is targeting computationally intensive use cases such as sound and video processing [12].

3.4 Future of Web Applications

Web Components are the fundamental change on how we build a web application. This set of new web technologies allows developers to define their own reusable HTML element by extending the existing HTML vocabulary. This is an innovation stems from the very idea of extensibility and the modularity. Along with other Model-View-Controller (MVC) frameworks for the web development, Web Components as the web standard will simplify the development and the maintenance.

Remarkable progresses have been made since the introduction of the Web Components in 2011. All major browsers have started implementing the underlying technology needed to run Web Components natively [88]. While browser vendors are still working on the native implementation, polyfills have been assisting to make Web Components available to developers [53]. Polyfill is additional JavaScript code which provides facilities that are not built into a web browser. It implements technology that a developer expects the browser to provide natively, providing a more uniform

API landscape.

The web application is bound to an HTML page by nature. When a browser window refreshes or closes, user loses all the states with it. This has been an obstacle for the Web to be a competent application platform. In order to have the persistence, the specification for Service Worker is currently being drafted at the time of writing [92]. A Service Worker runs on a dedicated thread separated from the main thread and it does not need to be attached to a web page or user-facing elements [36]. With Service Worker, the web application can have the 'app-like' user experience; a main application controller maintains user's states while navigating multiple pages. It also can cache resources locally so user can access the page without an active network connection.

3.5 Web Audio API

This section presents a detailed and in-depth discussion of several facets of the Web Audio API. Since it is a part of the W3C standard documents, it is beneficial to understand transitional process and the current status as a web standard. Subsequently The fundamental concept of API and the implementation detail of Chrome browser are investigated. Lastly, I offer an observation of the advantages and disadvantages of Web Audio API.

Specification	FPWD	LC	RC	PR	Rec
Web Audio API	Dec 2011	Q2 2015	Q4 2015	Q1 2016	Q1 2016
Web MIDI API	Q3 2012	Q2 2015	Q4 2015	Q1 2016	Q1 2016

Table 3.1: W3C milestones of Web Audio API Specification [9]

3.5.1 W3C Specification and Browser Support

It is important to underscore that Web Audio API is merely an API specification and not the implementation detail. This abstraction is intentional so that multiple browser vendors can compete in the engineering effort. The W3C specification document is public including all relevant discussions within working groups.

Flattening out every detail and corner case often takes years to finish. From the beginning to its end, a W3C specification document goes through different transitional status [106]. The current status of Web Audio API is Editor's Working Draft (WD) and is expected to be Last Call (LC) at the third quarter of 2016. The maturity of the specification is considered young and some parts of the document are still in flux.

The browser implementation is independent to the transitional status of document. Any browser vendor can build their version based on the unstable specification. At the time of writing, Web Audio API is implemented all the major web browsers including Chrome, Edge, FireFox and Safari. Although there is an issue of interoperability, Web Audio API is generally accepted as a solid specification for its purpose and pursued by major browser vendors including Apple, Google, Microsoft and Mozilla.

3.6 Web Audio Fundamentals: Context, Nodes and Parameters

After the birth of the Music-N computer music languages in the 1950s, there have been several attempts to innovate how to program the computer-generated sound. Some of earlier descent are statically-compiled (e.g. CSound, CLM and STK) and some of recent efforts are dynamic and interactive (e.g. SuperCollider and ChucK). From the language perspective, Web Audio API is comparable to the latter because it is JavaScript which is known to be highly dynamic, but it also inherits many traditional elements such as UGen (unit generator) and the different control rate (a-rate or k-rate) from Music-N or CSound.

In Web Audio API, AudioNode corresponds to the unit generator. It also owns a set of AudioParam objects that represents controllable parameters. Similar to other graphics APIs in the browser, an AudioContext object must be created to initiate the audio processing. The context itself can also perform system-wide functions such as audio file decoding or thread control.

AudioNode is a building block that can be connected to form a graph. The underlying audio engine traverses the graph to produce sound. The following is the list of available nodes with a brief description.

- AnalyserNode: provides real-time frequency and time-domain analysis information.
- BiquadFilterNode: represents a biquad filter, convenient parameter control with

```
// Create an audio context.
var context = new AudioContext();

// Create a gain node from the context.
var amplifier = context.createGain();

// Adjust gain parameter.
amplifier.gain.value = 0.5;

// Stop the context and end audio processing.
context.close();
```

Listing 1: AudioContext and AudioNode

frequency, Q, and gain.

- ChannelMergerNode: combines channels from multiple audio streams into a single audio stream.
- ChannelSplitterNode: allows to access the individual channels of an audio stream in the routing graph.
- ConvolverNode: applies a linear convolution effect given an impulse response.
- DelayNode: delays the incoming audio signal by a certain amount.
- DynamicsCompressorNode: implements dynamics compression effect.
- GainNode: changes the gain of an audio signal.
- IIRFilterNode: implements higher-order filters with explicit coefficients.
- OscillatorNode: an audio source generating a periodic waveform.

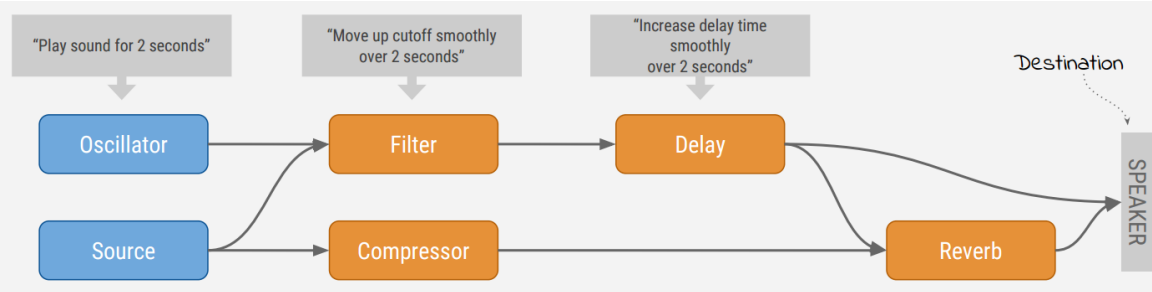

Figure 3.2: Connection and parameter control

- PannerNode: positions or spatializes an incoming audio stream in three-dimensional space.
- ScriptProcessorNode: generates, processes, or analyses audio directly using JavaScript.
- StereoPannerNode: positions an incoming audio stream in a stereo image using a low-cost equal-power panning algorithm.
- WaveShaperNode: implements non-linear distortion effects.

The figure 3.2 and the code listing 2 is the demonstration of how Web Audio API can be programmed to create a simple synthesizer.

3.7 Processing Mechanism

As stated in the specification, high quality audio processing relies on a dedicated thread that is independent to the main JavaScript thread of the browser. The document recommends for this thread to have a high priority in the operating system's

```
// Create an audio context.
var context = new AudioContext();

// Create AudioNodes.
var osc = context.createOscillator();
var samp = context.createBufferSource();
var filter = context.createBiquadFilter();
var comp = context.createDynamicsCompressor();
var delay = context.createDelay();
var reverb = context.createConvolver();

// Make connections.
osc.connect(filter).connect(delay).connect(reverb);
samp.connect(filter);
samp.connect(comp).connect(reverb);
delay.connect(context.destination);
reverb.connect(context.destination);

// Schedule sound production and parameter change.
osc.start(0.0);
osc.stop(2.0);
filter.frequency.setValueAtTime(440, 0.0);
filter.frequency.linearRampToValueAtTime(1000, 2.0);
delay.delayTime.setValueAtTime(0.25, 0.0);
delay.delayTime.linearRampToValueAtTime(1.0, 2.0);
```

Listing 2: The example code for figure 3.2

thread manager, otherwise it can cause undesirable side effects in the audio stream such as glitches or audio dropout.

The entire audio processing happens outside of the control (main) thread, thus the control thread is solely used to issue the instruction to the audio thread. This is the very intention of the design because the browser's main thread is often busy with other tasks thus not suitable for computationally intensive operation such as real-time audio processing.

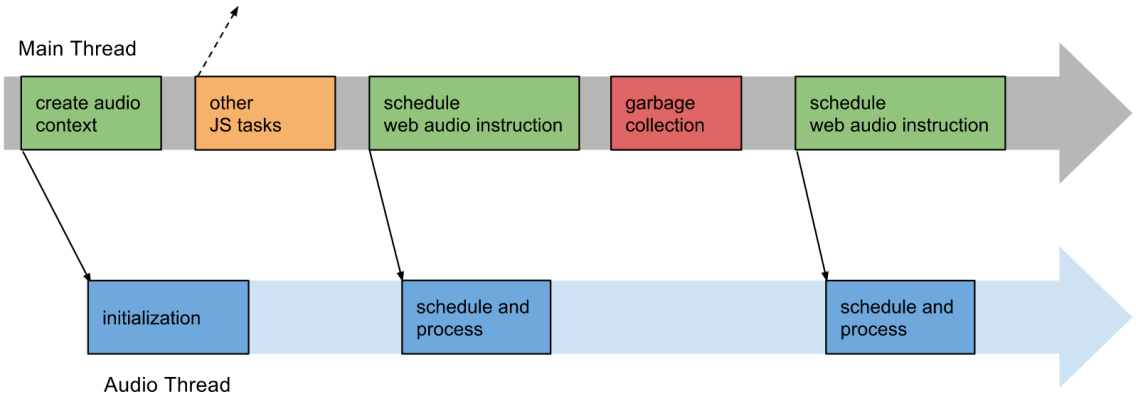

Figure 3.3: Web Audio API's multi-thread architecture

Note that issuing instructions to the audio thread is asynchronous operation. The browser infrastructure will do its best to process the instruction in time, but it does not guarantee the change to be sample-accurate. This is the reason why Web Audio API has sophisticated sample-accurate scheduling functions. If user schedules the instruction ahead of the expected timing, the sample accuracy of the scheduled event is guaranteed. For example, assigning a value to a certain attribute is not sample accurate. When the sample accuracy is required, the value assignment needs to be scheduled with a method.

```
1 // not sample accurate. |gain| value will be 0.5
2 // as quickly as possible.
3 gainNode.gain.value = 0.5;
4
5 // sample accurate. |gain| value will be 0.5 at the specified time.
6 gainNode.gain.setValueAtTime(0.5, 2);
```

Listing 3: Parameter change by setter and scheduled parameter change

As previously mentioned, the modern JS engine is reasonably fast so there might

not be any noticeable difference between these two operations, but the timing problem grows quickly when handling multiple AudioNodes and AudioParams.

Also the specification implies Web Audio API's engine is processing audio stream as a block of 128 samples. This 'render quantum' plays a very important role in Web Audio API's internal mechanism. The duration of render quantum defines the highest precision of user-imposed control change (so-called *k-rate* change) and only scheduled changes can happen within the render quantum. Making connection or assigning a value to the parameter during the ongoing render quantum will happen in the next render quantum.

```
1  // Create non-real-time WebAudio renderer with:
2  // 1 channel, 44100 samples-long and 44100Hz sample rate.
3  var context = new OfflineAudioContext(1, 44100, 44100);
4
5  var osc = context.createOscillator();
6  osc.connect(context.destination);
7  osc.start();
8
9  context.startRendering().then(function (buffer) {
10   // do something with the rendered buffer.
11 });
```

Listing 4: Non-real-time rendering with OfflineAudioContext

Web Audio API also supports the non-real-time audio rendering, which renders the result of an audio graph as fast as the machine can. The feature is useful when testing applications in an automated setting or the bandwidth of a graph is too large to be played in real-time. (i.e. too many AudioNodes)

3.8 Strength and Weakness

Web Audio API implementation has several advantages over other music programming platforms.

- **Dedicated audio thread**: Web Audio API specifies the implementation should have a dedicated thread for processing audio. Because the browser's main thread is commonly busy with user-facing tasks such as processing DOM and UI, having a dedicated thread not only improves the integrity of the audio stream, but it also guarantees the smooth operation on the main thread. This is a must-have feature for the platform that targets the real world production.

- **JavaScript API**: At the time of this writing, JavaScript is one of the most popular programming languages available [71]. It is easy to pick up compared to strongly-typed languages like C++ or Java and also a full-featured language that is highly extensible and works on both server and client side. More importantly, the language specification (ECMAScript) and the script engine are constantly being improved by industry leaders [30].

- **Seamless integration with other web APIs**: From my perspective, this is the greatest advantage of Web Audio API over other music programming platforms. While others had to use a loose coupling like Open Sound Control in order to draw simple graphics, Web Audio API can be tightly layered/composited with other web APIs such as Canvas or WebGL which is a full-blown graphics platform.

- **Strong community and ecosystem**: The Web has been driven by the community since its birth and the mentality of openness is the biggest asset for the platform to thrive. There are numerous open-source projects for the web development and Web Audio API can be used together with any of them. Also the development setup and debugging tools are far superior to what the existing audio programming platforms have offered.

It also has several weaknesses, but most of them comes from the nature of the web platform.

- **Strict security policy**: In principle, the browser does (should) not allow the direct access to the local resources such as file system, camera and audio input device. Also its 'sand-boxing' execution model prevents any third-party software component (e.g. audio effect plug-in) from running on user's computer without an explicit permission [11]. Although we computer musicians have neglected the security and the user safety so far, now we need to be more cautious because the extensive accessibility of the Web comes with the price of security and safety.

- **Browser and platform incompatibility**: An implementation from each browser vendor cannot be identical because of the different software infrastructure. Due to this incompatibility, web authors often have to spend hours (or even more) to reconcile the disparity in their applications. Moreover, a browser running on various operating systems or devices may produce different results because of the difference in underlying system software. Polyfills are available to reduce the wasteful engineering effort [78].

- **Slow progress of standard and implementation**: Some problems need to be addressed in the native level (i.e. browser's source code) and the fix can take from weeks to years to propagate from the prototype to the stable version. Either of an implementation bug or a specification problem can drag the application development for a significant amount of time. Fortunately the progress and the discussion on ongoing issues are public, so developers can communicate with browser implementors and the W3C working group to adapt their plan accordingly [108].

3.9 WAAX: Web Audio API eXtension

This section presents the software framework for web audio application, which is the integral part of this thesis. Before reporting its design and implementation, I believe the rationale and the motivation of the project should be addressed.

3.9.1 Rationale

Although Web Audio API is fairly capable for the general audio processing purpose, several missing links were found over last two years of working on actual web audio projects.

Firstly, the incompatibility of Web Audio API implementations needs to be reconciled on the client side. This can be done by importing an additional JavaScript called polyfill. Technically it overrides a set of function calls to mitigate differences between browsers. Some disparities cannot be resolved in the script level, so the

purpose of polyfill is to establish a common ground that is reasonably compatible.

Also W3C Audio Working Group and the web audio community raised an issue of the extensibility on the spec [99]. For a web standard, being extensible is important to enable developers to create new features on top of built-in APIs [87]. However, web developers found that some parts of Web Audio API are hidden in the implementation indicating the limited extensibility upon AudioParam and AudioNode classes. For example, if one wants to create a new AudioNode-like class by compositing multiple built-in AudioNodes, it is impossible because the constructor of the class is not exposed. Therefore developers have to rely on a complex intermediate layer for such composition.

Web Audio API also lacks unit generators that are commonly available in other audio programming platforms and developers are responsible for creating custom unit generators by connecting built-in AudioNodes. This is the very intention of the API design, however, the community effort does not scale much further with the crippled extensibility.

Maintaining the sanity of a large-scale web project is also laborious. HTML and JavaScript are not designed with the well-defined paradigm like OOP (object-oriented programming) or MVC (model-view-controller) at first place. The code base quickly gets chaotic as the application grows. The lack of proper scaffolding like OOP or MVC makes developers waste hours on creating reliable user interface and application routing system, and eventually leads a project to fail due to the overwhelming maintenance cost. To that end, new frameworks like Angular, Polymer

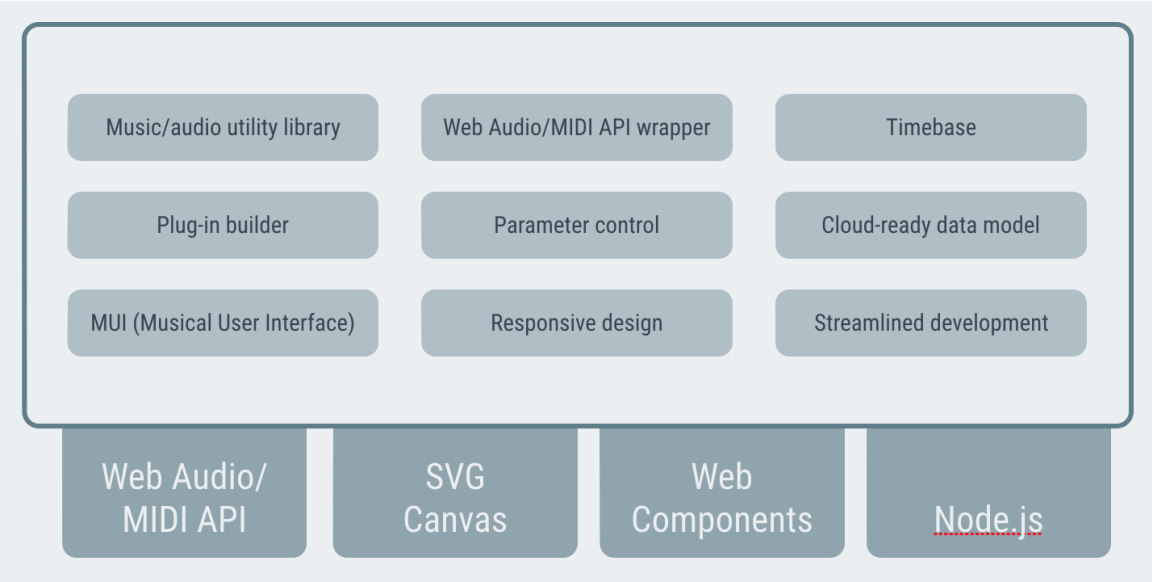

Figure 3.4: Layers and components in WAAX

and React were introduced recently and the web music app development can benefit from them as well [5, 83, 85].

Web developers also have to deal with the chaotic development setup. As of 2015, a text editor and a terminal window are still considered the primary setup for the majority of web developers, not to mention having to reload the browser by hand to preview changes. To reduce the manual labour and simplify the workflow, a variety of automation systems such as Grunt or Gulp became popular [46, 47]. Web developers are also actively working on the idea of better packaging, dependency management and deployment. Browserify and WebPack projects are the most recent progress to this end and it simplifies the development process significantly [15, 112].

All in all, we have opportunities to utilize these powerful apparatuses and also have several issues to resolve in our hand. My biggest motivation was to incorporate

essential elements into one package for the development of web music application, and WAAX (Web Audio API eXtension) is the result of such effort. It features:

- Cross-browser compatibility
- Extensible plugin system for Web Audio API
- A variety of custom unit generators
- Musically tailored UI elements based on Polymer library [83]
- Automated packaging and deployment based on Gulp task runner [47]

The primary objective of WAAX is to assist the real-world production of web music application, which is the point distinguishes this project from other music programming platforms commonly centered around the experimental computer music.

Another agenda of WAAX is to bring two opposite ends of the spectrum closer together: the audio developers who are not familiar with web technology, and the web developers who lack a background in computer music. I believe the accessibility of WAAX project is suitable for the pedagogical use case, so I designed a 2-day workshop on web music technology based on WAAX. The detailed report on the workshop can be found in the appendix of the thesis.

3.9.2 WAAX Core Library

WAAX core library contains fundamental features for other parts of the library to work properly. The first step of it starts with loading the polyfill to make it work regardless of the type of browser.

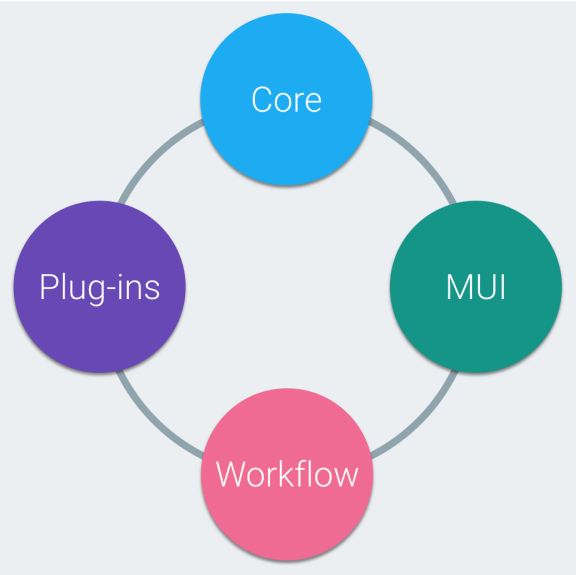

Figure 3.5: WAAX system overview

- Polyfill: WAAX library is optimized for Chrome, but this polyfill attempts to reconcile possible incompatibilities for FireFox and Safari. Also it masks out unavailable features in the browser.

- Music math helpers: useful math functions such as MIDI pitch to frequency, decibel to amplitude and more.

- AudioNode composition: addresses layering/composition issue of Web Audio API by providing extensible AudioNode plugin architecture.

- AudioParam utilities: provides terse method signatures for AudioParam scheduling.

- Timebase: provides abstraction of coordination between musical and real-time.

WAAX adds few convenient methods to the prototype of AudioNode and AudioParam. The following example shows the example usage of these helpers.

```
// Create an instance of WAAX. This creates an instance of
// AudioContext internally.
var WX = WAAX.create();

// Create multiple AudioNodes.
var osc = WX.OSC();
var gain1 = WX.Gain();
var gain2 = WX.Gain();
var gain3 = WX.Gain();
var gain4 = WX.Gain();

// Fan-out (parallel) connection: osc to gain1 and osc to gain2.
osc.to(gain1, gain2);

// Serial connection: osc to gain3, gain4 and then the master output.
osc.to(gain3).to(gain4).to(WX.master);

// Disconnect osc from everything.
osc.cut();
```

Listing 5: AudioNode construction and making connection with WAAX

Additionally Web Audio API's AudioParam methods are quite verbose. Sometimes it is cumbersome to type long method names repetitively. So WAAX abstracts them nicely into a compact and chainable form.

WAAX core library also offers numerous JavaScript object and math helpers. The following is non-exhaustive list of utilities. The complete API reference can be retrieved at the project web site [19].

```
// Create an oscillator node.
var osc = WX.OSC();

// Schedule parameter change: set to 440 at 0.0 second,
// linear ramp to 880 up to 0.25 second, and then slew
// back to 440 between 0.25 and 0.75 second.
osc.frequency
    .step(440, 0.0, 0)
    .line(880, 0.25, 1)
    .slew(440, [0.25, 0.75], 3);
```

Listing 6: AudioParam control with WAAX

3.9.3 Plugins

As previously mentioned, a limited range of built-in AudioNodes has been a pain point for developers who are not familiar with audio processing. A collection of WAAX plugins is designed to meet those demands. They are almost identical to the unit generator in ChucK or STK so you can chain them to create a signal path. However, the plugin is more extensive in scope and scale because it is comprised of multiple synthesis/processing building blocks.

WAAX categorizes plugins into 3 types: Generator, Processor and Analyzer.

Sampler, Synthesizer or a simple oscillator fall into the generator type. An audio effect such as a reverberation or chorus is a processor. The analyzer types are used for the visualization from FFT or waveform data.

To make plugin authoring easier, Plugin builder in WAAX core library offers a straight-forward procedure so developers can follow the guideline of Web Audio Plugin (WAPL) proposal. In that way, they can just focus on the black box inside of the plugin as shown in Figure 3.6. The following section describes how to write a WAAX

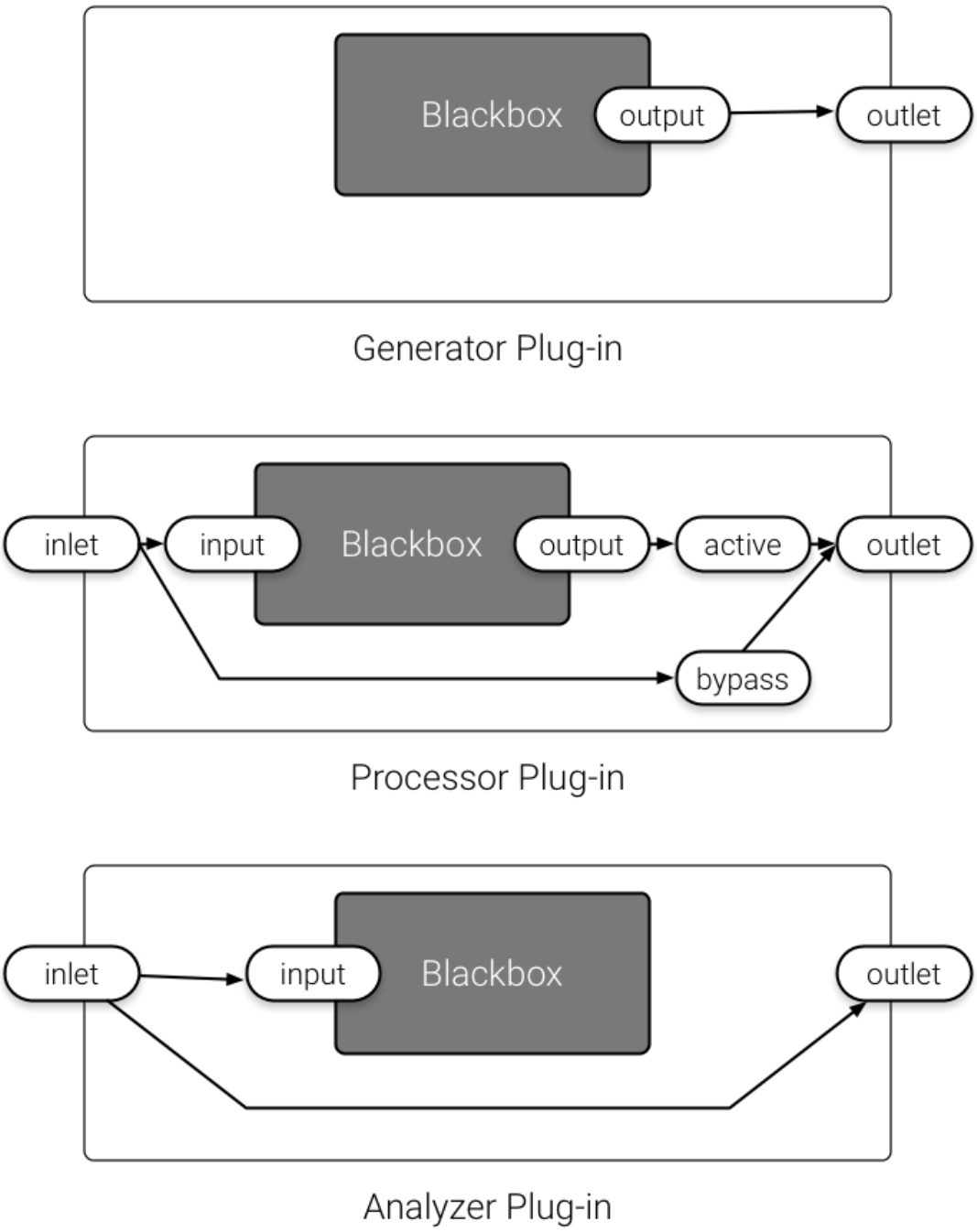

Figure 3.6: 3 types of plugin in WAAX

plugin.

Step 1: IIFE Encapsulation and Class Definition

Listing 7 shows a fundamental scaffolding of WAAX plugin. For the safe encapsulation, using IIFE (Immediately-Invoked Function Expression) is strongly recommended [63]. All in all, this is simply a definition of JavaScript class.

```
(function (WX) {

  // Plugin constructor (2)
  function MyPlugin (preset) { /* see step 2 */ }

  // Prototype (3)
  MyPlugin.prototype = { /* see step 3 */ };

  // Finalizing Plugin (4)
  WX.Plugin.extendPrototype(MyPlugin, 'Processor');
  WX.Plugin.register(MyPlugin);

})(WX);
```

Listing 7: Step 1 - WAAX plugin class definition in IIFE pattern

Step 2: Plugin Constructor

Among substeps in Listing 8, *(a)*, *(c)* and *(d)* are mandatory for the plugin to run properly. The substep *(a)* is critical because it creates the internal routing between the input or the output. In the current version, you can choose one plug-in type from Generator, Processor and Analyzer.

The next step is to define WAAX parameters and note that you need to define AudioParam handlers in the prototype.

```
function MyPlugin(preset) {

  // (a) define plug-in type
  WX.Plugin.defineType(this, 'Processor');

  // (b) create and patch native nodes
  this._input.to(this._output);

  // (c) define WAAX parameters
  WX.defineParams(this, {
    sweet: {
      type: 'Boolean', default: false
    },
  });

  // (d) initialize preset
  WX.Plugin.initPreset(this, preset);
}
```

Listing 8: Step 2 - Definition of plugin constructor

Step 3: Plugin prototype

A plugin prototype consists of plugin information, the default preset and parameter handlers. In Listing 9, the substep (a) and (b) are straightforward. Writing a parameter handler is also intuitive as long as there is a scheme for the parameter scaling or mapping.

Step 4: Finalizing Plugin

The final step is declaring of plugin type and injecting the plugin prototype to the custom definition. (See Listing 10) Also the last line registers the custom plugin into the WAAX namespace, so it can be instantiated from an WAAX object.

```
MyPlugin.prototype = {
  // (a) plug-in info
  info: {
    name: 'MyPlugin',
    api_version: '1.0.0',
    plugin_version: '1.0.0',
    author: 'Hongchan Choi',
    type: 'Processor',
    description: 'Dummy Plug-in'
  },

  // (b) default preset
  defaultPreset: {
    sweet: false
  },

  // (c) define handlers
  $sweet: function (value, time, xtype) {
    // do something when parameter 'sweet' changed
  }
};
```

Listing 9: Step 3 - Definition of plugin prototype

Using plugin in an HTML page

Once a plugin definition is written in a file, it can be used with WAAX library in a regular HTML file. (See Listing 11)

Lastly, the following list is the plugin collection that WAAX currently offers.

- CMP1: a simple dynamics compressor based on built-in compressor node.
- Chorus: implements chorus effect by Jon Dattorro [27].
- ConVerb: a convolution reverberator based on built-in convolution node.
- EQ4: implements a standard 4-band parametric equalizer.

```
WX.Plugin.extendPrototype(MyPlugin, 'Processor');
WX.Plugin.register(MyPlugin);
```

Listing 10: Step 4 - Finalizing plugin definition

```
<script src="waax.min.js"></script>
<script src="MyPlugin.js"></script>
<script>
  var myplugin = WX.MyPlugin({ sweet: true });
  myplugin.set('sweet', false);
  myplugin.to(WX.Master);
</script>
```

Listing 11: Using WAAX plugin in HTML document

- FMK1: a simple FM synthesizer with 2 FM operators per voice.
- Fader: a fader plugin with volume control in decibel.
- FilterBank: 6 cascaded biquad filters with harmonic control.
- Impulse: impulse train generator with frequency control.
- Noise: noise generator with color control.
- SP1: a polyphonic PCM sampler.
- SimpleOsc: an example plugin.
- StereoDelay: implements ping-pong delay.
- WXS1: implements monophonic dual-oscillator subtractive synthesizer.

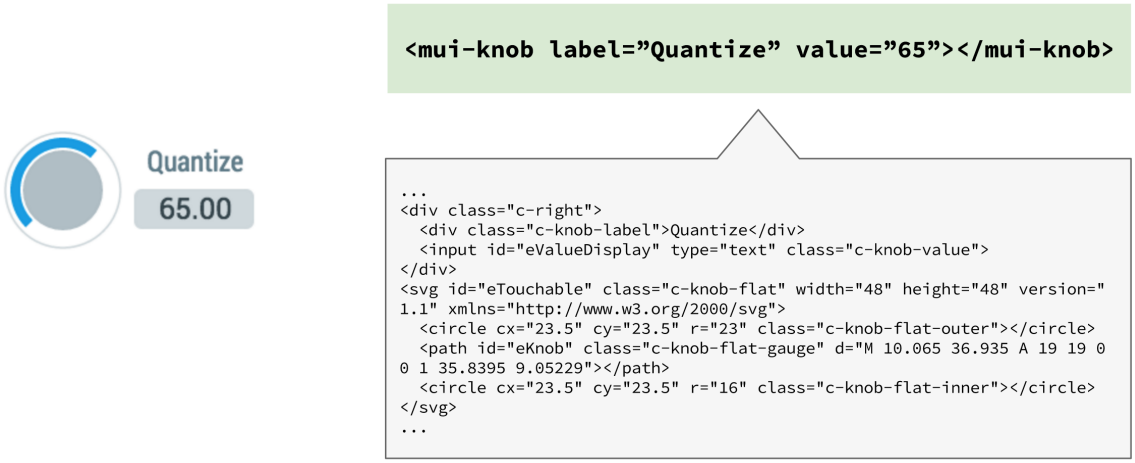

Figure 3.7: An example of $\langle mui - knob \rangle$ custom UI element

3.9.4 MUI Elements

Designing UI elements for music applications is fairly different from the common practice of application user interface. It should be functional, but it also needs to offer the sense of playability and rich information. This is where MUI elements come in. With MUI, you can create a musically-tailored UI element by writing a single line of HTML.

This level of encapsulation made possible by *Web Components* technology, which is comprised of 4 different APIs [45]. MUI elements are technically based on Polymer library to utilize Web Components [83]. MUI takes music GUI design to the next level. If a WAAX plugin has well-defined parameters, it will automatically build the entire control surface for it with a single line of code. In addition, some MUI elements are designed to be used in conjunction with the real-time hosted back-end so your knob can directly linked to the collaborator's.

MUI elements are highly modular and portable by the extensible nature of Web Components. I believe the technology has potential of changing how we build the music application fundamentally. It is also suitable for rapid prototyping and easily can be refined into the production seamlessly unlike other experimental music programming tools.

3.9.5 Enhanced Workflow

Without the proper tooling, the web development is full of repetitive manual labors. This tiny tool chain enormously boosts the productivity with the task automation and pre-configured templates. From bootstrapping a project to the server deployment, the majority of tasks can be scripted. WAAX's workflow is built on top of Gulp library.

```
1 git clone https://github.com/hoch/WAAX waax
2 cd waax
3 npm install
4 gulp
```

Listing 12: Installation of WAAX and launching a development server

As shown in Listing 12, it only takes 4 commands from the installation to launching a server. Although prerequisites like git, Node.js and Gulp must be installed on the machine, these are essential tools in the modern web development [41, 81].

3.10 Future Work

The unparalleled velocity of the web technology makes working on WAAX even more exciting. Over the last few months, this chapter has been constantly being rewritten

and updated. There are numerous things that I want to add to WAAX in the future, but I would like to discuss few upcoming features which have significant potential to the web music technology.

3.10.1 AudioWorkletNode

At the time of writing, the new proposal of AudioWorkletNode is under W3C Audio Working Group discussion [10]. The goal of AudioWorkletNode is to replace ScriptProcessorNode in the current specification. The group made a decision to deprecate ScriptProcessorNode because several flaws have been found in its design [8]. To make a long story short, AudioWorkletNode is an improved version of ScriptProcessorNode allowing to execute user-supplied JavaScript code on the audio thread. This feature will be the path for the custom DSP or synthesis code (i.e. sample-level audio manipulation) into the Web Audio engine.

This significantly simplifies the abstraction of custom audio processing in Web Audio API. Sharing a package of custom audio nodes between developers will become as simple as sharing VST plugins, but without the installation process. Once AudioWorkletNode reaches the stable version of each browser, WAAX will offer various unit generators that are available from PureData, STK or Faust in AudioWorkletNode form.

3.10.2 Progressive Web Application

Progressive Web Application is a hybrid approach that combines the best of the web and the best of apps [84]. This is perhaps one of the most important innovations in the web platform for the last decade. The very first visit in a browser tab, no installation is required. As the user progressively builds a relationship with the web application over time, it becomes more and more powerful.

Progressive Web Apps are:

- Responsive: Fit any form factor: desktop, mobile, tablet, or whatever is next.
- Connectivity independent: Enhanced with service workers to work offline or on low quality networks.
- App-like: Feel like an app to the user with application-style interactions and navigation because it is built on the app shell model.
- Fresh: Always up-to-date thanks to the service worker update process.
- Safe: Served via HTTPS to prevent snooping and ensure content has not been tampered with.
- Discoverable: Are identifiable as "applications" thanks to W3C manifests and service worker registration scope allowing search engines to find them.
- Re-engageable: Make re-engagement easy through features like push notifications.

- Installable: Allow users to keep applications on their home screen without the hassle of an app store.
- Linkable: Easily share via URL and not require complex installation.

What WAAX can do is to provide the "app shell" for music application based on the idea of progressive web app. Considering how important the local (client-side) experience is for music apps, this technology will transform the future of web-based music application.

Chapter 4

Case Study: Musicking on the Web

In previous two chapters I discussed two fundamental facets of computer-mediated musicking. The former described a new conceptual model, and the latter presented a new software framework that facilitates the development of music application on the browser. This chapter reviews 4 real-world projects that combine proposed ideas into a tangible form, and evaluates them from both theoretical and technical standpoints.

4.1 GridFlux

From the outset, the goal of GridFlux was to determine the extent to which web music technology can reach in comparison with native music applications. From a functional perspective, this project resembles (and is heavily inspired by) comprehensive rhythm programming tools such as the FXPansion's Geist or Native Instruments' Maschine system [39, 74]. This group of applications typically integrates a multi-voice sampler,

a high-resolution sequencer and various effect processors.

4.1.1 Key features and Design Overview

- 16 voices, multi-voice drum sampler. Each voice has various controls like:
  - AHR (attack-hold-release) envelope
  - 2-band equalizer
  - 3 send-return effects (chorus, delay, convolution reverb)
  - 1 nonlinear saturation effect and 1 master compressor.
- High-resolution MIDI sequencer
  - Supports 480 ppq (pulse per quarter note) resolution.
  - Variable quantize, swing and tempo control.
  - Features recording via Web MIDI API.
- Cloud-storage for loading and storing rhythm patterns.
  - Firebase real-time hosted service.
  - Supports per-pattern collaboration.

4.1.2 Discussion

As this project aimed to replicate a conventional rhythm workstation, the implementation of a real-time collaborative support was not considered. For this reason, the

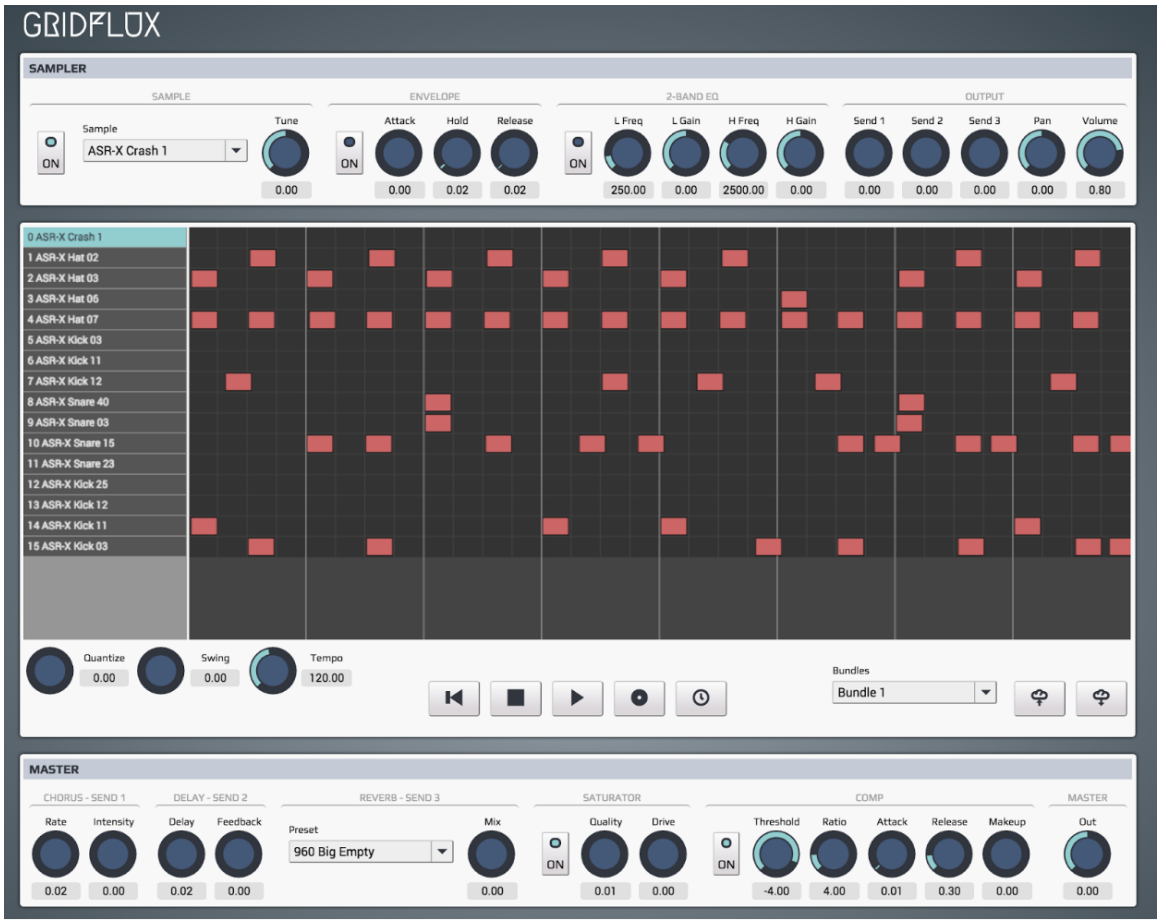

Figure 4.1: GridFlux, a feature-complete rhythm workstation on the Web

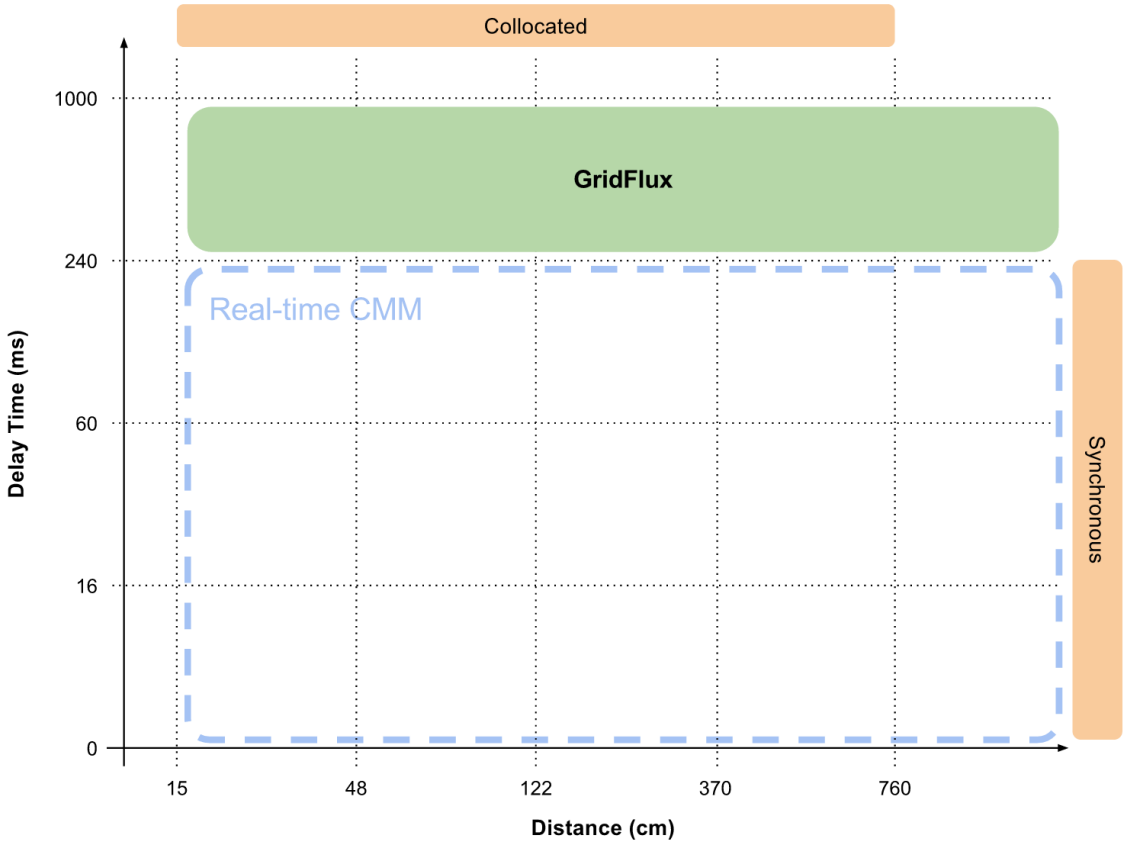

Figure 4.2: Collaboration model of GridFlux

collaboration model of GridFlux is rather old-fashioned. It operates on a 'per-file' basis and the change in a client does not upstream to the server automatically.

However, being able to share patterns without leaving the application turned out to be intuitive and effective. The web-based music collaboration services like SoundMondo and Allihoopa are also using per-file based collaboration system to support sharing of large-scale music material [97, 75]. It is a proven way of online music collaboration.

The non-trivial effort was invested to implement a high-resolution sequencer. Unlike numerous sequencer applications on the Web, GridFlux supports the resolution of 480 ppq (pulse per quarter note), which is the norm of professional MIDI sequencers. It is imperative to have this level of precision for features like variable swing and quantization. In addition, the subtle nuance in the rhythm pattern also can be expressed by pulling or pushing the note onset by a couple of ticks as in the professional sequencer.

From the software engineering perspective, early stage development was hampered by the lack of proper MVC and OOP frameworks, but it quickly stabilized after the adoption of the Polymer library.

Another challenge was to maintain a coherent association between a synthesizer parameter and a UI element. For the sake of rapid prototyping, I used the 'two-way binding' technique to interconnect multiple synthesizer parameters and UI elements (Figure 4.3) [26].

This technique became wildly popular after the introduction from celebrated JavaScript frameworks such as Angular and Ember [5, 32]. However, the native support for implicit data binding will be deprecated in favor of the explicit propagation due to several issues [70]. Thus, the feature in GridFlux needs to be modified accordingly in the near future.

GridFlux is a showcase of how a competent music application can be built using web technology. In addition, WAAX and MUI elements played a crucial role by saving a considerable amount of effort and time. Lastly, the per-file basis collaboration

Figure 4.3: An example of data binding with MUI elements and synth parameters

is a sensible design choice for certain styles of musicking, and I argue the pattern programming falls into that category.

4.2 Envvy

The analog monophonic synthesizer is well known for its strength in bass or lead sound in the electronic ensemble. Roland's TB-303 pioneered the genre of the 'patternized bassline' by attaching a simple sequencer to a monophonic synthesizer and it has been a popular way of programming synth bass [100]. With a twist of instant collaboration, Envvy extends the mentality of pattern programming to the Web.

4.2.1 Key Features

- 1-voice subtractive synthesizer with 1 white noise generator
- 8-step sequencer with individual envelope for each step
- 6 controllable parameters: pitch, pitch modulation, noise gain, filter cutoff and filter resonance
- Synchronization across clients without a dedicated server
- Sharing a pattern via data-embedded URL

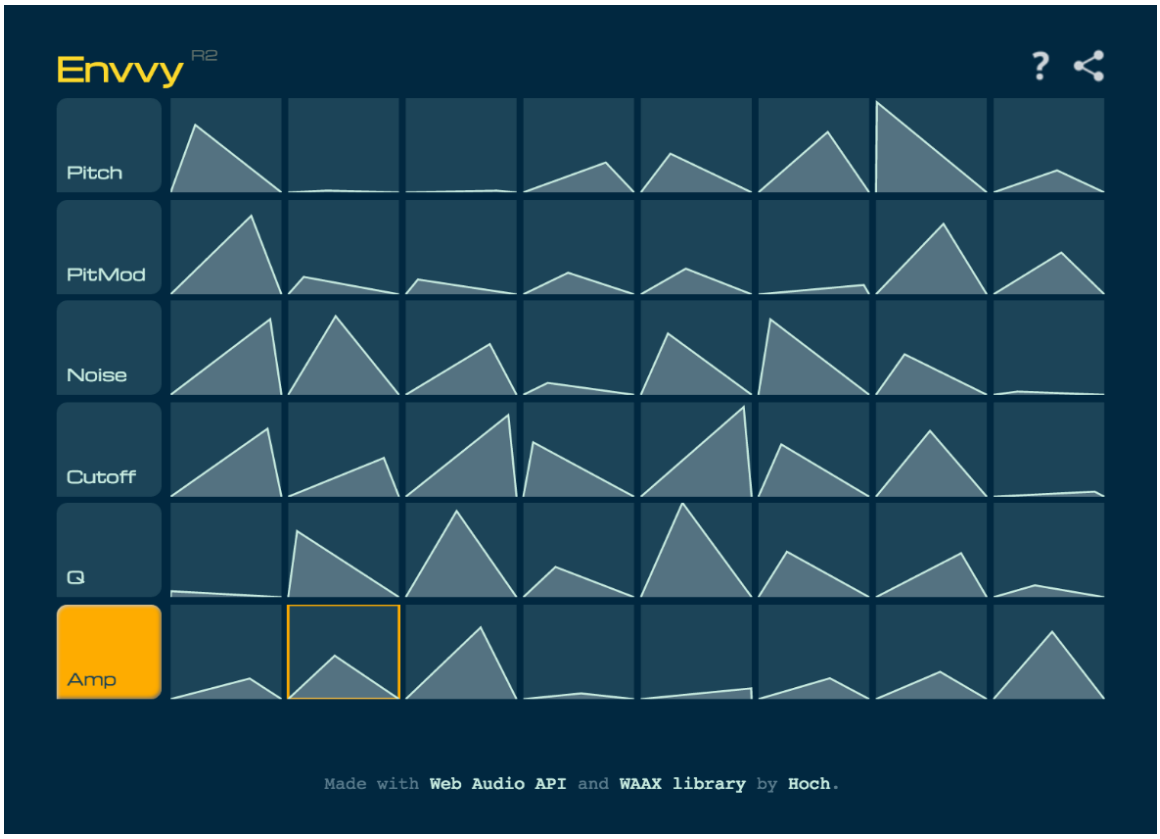

Figure 4.4: Envvy, a step-sequencing synthesizer

4.2.2 Discussion

Envvy's most prominent trait is that despite its visual resemblance to the typical step sequencer. However, each step has a fine-grained control by shaping the envelope inside and each row corresponds a parameter in the synthesizer. With handy features like pattern randomization and parameter morphing, Envvy has its own sense of playability.

Envvy supports co-located collaboration through instant synchronization, but it does not rely on a dedicated back-end or a central time server. This notion of 'world metronome' constitutes Envvy's inherent power.

Commonly, the synchronization between clients requires a clock server (e.g. a dedicated server broadcasting clock pulse via OSC), but Envvy synthesizes a time line which is derived from the system clock on the machine. When user visits the page, the application calculates the current position from the unix epoch origin (i.e. 00:00:00 UTC on 1 January 1970) in terms of the measure and wraps the measure around with the number of 8.

There are two big assumptions for the system to work properly; the tempo must be fixed and the material must be looped. Based on these two conditions, Envvy automatically locks all the participants into a unified time line and there is no extra cost of setting up a dedicated server or a clocking mechanism.

As shown Figure 4.6, the collaboration model of Envvy is notably unique. The idea of having a virtually synchronized clock across all the participants places Envvy into an inconceivable position on the plane. However, it is rather unfortunate people

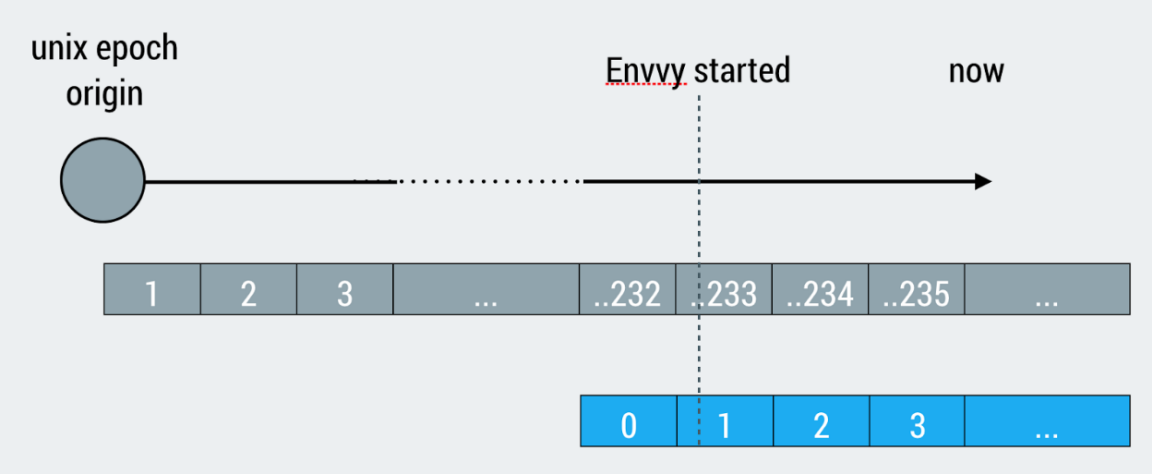

Figure 4.5: The notion of Ubiquitous Metronome in Envvy

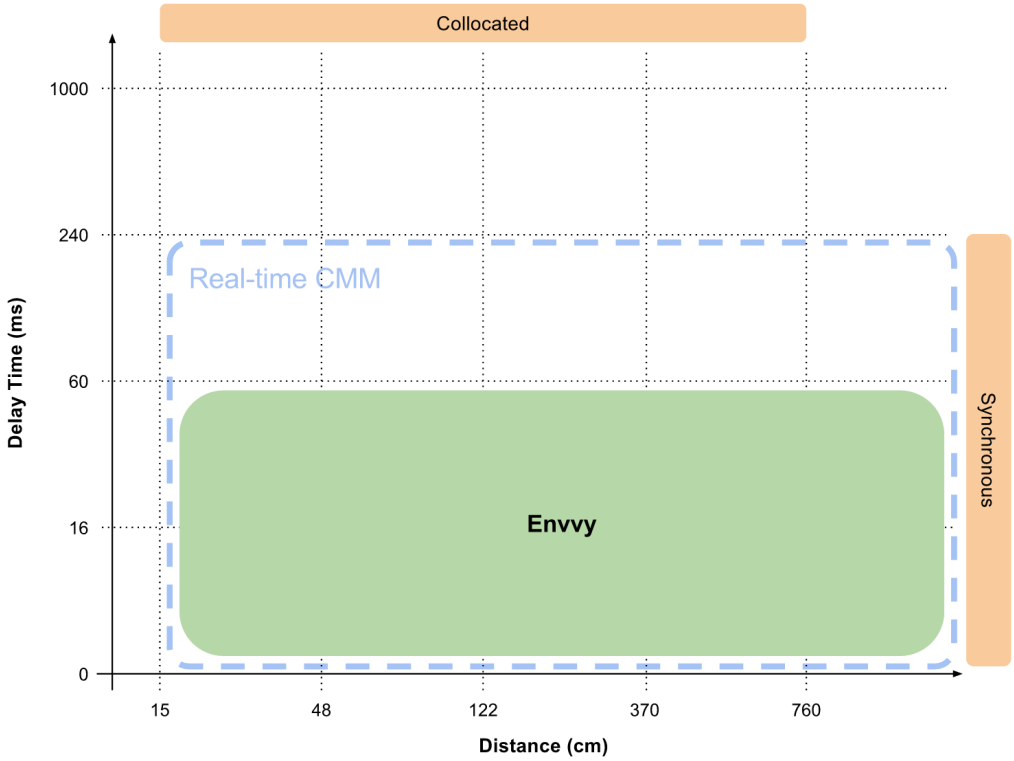

Figure 4.6: Collaboration model of Envvy

cannot share the state of collaborative work on the fly while the beat is synchronized. Again, the loose definition of 'synchronicity' in the model here fails to illustrate the nature of collaboration.

Another interesting bit is the synchronization between remote participants. Generally this type of synchronization does not present any advantage to expand upon, but it certainly worth investigating in the future. All in all, Envvy is a concise statement of simple user experience with the rich expressiveness and instant collaboration.

4.3 Music on Cloud

Music on Cloud is a highly experimental prototype to demonstrate how a collaborative music application can be built with WAAX and MUI elements, and Google Realtime API. This API provides developers with collaborative data model that is used in Google Docs [44]. Instead of a regular text document or a spread sheet, Music on Cloud works on a custom data model that represents musical content and multiple clients can edit the shared data concurrently.

Using Google's Realtime API offers several advantages over other services. The biggest strength was the integration with Google Drive. This includes the strong authentication, the real-time collaboration and the ample server-side storage. The authentication system allows user to have the superb control over the permission and the sharing. Furthermore, Google Drive's ecosystem makes it easy to interconnect with other Drive apps.

However, the prototype revealed several issues in this type of collaboration system.

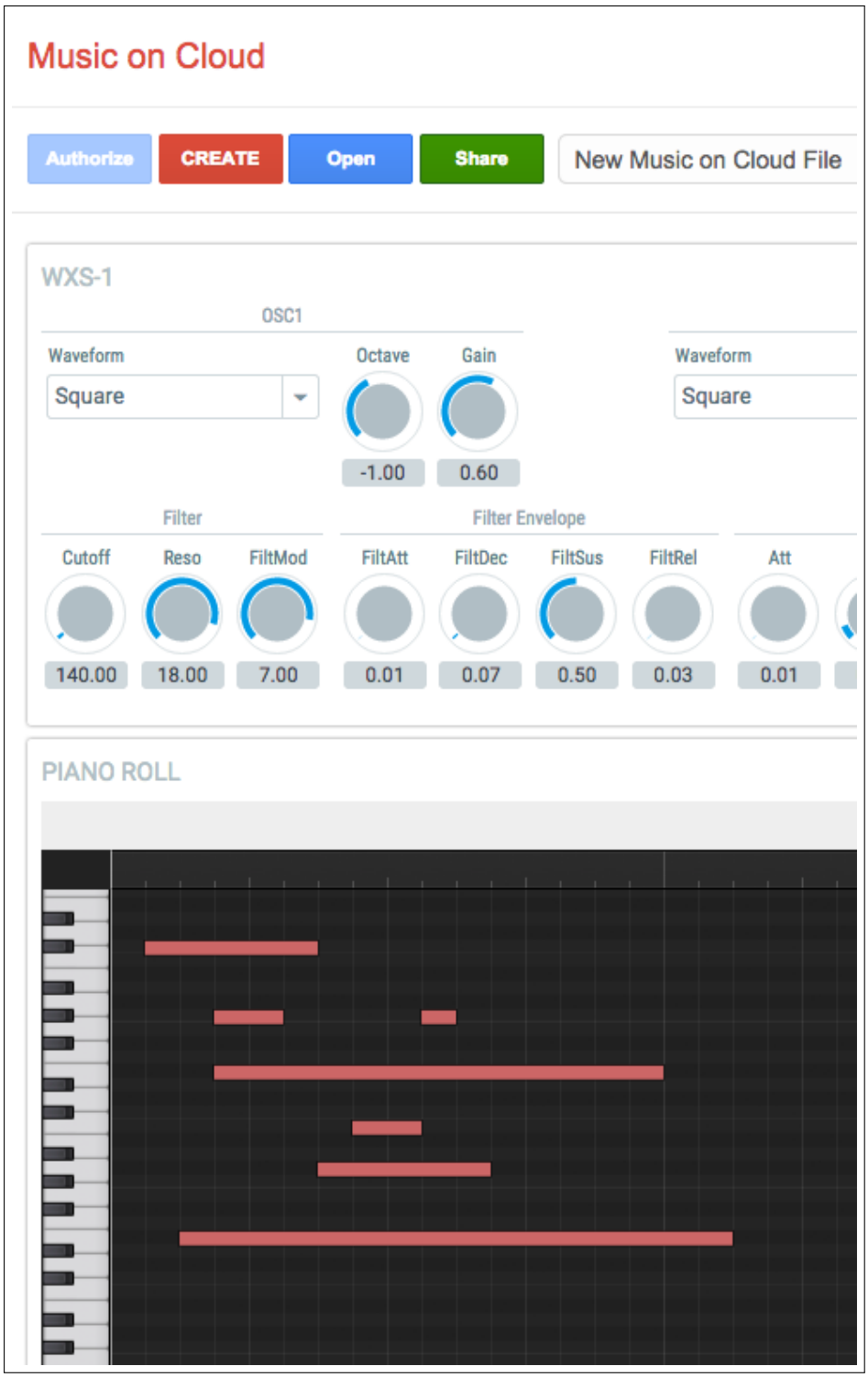

Figure 4.7: Music on Cloud

Firstly, the real-time collaboration services are tailored to the general purpose (e.g. text editing) where the relatively low synchronicity is acceptable. Thus the latency and the jitter introduced by the service are inadequate for CMMs below 60ms of delay.

Since the application entirely depends on the proprietary service, the room of optimization for musical purpose is almost non-existent. All the data transaction needs to go through the cloud back-end and it introduces the inevitable latency no matter how far or close participants are located.

Despite the lack of high synchronicity in the system, the ability to convey the sense of 'awareness' was notable. In this prototype, the awareness information was displayed by highlighting a UI element that is being controlled by other participants. I believe this opens a new area of interaction in CMM and the future study is expected to comprehend the meaning of this technique.

The collaboration profile of this system is not suitable for the performance-oriented musicking, but perfect for the production-oriented one. This is mainly because the system is free from the restriction of time and location, plus the communication channel goes together with the content itself.

4.4 Canopy

Programming computer-generated sound can be daunting due to the lack of proper debugging tools. Similarly, debugging audio software is even more convoluted, in that normally one needs to deal with hundreds of thousands of numbers bursting out of

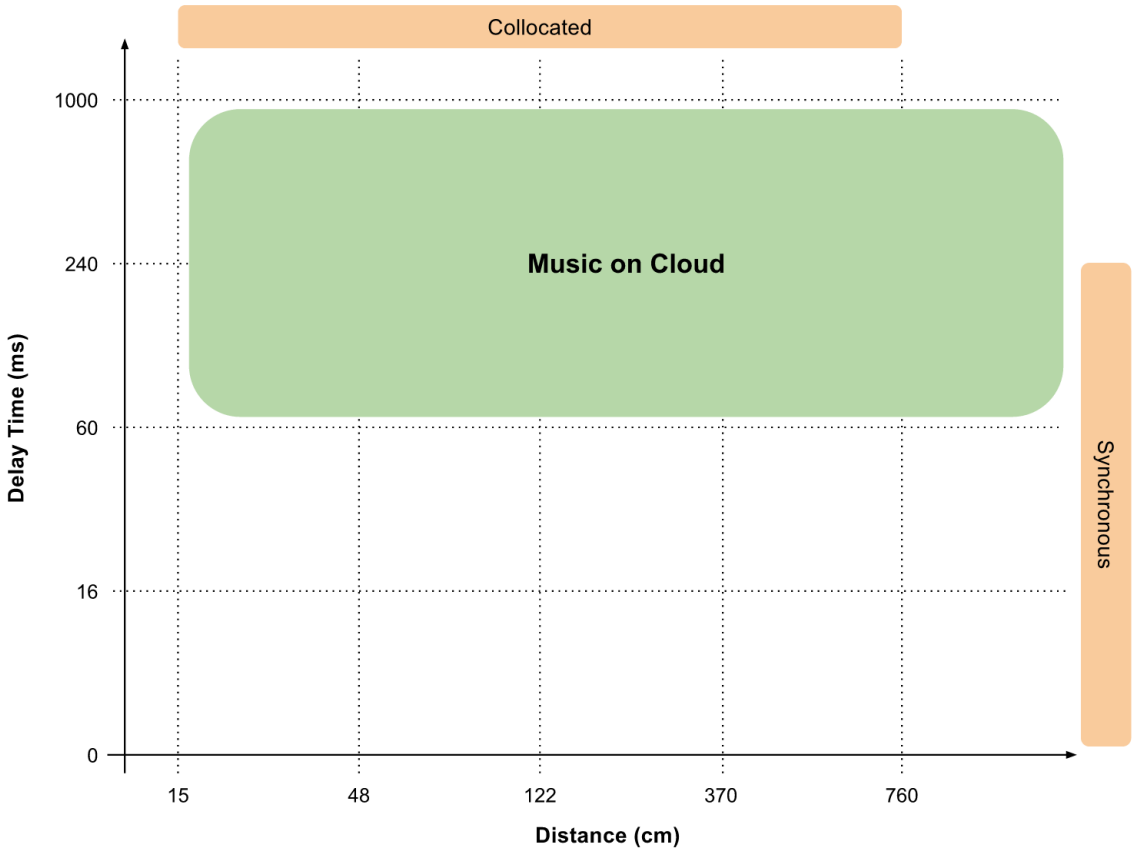

Figure 4.8: Collaboration model of Music on Cloud

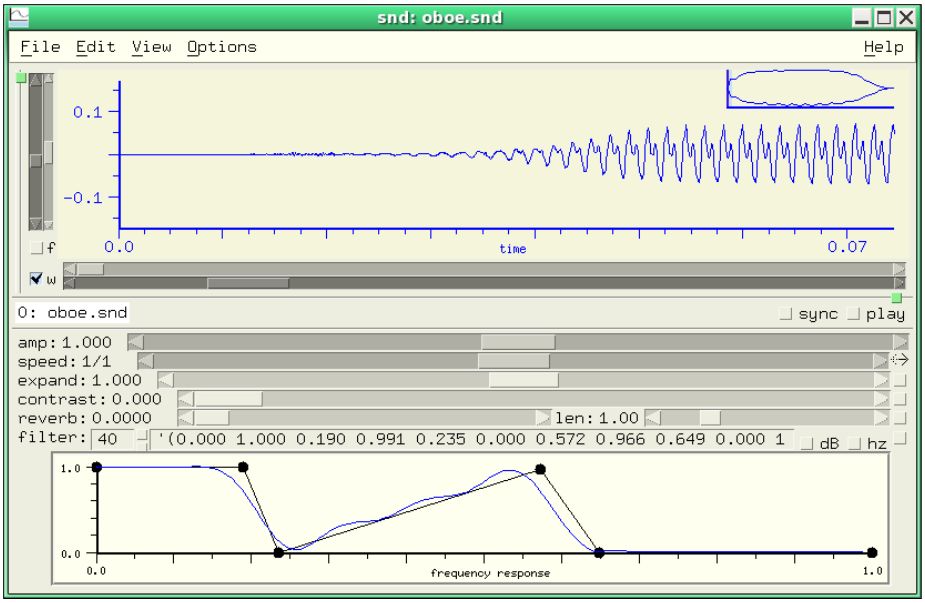

Figure 4.9: SND

a terminal window. Considering most of modern audio programming platforms are designed for the real-time use cases, it is often extremely challenging to catch errors lie within few milliseconds from somewhere in an audio stream. Debugging tools like GDB or LLDB are helpful in finding those bugs to some extent, but setting up breakpoints and stepping through will not work well if you have to repeat hundred-thousands times by hand [37, 102]. To that end, being able to take a precise snapshot of the audio stream from a specific time frame is immensely useful in audio debugging.

Canopy is a web-based debugging tool for Web Audio API code. You can write code and render it to an audio buffer which can be visually inspected. It also analyzes the code and draws a diagram of audio graph to display how everything is connected. Its original inspiration is SND sound editor from CCRMA [94]. SND's rich visualization and graphing feature allows us to understand the sound better. However,

Canopy is centered around the code editor, whereas SND offers on-screen audio editing. Canopy's goal is to verify the code and the sonic outcome, not to manipulate the audio content.

4.4.1 Key Features

The visually rich user interface makes Canopy a versatile educational tool for computer music programming. One can understand the code and how it is interpreted into the digital audio by looking into waveform. The audio graph visualization helps understand how the synthesis algorithm works. Also the code editor in Canopy is equipped with auto completion for Web Audio API, so it makes writing code much easier for the beginners. Thanks to its GitHub Gist integration, code snippets can be shared by simply exchanging the URL to them [1].

4.4.2 Technical Effort and Future Work

A significant amount of engineering effort was invested in making the user experience smooth and perceptually seamless. The waveform and audio graph inspectors, in particular, offer fluid zooming and panning comparable to these functions in native audio editing software.

The overall structure was built on top of a Polymer framework. All the distinctive parts in the application such as waveform inspector, audio graph inspector, code editor, and gist loader are cleanly compartmentalized and it makes the cost of maintenance considerably low. Eventually these modules will rolled out to the MUI

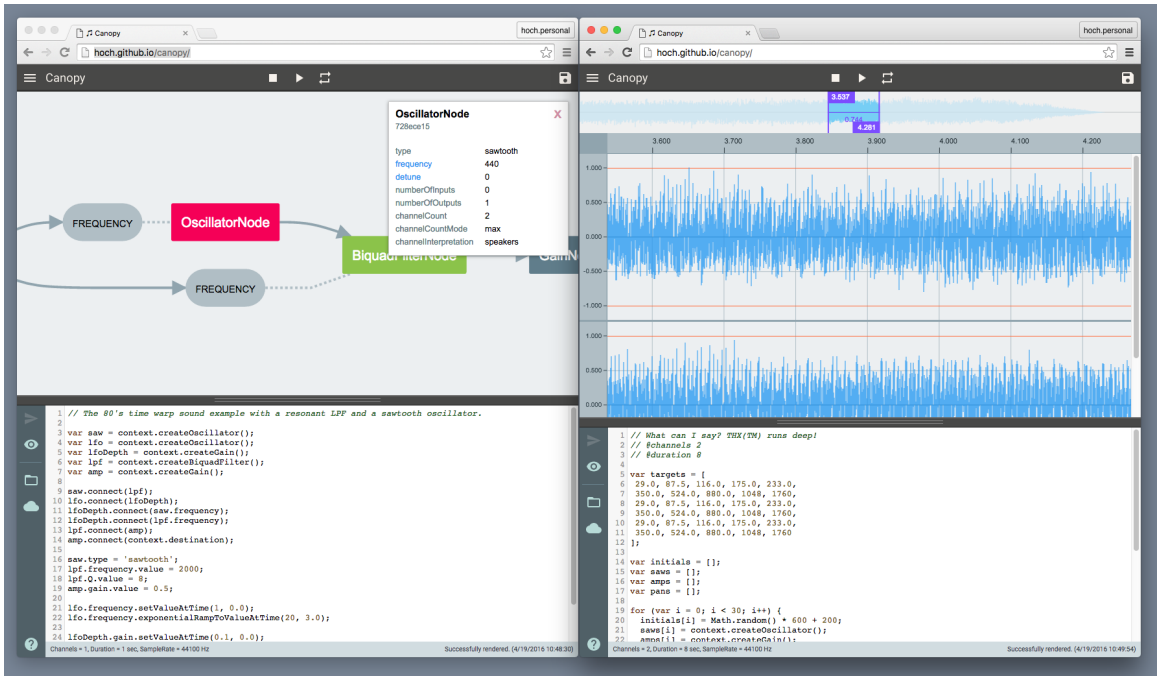

Figure 4.10: Graph view and waveform view in Canopy

element collection, so any developer can add these features in their application with the minimum effort.

The GitHub gist integration has been useful, but it does not support the collaborative code editing. This design was intentional because I believed the version control is more important than the real-time collaboration when it comes to writing code. However, the collaborative code editing would be a very interesting exploration as well. The services like JSFiddle or JS Bin have became the norm of code sharing and collaborative debugging on the Web and Canopy can be the audio counterpart of JS debugging [68, 67].

From its inception, there have been numerous iterations through the constant feedback from Chromium Web Audio API team to make Canopy better. Unlike previous examples, Canopy is more than a prototype; it actually has a real-world use cases such as testing, verification and bug reporting. At the time of this writing, Canopy reached its first stable release. (v1.0.0) It is optimized and stabilized for the production purpose.

4.5 Conclusion

In this chapter, I described four contrasting case studies that use web music technology through the WAAX library; GridFlux was a polished showcase of web music applications comparable to the conventional rhythm workstation. Envvy was an experimental attempt to introduce a new method of collaborative musicking with the notion of a ubiquitous metronome. Music on Cloud was a proof-of-concept of using

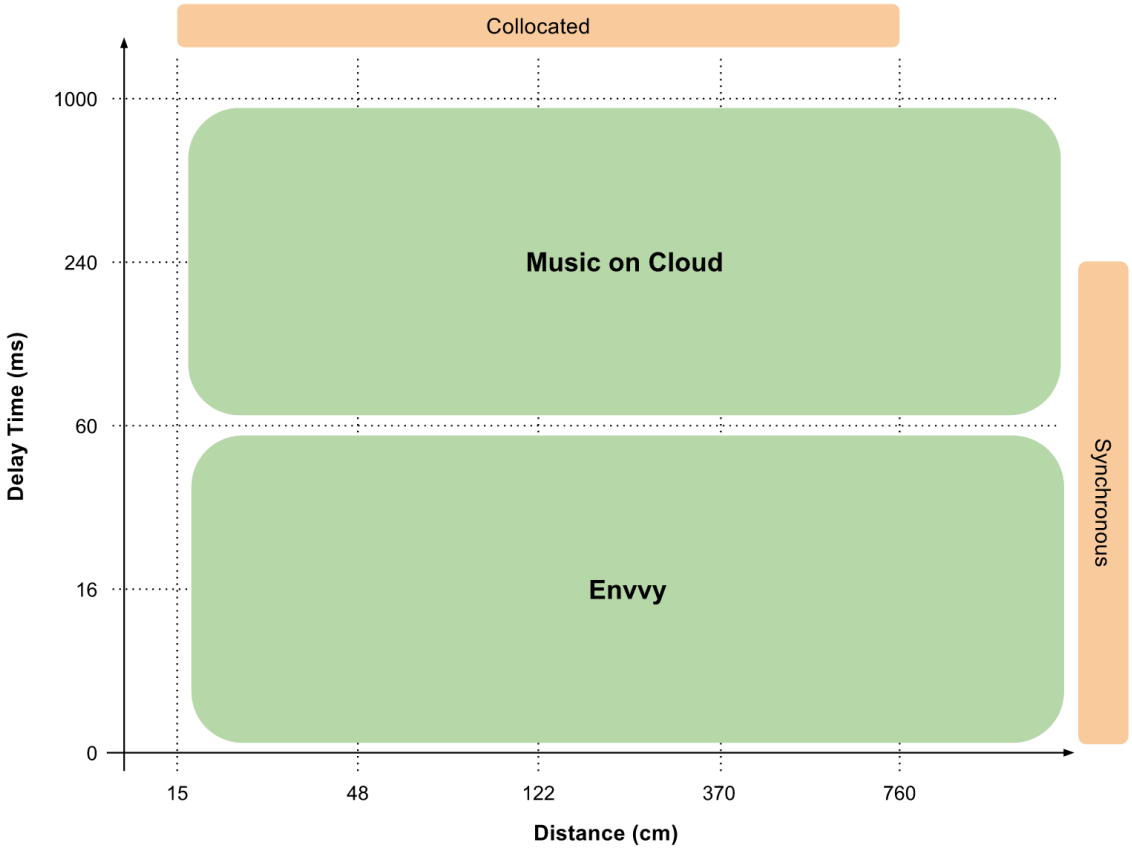

Figure 4.11: Combined collaboration model of Envvy and Music on Cloud

a general-purpose infrastructure for collaborative musicking. Lastly, the report of Musicking on the Web workshop showed a pedagogical use case of the web music technology.

Throughout this chapter, the collaboration model has been used only for analyzing case studies. Conversely we can use the model to design a new collaboration profile. For example, the profile from Envvy and Music on Cloud can be merged into one. Combining them allows us to cover the entire model space and it opens the limitless possibility of how we work together.

WAAX was started as an enthusiastic exploration rather than a software product which must be designed with a clear target audience in mind. However, I believe WAAX successfully proved its efficacy by saving a significant amount of effort and time. Furthermore, numerous lessons from these projects were reflected back to the library making it more solid and feature-rich. MUI elements in particular transformed the way of creating web music application by making the development process fast while lowering the maintenance cost at the same time.

Lastly, future research must investigate collecting real-world usage data. This alone can be a worthy topic for research areas such as HCI or NIME:

- What constitutes the successful musicking?
- How do we measure the success of collaboration?
- Which UI or model works better for a certain style of musical task?
- Is there any rule of thumb in designing a collaboration model?

There are so many unexplored aspects. This thesis serves as a mere starting point of a much larger chapter.

Chapter 5

Conclusions

*"Is it possible to **collaborate** for the music creation using the Web?"*

The ultimate question proposed in this thesis has been explored, until now with a variety of ideas, empirical data, and discussion. Recent advancement of the web ecosystem has placed several unconnected dots in the ether including a cutting-edge web music API, a faster browser, and better networking infrastructure. The goal of this final chapter is to connect these dots as a constellation in the ever-expanding universe.

5.1 Contributions

This theoretical investigation on music collaboration started from Small's concept of 'Musicking'. To figure out what constitutes the musical collaboration I incorporated

various ideas from CSCW (Computer-Supported Cooperative Work) and computer music. As a result of the study, chapter two proposes a concise classification model for CMM (computer-mediated musicking) that is comprised of the delay time and the distance between participants. This new apparatus serves two purposes; the analysis apparatus and the design space for CMM.

Chapter three describes the technical/engineering effort in support of musicking on the Web. WAAX project - presently still evolving - aims to bridge the gap between both ends of the spectrum; the audio developers who are generally unfamiliar with web programming, and the web developers who do not have the computer music background. Beyond this, it offers a comprehensive development package of pre-designed plugins, boilerplate, and build tools to enhance the productivity.

To demonstrate the efficacy of the new classification model and WAAX library, the chapter four presents a series of case studies. Each case report details its design and implementation followed by the analytic modeling of musicking. GridFlux shows the potential of web-based music application compared to the native counterpart (C++ and equivalent). Envvy realizes the idea of instant collaboration and exemplifies how music UI can be redesigned with the web technology. Music on Cloud is a prototype that supports real-time collaboration built on top of the general purpose hosting service. Unlike previous cases, Canopy is a polished web audio tool that is designed toward a well-defined goal; musicking by sharing codes.

5.2 Challenge remains

Despite promising signs that suggest multiple future directions, there remain issues yet to be addressed. At the time of writing, the most critical missing link is the lack of a real-world user metric. This absence is primarily due to the fact that no attempts to create web music software have yet succeeded to reach a critical mass in terms of users. Without a concrete and sizable user base, data collection and analysis is pointless. The actionable analysis on user metric is the key to another important question; the definition of successful musicking and the evaluation of collaborative creativity. The topic is beyond the scope of this thesis, but it certainly needs to be pursued as future study. The lesson from this inquiry will enhance the collaboration model thus help us to build a better CMM system.

Another missed opportunity was to use the classification model to design a style of musicking by varying parameters. Better yet, different musicking styles can be mixed together to cover separate areas in the model. The chapter 4 discusses the possibility but did not reached the realization of the idea. Along with the user metric data collected from the real world, this new design process will make the musicking more productive and creative.

In addition, working with the Web brings many challenges. The complexity of the modern web platform is already beyond the imagination and it touches the life of billions of people. Thus the progress is a very conservative due to a variety of reasons; backward compatibility, security, privacy and performance. In that sense, using the Web as an experimental sandbox might place us at a peculiar position where the

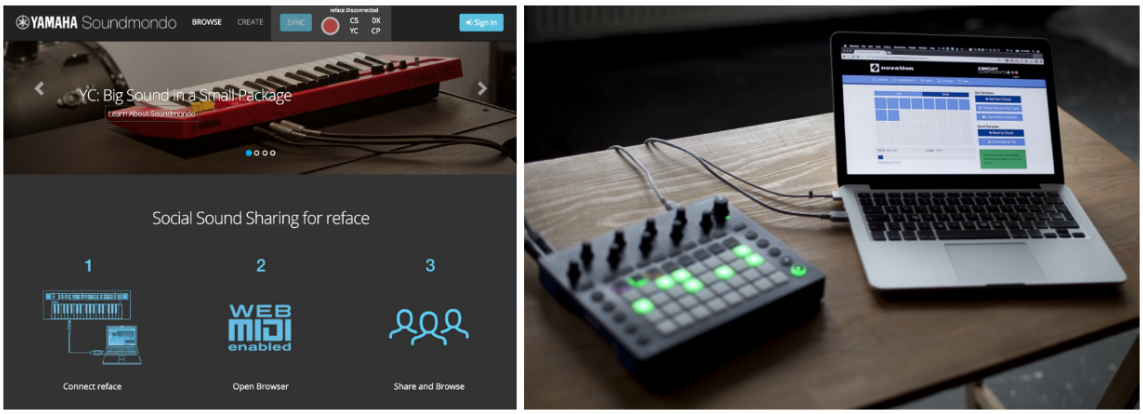

Figure 5.1: Yamaha SoundMondo and Novation Circuit with Web MIDI API [97, 22]

radical progress is impossible. Using JavaScript to extend the built-in functionality somewhat mitigates the issue, but it makes impossible to solve certain problems that needs to addressed by fixing the under the hood.

5.3 Future of Musicking on the Web

As described in the chapter 3, the most exciting addition to the web application technology will be Service Worker and AudioWorklet. The Service Worker and the notion of Progressive Web App allow us to build an application that goes beyond an HTML page. With AudioWorklet, the sample-level manipulation of audio stream is possible within Web Audio API. They enable the web developers to compete with native or mobile application in the near future.

From a long-term perspective, we need to pay attention on the movement of collaborative musicking on the Web. SoundMondo and Circuit are compelling examples

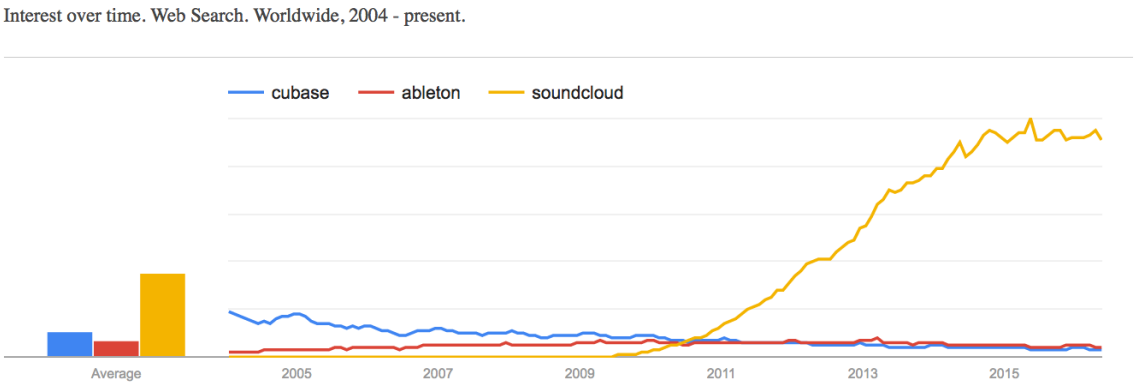

Figure 5.2: Google Search Trend between 2004 - 2016: Cubase, Ableton and SoundCloud

of the industry-driven effort to build a social platform for music making [97, 22]. The ultimate accessibility of the web browser is changing how we share the material for music making. When exchanging musical ideas through the Web becomes the norm, having collaborative tools that can directly work on those materials without leaving the browser will become inevitable.

Where is the world of music production going? There are two clear trends that reflect the future of computer-mediated musicking. Firstly, there is a notable decline in traditional computer music software. Secondly, concurrent to the increased affordability and accessibility of music tools is an explosive desire and need for sharing user-created music content explodes. The professional music application is rapidly losing the battle over the casual and mobile one. If the paradigm shift to the mobile was all about offering optimum user experience to an individual user, the next shift to the web platform will be about sharing the content or the process of musicking itself.

This trend only will accelerate as the infrastructure for media sharing or cloud-computing are rapidly advancing. In the near future, I am expecting to see a real-time communication service that offers the average latency of below 60ms, where the real-time coordination between remote collaborators becomes sensible. Although the data being shared will be mostly meta data rather than actual audio streams, I believe the quality of CMM will be drastically improved and producing a musical composition together in a web application will not be a dream anymore.

Appendix A

"Musicking on the Web" Workshop: Web Music in the classroom

The "Musicking on the Web' workshop was a 2-day technical workshop planned to promote the adoption of the web music technology (i.e. Web Audio/MIDI API) along with the best practice for the design of web-based music application. It was hosted at Google Campus Seoul, recently established by Google as a startup incubator at Seoul [43]. The event was comprised of a series of lecture, hackathon and demo session around the web music technology.

Several professors in South Korea informally asked me to give the introduction of web music technology and the request initiated the planning of the event. In the early discussion with Campus Seoul, they wanted to see the participation from the industry

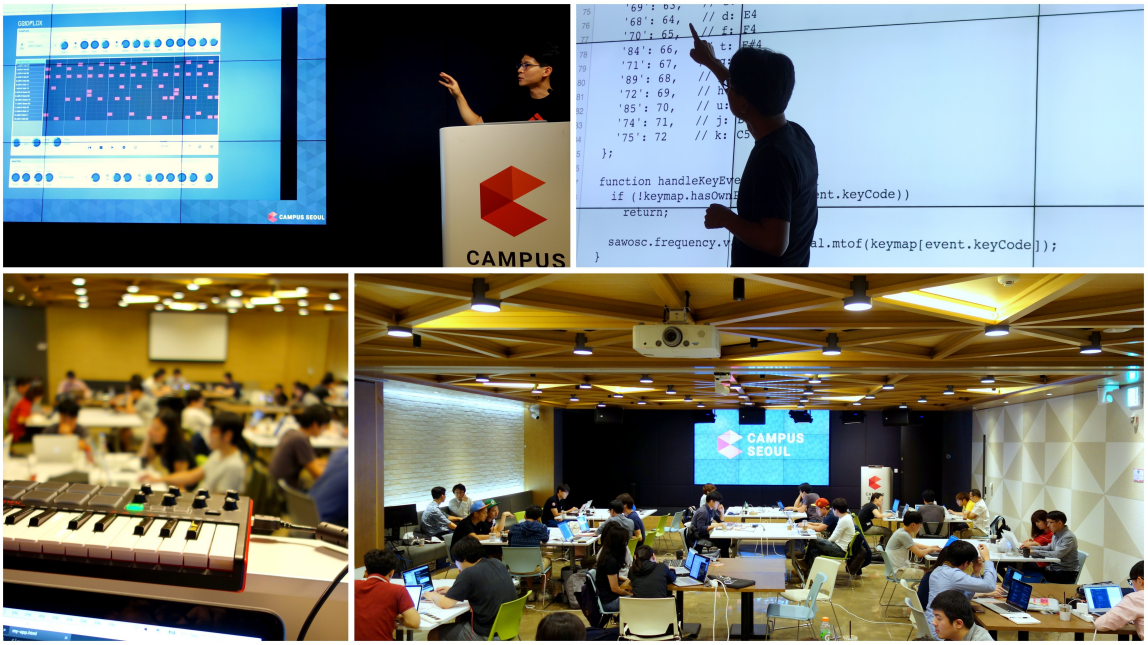

Figure A.1: Musicking on the Web Workshop at Google Campus Seoul

as well (mostly from startups) and it seemed to be the right balance to produce the interesting outcome. To that end, I handpicked 20 applicants from the academia and the other 20 from the industry. All in all, 66 people applied for the workshop but 40 participants were accepted based on the interest and the technical skill level.

A.0.1 Discussion

Slides and example codes were very well-received by the participants [18]. This is the most important accomplishment in this project because the majority of material can be reused for the educational event for the web music technology. Along with the technical aspects, I tried to emphasize the importance of collaborative model and the high-level design. Considering most of web music hackathon events focuses on

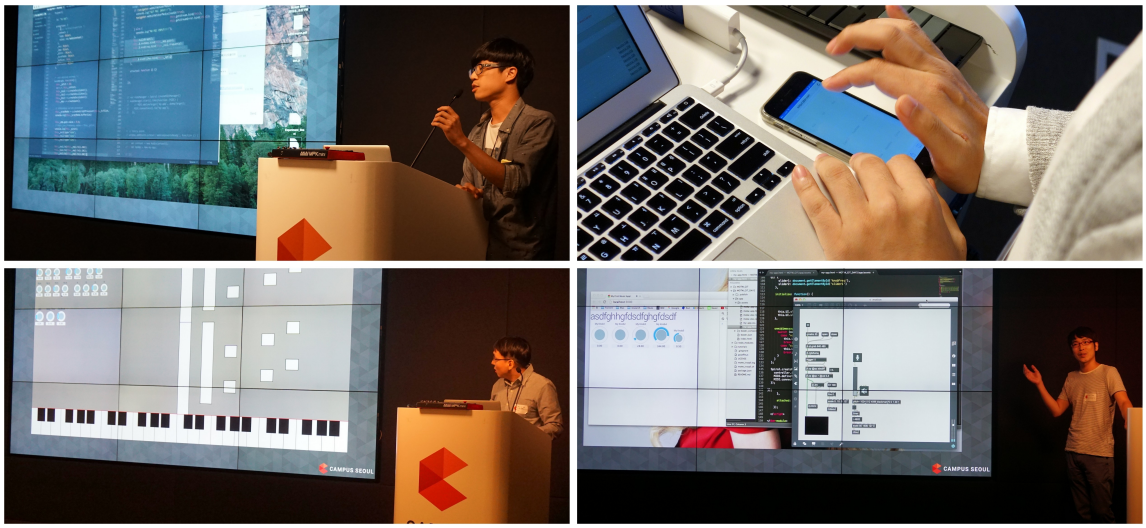

Figure A.2: Various projects from participants

creating small demos, this is the biggest distinction between this event than others.

After two days of intensive lecture and code lab, 4 final projects were presented. The scope of presented projects spans from the pitch recognition to the mobile drum sampler. My expectation for final projects were fairly low and the majority of participants could not finish their projects in time. It is because most of participants were not familiar with either the theoretical topic (people from industry) or the development set up (people from academia). However, participants who were able to finish and present their projects showed the impressive outcome.

The followings are excerpts from the collected feedback from participants (translated from Korean):

> *"Although it was somewhat difficult, the response from students was exceptional. This event made me believe that web music technology has huge*

potential and there is a lot to explore. The interest around it is enormously high, so please hold another workshop next year!" - Prof. Woon Seung Yeo (Ewha Women's University)

"I felt really sorry about the event only being two days. Throughout the workshop we found there are so many exciting ideas flowing around in the class. It would have been better to have more discussion with you and other participants. I would love to hear your vision on the future of the web music technology and what is exciting ahead." - Sang Hee Won (Seoul National University)

"I joined the event out of curiosity, but now I became a firm believer of WebAudio. I started working with my friends to find something exciting with WebAudio before someone else does." - Yoon Chang Han (Seoul National University)

"I had a hard time working on the team project because of my coding skill. Personally I learned a lot about computer-generated music, the web development, and more importantly, new ideas" - Yoon Ah Yang (Ewha Women's University)

"Currently preparing a startup company, but I majored Cello before my life as an engineer. This whole workshop really resonated with me and showed me what I can do with music and the Web." - Hyuk Chang (Polariant)

The brevity of the two day event was a common source of complaint from participants. Some people were disappointed because they were expecting to see a complete back-end or application service to build a business. Also working on the intersection of computer music and the Web is never an easy task and it was technically challenging for people who are not proficient to either the web development or computer music.

Lastly, my primary objective was to test the web music technology in a classroom setting. The result was very promising. Regardless of the technical skill level, the potential of the web platform in musicking generated an enthusiastic and active response.

Bibliography

[1] *About gists*. 2016. URL: https://help.github.com/articles/about-gists/ (visited on 05/01/2016).

[2] *Active X*. 2016. URL: https://en.wikipedia.org/wiki/ActiveX (visited on 05/01/2016).

[3] *Adobe has an epically abysmal security record*. 2013. URL: http://money.cnn.com/2013/10/08/technology/security/adobe-security (visited on 05/01/2016).

[4] Robert Nyman Alon Zakai. *Gap between asm.js and native performance gets even narrower with float32 optimizations*. 2013. URL: https://hacks.mozilla.org/2013/12/gap-between-asm-js-and-native-performance-gets-even-narrower-with-float32-optimizations (visited on 05/01/2016).

[5] *Angular*. 2016. URL: https://angularjs.org/ (visited on 05/01/2016).

[6] *Are we fast yet?* 2014. URL: https://arewefastyet.com (visited on 05/01/2016).

[7] *Audio Data API*. 2013. URL: https://wiki.mozilla.org/Audio_Data_API (visited on 05/01/2016).

[8] *Audio Workers*. 2015. URL: `https://github.com/WebAudio/web-audio-api/issues/113` (visited on 05/01/2016).

[9] *Audio Working Group Charter Overview*. 2015. URL: `https://www.w3.org/2011/audio/charter/Overview.html` (visited on 05/01/2016).

[10] *AudioWorkletNode examples*. 2015. URL: `https://github.com/WebAudio/web-audio-api/issues/%7B776,777,778,779%7D` (visited on 05/01/2016).

[11] Chromium authors. *Sandbox FAQ*. 2015. URL: `https://www.chromium.org/developers/design-documents/sandbox/Sandbox-FAQ` (visited on 05/01/2016).

[12] WebAssembly authors. *WebAssembly FAQ*. 2016. URL: `https://github.com/WebAssembly/design/blob/master/FAQ.md#is-webassembly-trying-to-replace-javascript` (visited on 05/01/2016).

[13] Álvaro Barbosa. "Displaced soundscapes: A survey of network systems for music and sonic art creation". In: *Leonardo Music Journal* 13 (2003), pp. 53–59.

[14] *Blink*. 2016. URL: `http://www.chromium.org/blink` (visited on 05/01/2016).

[15] *Browserify*. 2016. URL: `http://browserify.org/` (visited on 05/01/2016).

[16] Linda Candy and Ernest Edmonds. "Modeling co-creativity in art and technology". In: *Proceedings of the 4th conference on Creativity & cognition*. ACM. 2002, pp. 134–141.

[17] Chris Chafe, Juan-Pablo Cáceres, and Michael Gurevich. "Effect of temporal separation on synchronization in rhythmic performance". In: *Perception* 39.7 (2010), pp. 982–992.

[18] Hongchan Choi. *MOTW 2015 Workshop Page*. 2015. URL: `http://hoch.github.io/motw-2015` (visited on 05/01/2016).

[19] Hongchan Choi. *WAAX API Reference*. 2015. URL: `http://hoch.github.io/WAAX/reference/1.0.0-alpha3/WX.html` (visited on 05/01/2016).

[20] Hongchan Choi and Jonathan Berger. "WAAX: Web Audio API eXtension." In: *NIME*. 2013, pp. 499–502.

[21] *Chrome V8*. 2016. URL: `https://developers.google.com/v8` (visited on 05/01/2016).

[22] *Circuit, The inspirational grid-based groove box*. 2016. URL: `https://us.novationmusic.com/circuit/circuit` (visited on 05/01/2016).

[23] *Consortium Process Document*. 2015. URL: `http://www.w3.org/Consortium/Process` (visited on 05/01/2016).

[24] Dana Cowley. *Strategy Game WebGL demo available now*. 2014. URL: `https://www.unrealengine.com/blog/strategy-game-webgl-demo-available-now` (visited on 05/01/2016).

[25] Luke Dahl. "Designing New Musical Interfaces as Research". In: *Leonardo* (2015).

[26] *Data-binding Revolutions with Object.observe - HTML5 Rocks*. 2014. URL: `http://www.html5rocks.com/en/tutorials/es7/observe/` (visited on 05/01/2016).

[27] Jon Dattorro. "Effect design, part 2: Delay line modulation and chorus". In: *Journal of the Audio engineering Society* 45.10 (1997), pp. 764–788.

[28] Paul Dourish and Victoria Bellotti. "Awareness and coordination in shared workspaces". In: *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. ACM. 1992, pp. 107–114.

[29] Jonas Echterhoff. *On the future of web publishing in Unity*. 2014. URL: `http://blogs.unity3d.com/2014/04/29/on-the-future-of-web-publishing-in-unity/` (visited on 05/01/2016).

[30] *ECMAScript*. 2015. URL: `http://www.ecmascript.org/` (visited on 05/01/2016).

[31] Brendan Eich. *From ASM.JS to WebAssembly*. 2015. URL: `https://brendaneich.com/2015/06/from-asm-js-to-webassembly/` (visited on 05/01/2016).

[32] *Ember.js - A framework for creating ambitious web applications*. 2011. URL: `http://emberjs.com/` (visited on 05/01/2016).

[33] *Emscripten*. 2015. URL: `http://kripken.github.io/emscripten-site/index.html` (visited on 05/01/2016).

[34] Daniel Fallman. "The interaction design research triangle of design practice, design studies, and design exploration". In: *Design Issues* 24.3 (2008), pp. 4–18.

[35] Robin Fencott and Nick Bryan-Kinns. "Computer musicking: HCI, CSCW and collaborative digital musical interaction". In: *Music and Human-Computer Interaction*. Springer, 2013, pp. 189–205.

[36] Matt Gaunt. *Introduction to Service Worker*. 2016. URL: `http://www.html5rocks.com/en/tutorials/service-worker/introduction/` (visited on 05/01/2016).

[37] *GDB: The GNU project debugger*. 2016. URL: `https://www.gnu.org/software/gdb/` (visited on 05/01/2016).

[38] *Gecko*. 2016. URL: `https://developer.mozilla.org/en-US/docs/Mozilla/Gecko` (visited on 05/01/2016).

[39] *GEIST, Next-generation sampling drum machine*. 2016. URL: `https://www.fxpansion.com/products/geist/` (visited on 05/01/2016).

[40] *GIF89a Specification*. 1989. URL: `https://www.w3.org/Graphics/GIF/spec-gif89a.txt` (visited on 05/01/2016).

[41] *Git*. 2015. URL: `https://git-scm.com/about` (visited on 05/01/2016).

[42] *Google announces price cut for all Compute Engine instances, Google Drive has passed 240M active users*. 2014. URL: `http://thenextweb.com/google/2014/10/01/google-announces-10-price-cut-compute-engine-instances-google-drive-passed-240m-active-users/` (visited on 05/01/2016).

[43] *Google Campus Seoul*. 2016. URL: `https://www.campus.co/seoul/en` (visited on 05/01/2016).

[44] *Google Realtime API*. 2016. URL: `https://developers.google.com/google-apps/realtime/overview` (visited on 05/01/2016).

[45] W3C Web Application Working Group. *Introduction to Web Components*. 2015. URL: `https://www.w3.org/TR/components-intro/` (visited on 05/01/2016).

[46] *Grunt - JavaScript Task Runner*. 2016. URL: `http://gruntjs.com/` (visited on 05/01/2016).

[47] *Gulp - The Streaming Build System*. 2016. URL: `http://gulpjs.com/` (visited on 05/01/2016).

[48] Michael Gurevich. "JamSpace: designing a collaborative networked music space for novices". In: *Proceedings of the 2006 conference on New interfaces for musical expression*. IRCAM. 2006, pp. 118–123.

[49] Carl Gutwin. "The effects of network delays on group work in real-time groupware". In: *ECSCW 2001*. Springer. 2001, pp. 299–318.

[50] Carl Gutwin and Saul Greenberg. "The effects of workspace awareness support on the usability of real-time distributed groupware". In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 6.3 (1999), pp. 243–281.

[51] Edward Twitchell Hall. "The hidden dimension ." In: (1966).

[52] Brian Hayes. "Commun". In: *ACM, Cloud Computing* 51.7 (2008), pp. 9–11.

[53] Marcus Hellberg. *Web Components in production use: are we there yet?* 2016. URL: https://vaadin.com/blog/-/blogs/web-components-in-production-use-are-we-there-yet- (visited on 05/01/2016).

[54] *History of ActionScript*. 2016. URL: https://en.wikipedia.org/wiki/ActionScript#ActionScript_3.0 (visited on 05/01/2016).

[55] *History of ActionScript*. 2016. URL: http://audiotool.com (visited on 05/01/2016).

[56] *History of Flash*. 2016. URL: https://en.wikipedia.org/wiki/Adobe_Flash_Professional (visited on 05/01/2016).

[57] Jim Hollan and Scott Stornetta. "Beyond being there". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 1992, pp. 119–125.

[58] *HTML Canvas 2D Context*. 2015. URL: https://www.w3.org/TR/2dcontext/ (visited on 05/01/2016).

[59] *HTML5*. 2013. URL: https://en.wikipedia.org/wiki/HTML5#History (visited on 05/01/2016).

[60] *HTML5*. 2013. URL: http://www.browserscope.org/timeline (visited on 05/01/2016).

[61] *HTML5 is a W3C Recommendation*. 2014. URL: https://www.w3.org/blog/news/archives/4167 (visited on 05/01/2016).

[62] Paul Hudak and Jonathan Berger. "A model of performance, interaction, and improvisation". In: *ICMC*. Citeseer. 1995, pp. 1–8.

[63] *Immediately-Invoked Function Expression*. 2015. URL: `https://en.wikipedia.org/wiki/Immediately-invoked_function_expression` (visited on 05/01/2016).

[64] *Institute for Telecommunication Science, Federal Standard 1037C*. 1996. URL: `http://www.its.bldrdoc.gov/fs-1037/dir-024/_3492.htm` (visited on 05/01/2016).

[65] *Interview with the father of the MP3*. 2015. URL: `http://www.internethistorypodcast.com/2015/07/on-the-20th-birthday-of-the-mp3-an-interview-with-the-father-of-the-mp3-karlheinz-brandenburg` (visited on 05/01/2016).

[66] *JavaScriptCore*. 2014. URL: `http://trac.webkit.org/wiki/JavaScriptCore` (visited on 05/01/2016).

[67] *JS Bin - Collaborative JavaScript Debugging*. 2016. URL: `http://jsbin.com/` (visited on 05/01/2016).

[68] *JS Fiddle*. 2016. URL: `https://jsfiddle.net/` (visited on 05/01/2016).

[69] *Khronos Releases Final WebGL 1.0 Specification*. 2011. URL: `https://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification` (visited on 05/01/2016).

[70] Adam Klein. *An update on Object.observe*. 2015. URL: `https://esdiscuss.org/topic/an-update-on-object-observe` (visited on 05/01/2016).

[71] Alyson La. *Language Trends on GitHub*. 2015. URL: https://github.com/blog/2047-language-trends-on-github (visited on 05/01/2016).

[72] *LiveAudio Script Syntax*. 1996. URL: http://www.uv.es/jordi/v3/html/live/audio.html (visited on 05/01/2016).

[73] Kurt Luther et al. "Why it works (when it works): Success factors in online creative collaboration". In: *Proceedings of the 16th ACM international conference on Supporting group work*. ACM. 2010, pp. 1–10.

[74] *Maschine, Production System*. 2016. URL: http://www.native-instruments.com/en/products/maschine/production-systems/maschine/ (visited on 05/01/2016).

[75] *Maschine, Production System*. 2016. URL: https://allihoopa.com/ (visited on 05/01/2016).

[76] Ali Mesbah and Mukul R Prasad. "Automated cross-browser compatibility testing". In: *Proceedings of the 33rd International Conference on Software Engineering*. ACM. 2011, pp. 561–570.

[77] *MOSAIC Versions*. 2016. URL: http://www.ncsa.illinois.edu/enabling/mosaic/versions (visited on 05/01/2016).

[78] Martin Muzatko and et. al. *HTML5 Cross Browser Polyfills*. 2016. URL: https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills (visited on 05/01/2016).

[79] *NCSA MOSAIC*. 2016. URL: `http://www.ncsa.illinois.edu/enabling/mosaic` (visited on 05/01/2016).

[80] *Netscape Navigator 3 release note*. 1996. URL: `ftp://ftp.polito.it/people/bertola/netscape/3.0/README.txt` (visited on 05/01/2016).

[81] *Node.js*. 2015. URL: `https://nodejs.org/en/` (visited on 05/01/2016).

[82] *NPAPI*. 2016. URL: `https://en.wikipedia.org/wiki/NPAPI` (visited on 05/01/2016).

[83] *Polymer Project*. 2016. URL: `https://www.polymer-project.org/1.0/` (visited on 05/01/2016).

[84] *Progressive Web Apps*. 2016. URL: `https://developers.google.com/web/progressive-web-apps?hl=en#learnmore` (visited on 05/01/2016).

[85] *React - A JavaScript library for building user interfaces*. 2016. URL: `https://facebook.github.io/react/` (visited on 05/01/2016).

[86] *Realtime Hosted Service Latency Stats*. 2016. URL: `http://www.leggetter.co.uk/real-time-web-technologies-guide/realtime-hosted-service-latency` (visited on 05/01/2016).

[87] Francois Remy. *The Extensible Web Manifesto*. 2013. URL: `https://www.w3.org/community/nextweb/2013/06/11/the-extensible-web-manifesto/` (visited on 05/01/2016).

[88] Jon Rimmer. *Are We Componentized Yet?* 2015. URL: `http://jonrimmer.github.io/are-we-componentized-yet/` (visited on 05/01/2016).

[89] Charles Roberts, Graham Wakefield, and Matthew Wright. "The Web Browser As Synthesizer And Interface." In: *NIME*. 2013, pp. 313–318.

[90] Tom Rodden. "A survey of CSCW systems". In: *interacting with Computers* 3.3 (1991), pp. 319–353.

[91] R Keith Sawyer et al. *Group creativity: Music, theater, collaboration*. Psychology Press, 2014.

[92] *Service Workers Specification*. 2016. URL: `https://slightlyoff.github.io/ServiceWorker/spec/service_worker/` (visited on 05/01/2016).

[93] Christopher Small. *Musicking: The meanings of performing and listening*. Wesleyan University Press, 2011.

[94] *SND*. 2016. URL: `https://ccrma.stanford.edu/software/snd/snd/snd.html` (visited on 05/01/2016).

[95] *Software Synthesizer*. 2016. URL: `https://en.wikipedia.org/wiki/Software_synthesizer#Microsoft_GS_Wavetable_SW_Synth` (visited on 05/01/2016).

[96] *Sound on Sound: Notes on QuickTime v.2.0*. 1994. URL: `http://www.soundonsound.com/sos/1994_articles/aug94/applenotes.html` (visited on 05/01/2016).

[97] *Soundmondo - Social Sound Sharing for Reface*. 2016. URL: `https://soundmondo.yamahasynth.com/` (visited on 05/01/2016).

[98] *SpiderMonkey*. 2016. URL: `https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey` (visited on 05/01/2016).

[99] *TAG issue: Subclassing / Node composition*. 2016. URL: https://github.com/WebAudio/web-audio-api/issues/251 (visited on 05/01/2016).

[100] *TB-303 Acid Flashback - Roland U.S. Blog*. 2013. URL: http://www.rolandus.com/blog/2013/03/28/tb-303-acid-flashback/ (visited on 05/01/2016).

[101] *The birth of the web*. 2016. URL: http://home.cern/topics/birth-web (visited on 05/01/2016).

[102] *The LLDB Debugger*. 2016. URL: http://lldb.llvm.org/ (visited on 05/01/2016).

[103] *The Open Web Platform*. 2015. URL: https://www.w3.org/wiki/Open_Web_Platform (visited on 05/01/2016).

[104] *Thoughts on Flash*. 2013. URL: http://www.apple.com/hotnews/thoughts-on-flash (visited on 05/01/2016).

[105] *W3C SVG History*. 2004. URL: https://www.w3.org/Graphics/SVG/History (visited on 05/01/2016).

[106] *W3C Technical Report Development Process*. 2005. URL: https://www.w3.org/2005/10/Process-20051014/tr.html#rec-advance (visited on 05/01/2016).

[107] *Web Audio API*. 2010. URL: http://webaudio.github.io/web-audio-api (visited on 05/01/2016).

[108] *Web Audio API Specification Issue Tracker*. 2016. URL: https://github.com/WebAudio/web-audio-api/issues (visited on 05/01/2016).

[109] *Web MIDI API*. 2013. URL: `http://webaudio.github.io/web-midi-api` (visited on 05/01/2016).

[110] *WebKit*. 2016. URL: `https://webkit.org/` (visited on 05/01/2016).

[111] *WebKit Changeset 58269*. 2010. URL: `https://trac.webkit.org/changeset/58269` (visited on 05/01/2016).

[112] *WebPack*. 2016. URL: `https://webpack.github.io/` (visited on 05/01/2016).

[113] *What is HyperText*. 1999. URL: `https://www.w3.org/WhatIs.html` (visited on 05/01/2016).

[114] Jackie Wiggins. "Compositional process in music". In: *International handbook of research in arts education*. Springer, 2007, pp. 453–476.

[115] Lonce Wyse and Srikumar Subramanian. "The viability of the web browser as a computer music platform". In: *Computer Music Journal* 37.4 (2013), pp. 10–23.